aws

用户指南

AWS Proton

AWS Proton: 用户指南

Table of Contents

什么是 AWS Proton？

AWS Proton 为：

- 无服务器和基于容器的应用程序的自动化基础设施即代码预置和部署

  AWS Proton 服务是一个双重自动化框架。作为管理员，您创建版本控制的服务模板，从而为无服务器和基于容器的应用程序定义标准化基础设施和部署工具。作为应用程序开发人员，您可以从可用的服务模板 中进行选择，以自动完成应用程序或服务部署。

  AWS Proton 识别所有使用过时模板版本的现有服务实例。作为管理员，您可以请求 AWS Proton 一键升级它们。

- 标准化基础设施

  平台团队可以使用 AWS Proton 和版本控制的基础设施即代码模板。他们可以使用这些模板定义和管理包含架构、基础设施资源和 CI/CD 软件部署管道的标准应用程序堆栈。

- 与 CI/CD 集成的部署

  在开发人员使用 AWS Proton 自助式界面选择服务模板 时，他们将选择标准化应用程序堆栈定义以部署代码。AWS Proton 自动预置资源，配置 CI/CD 管道以及将代码部署到定义的基础设施中。

适用于平台团队的 AWS Proton

作为管理员，您或您的平台团队成员可以创建包含基础设施即代码的环境模板 和服务模板。环境模板定义多个应用程序或资源使用的共享基础设施。服务模板 定义在环境 中部署和维护单个应用程序或微服务所需的基础设施类型。AWS Proton 服务 是服务模板的实例化形式，通常包括多个服务实例 和一个管道。AWS Proton 服务实例 是服务模板 在特定环境 中的实例化形式。您或您的团队中的其他人可以指定哪些环境模板与给定的服务模板 兼容。有关模板的更多信息，请参阅AWS Proton 模板。

您可以在 AWS Proton 中使用以下基础设施即代码提供商：

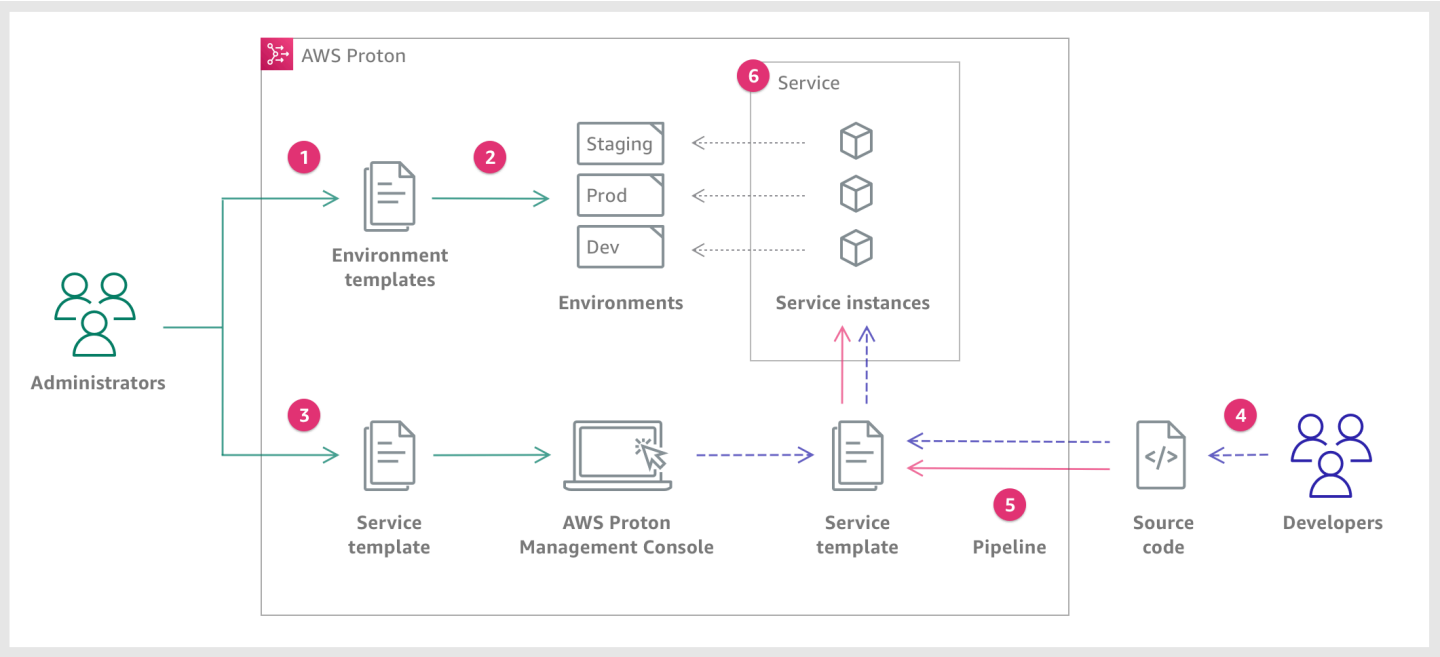
- AWS CloudFormation
- Terraform

适用于开发人员的 AWS Proton

作为应用程序开发人员，您选择一个标准化服务模板，AWS Proton 使用该模板创建一个服务 以在服务实例 中部署和管理应用程序。AWS Proton 服务 是服务模板的实例化形式，通常包括多个服务实例和一个管道。

AWS Proton 工作流

下图是上一段落中讨论的主要 AWS Proton 概念的可视化形式。它还简要概述了简单 AWS Proton 工作流的组成部分。



1 作为管理员，您使用 AWS Proton 创建并注册环境模板，其中定义了共享资源。

2 AWS Proton 根据环境模板部署一个或多个环境。

3 作为管理员，您使用 AWS Proton 创建并注册服务模板，其中定义了相关的基础设施、监控和 CI/CD 资源以及兼容的环境模板。

4 作为开发人员，您选择注册的服务模板并提供指向源代码存储库的链接。

5 AWS Proton 为您的服务实例预置具有 CI/CD 管道的服务。

6 AWS Proton 按照选定的服务模板中的定义，预置和管理运行源代码的服务和服务实例。服务实例是选定服务模板在环境中针对单个管道阶段的实例化形式（例如 Prod）。

设置

完成本节中的任务，以便创建和注册服务和环境模板。您需要使用这些模板在 AWS Proton 中部署环境和服务。

> **Note**
>
> 我们免费提供 AWS Proton。您可以免费创建、注册和维护服务和环境模板。您也可以依靠 AWS Proton 自行管理它自己的操作，例如存储、安全性和部署。您在使用 AWS Proton 时产生的唯一费用如下所示：
>
> - 部署和使用您指示 AWS Proton 为您部署和维护的 AWS Cloud 资源的成本。
> - 保持到代码存储库的 AWS CodeStar 连接的成本。
> - 维护 Amazon S3 存储桶的成本 - 如果您使用存储桶向 AWS Proton 提供输入。如果您切换到使用 the section called “模板捆绑包” 的 Git 存储库的 the section called “模板同步配置”，则可以避免这些成本。

主题

使用 IAM 进行设置

在您注册 AWS 时，将为您的 AWS 账户 自动注册 AWS 中的所有服务，包括 AWS Proton。您只需为使用的服务和资源付费。

> **Note**
>
> 您和您的团队（包括管理员和开发人员）必须位于同一账户中。

注册AWS

如果您还没有 AWS 账户，请完成以下步骤创建一个。

注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，您将接到一通电话，要求您使用电话键盘输入一个验证码。

   当您注册 AWS 账户 时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务和资源。作为安全最佳实践，请 为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

创建 IAM 用户

要创建管理员用户，请选择以下选项之一。

选择一种方法来管理您的管理员	目的	方式	您也可以
在 IAM Identity Center 中（建议）	使用短期凭证访问 AWS。 这符合安全最佳实践。有关最佳实践的信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。	有关说明，请参阅《AWS IAM Identity Center 用户指南》中的入门。	按照《AWS Command Line Interface 用户指南》中的配置 AWS CLI 以使用 AWS IAM Identity Center，配置编程式访问。
在 IAM 中（不推荐使用）	使用长期凭证访问 AWS。	按照《IAM 用户指南》中的创建您的首个 IAM 管理员用户和组的说明操作。	按照《IAM 用户指南》中的管理 IAM 用户的访问密钥，配置编程式访问。

设置 AWS Proton 服务角色

您可能希望为 AWS Proton 解决方案的不同部分创建一些 IAM 角色。您可以使用 IAM 控制台提前创建这些角色，也可以使用 AWS Proton 控制台为您创建这些角色。

创建 AWS Proton 环境角色，以允许 AWS Proton 代表您对其他 AWS 服务（例如 AWS CloudFormation、AWS CodeBuild 以及各种计算和存储服务）进行 API 调用，从而为您预置资源。在环境或其中运行的任何服务实例使用 AWS 托管式预置时，需要具有 AWS 托管式预置角色。在环境或其任何服务实例使用 CodeBuild 预置时，需要具有 CodeBuild 角色。要了解 AWS Proton 环境角色的更多信息，请参阅the section called “IAM 角色”。在创建环境时，您可以使用 AWS Proton 控制台为这两个角色选择现有角色，或者为您创建具有管理权限的角色。

同样，创建 AWS Proton 管道角色 以允许 AWS Proton 代表您对其他服务进行 API 调用，从而为您预置 CI/CD 管道。要了解 AWS Proton 管道角色的更多信息，请参阅the section called “管道服务角色”。有关配置 CI/CD 设置的更多信息，请参阅the section called “设置账户 CI/CD 管道设置”。

> **Note**
>
> 由于我们不知道您在 AWS Proton 模板中定义哪些资源，因此，您使用控制台创建的角色具有广泛的权限，并且可以用作 AWS Proton 管道服务角色和 AWS Proton 服务角色。对于生产部署，我们建议您为 AWS Proton 管道服务角色和 AWS Proton 环境服务角色创建自定义的策略，以将权限范围缩小到部署的特定资源。您可以使用 AWS CLI 或 IAM 创建和自定义这些角色。有关更多信息，请参阅 AWS Proton 的服务角色 和 创建服务：

使用 AWS Proton 进行设置

如果要使用 AWS CLI 运行 AWS Proton API，请验证您是否已安装该 CLI。如果尚未安装，请参阅设置 AWS CLI。

AWS Proton 特定的配置：

- 创建和管理模板：
  - 如果您使用模板同步配置，请建立一个 AWS CodeStar 连接。
  - 否则，请设置一个 Amazon S3 存储桶。
- 预置基础设施：
  - 对于自托管式预置，您必须建立一个 AWS CodeStar 连接。
- （可选）预置管道：

- 对于 AWS 托管式预置和基于 CodeBuild 的预置，设置管道角色。
- 对于自托管式预置，设置一个管道存储库。

有关预置方法的更多信息，请参阅the section called “AWS 托管式预置”。

设置 Amazon S3 存储桶

要设置 S3 存储桶，请按照创建您的第一个 S3 存储桶中的说明设置一个 S3 存储桶。将 AWS Proton 的输入放入存储桶中，AWS Proton 可以在其中检索这些输入。这些输入称为模板捆绑包。您可以在本指南的其他章节中了解模板捆绑包的更多信息。

建立 AWS CodeStar 连接

要将 AWS Proton 连接到存储库，您需要创建一个 AWS CodeStar 连接，以在第三方源代码存储库上进行新提交时激活一个管道。

AWS Proton 使用连接执行以下操作：

- 在存储库源代码上进行新提交时激活服务管道。
- 对基础设施即代码存储库发出拉取请求。
- 每次将提交推送到模板存储库以更改某个模板时，都会创建新的模板次要或主要版本（如果该版本尚不存在）。

您可以使用 CodeConnections 连接到 Bitbucket、GitHub、GitHub Enterprise 和 GitHub Enterprise Server 存储库。有关更多信息，请参阅 AWS CodePipeline 用户指南中的 CodeConnections。

建立一个 CodeStar 连接。

1. 打开 AWS Proton 控制台。
2. 在导航窗格中，选择设置，然后选择存储库连接以转到开发人员工具设置中的连接页面。该页面显示一个连接列表。
3. 选择创建连接，并按照说明进行操作。

设置账户 CI/CD 管道设置

AWS Proton 可以预置 CI/CD 管道，以将应用程序代码部署到服务实例中。管道预置所需的 AWS Proton 设置取决于您为管道选择的预置方法。

AWS 托管式预置和基于 CodeBuild 的预置 - 设置管道角色

通过使用 AWS 托管式预置和 CodeBuild 预置，AWS Proton 可以为您预置管道。因此，AWS Proton 需要具有一个提供预置管道权限的服务角色。这两种预置方法都使用自己的服务角色。这些角色是在所有 AWS Proton 服务管道之间共享的，您可以在账户设置中配置一次这些角色。

使用控制台创建管道服务角色

1. 打开 AWS Proton 控制台。
2. 在导航窗格中，选择设置，然后选择账户设置。
3. 在账户 CI/CD 设置页面中，选择配置。
4. 请执行下列操作之一：
   - 让 AWS Proton 为您创建管道服务角色

     [启用 AWS 托管式管道预置] 在配置账户设置页面上的 AWS 托管式预置管道角色部分中：

     a. 选择新服务角色。

     b. 输入角色的名称，例如 **myProtonPipelineServiceRole**。

     c. 选中该复选框以同意在您的账户中创建具有管理权限的 AWS Proton 角色。

     [启用基于 CodeBuild 的管道预置] 在配置账户设置页面上的 CodeBuild 管道角色部分中，选择现有的服务角色，然后选择您在 CloudFormation 管道角色部分中创建的服务角色。或者，如果您没有分配 CloudFormation 管道角色，请重复前面的三个步骤以创建新的服务角色。
   - 选择现有的管道服务角色

     [启用 AWS 托管式管道预置] 在配置账户设置页面上的 AWS 托管式预置管道角色部分中，选择现有的服务角色，然后在您的 AWS 账户中选择一个服务角色。

     [启用 CodeBuild 管道预置] 在配置账户设置页面上的 CodeBuild 管道预置角色部分中，选择现有的服务角色，然后在您的 AWS 账户中选择一个服务角色。
5. 选择保存更改。

   将在账户设置页面上显示您的新管道服务角色。

自托管式预置 - 设置管道存储库

通过使用自托管式预置，AWS Proton 向您设置的预置存储库发送拉取请求 (PR)，并且您的自动化代码负责预置管道。因此，AWS Proton 不需要具有服务角色以预置管道。相反，它需要使用注册的预置存储库。存储库中的自动化代码必须担任相应的角色，以提供预置管道的权限。

使用控制台注册管道预置存储库

1. 如果您尚未创建 CI/CD 管道预置存储库，请创建一个存储库。有关自托管式预置中的管道的更多信息，请参阅the section called “自托管式预置”。
2. 在导航窗格中，选择设置，然后选择账户设置。
3. 在账户 CI/CD 设置页面中，选择配置。
4. 在配置账户设置页面上的 CI/CD 管道存储库部分中：
   a. 选择新存储库，然后选择存储库提供商之一。
   b. 对于 CodeStar 连接，选择您的连接之一。

      > **Note**
      >
      > 如果您尚未连接到相关的存储库提供商账户，请选择添加新的 CodeStar 连接，完成连接创建过程，然后选择 CodeStar 连接菜单旁边的刷新按钮。您现在应该可以在菜单中选择您的新连接。

   c. 对于存储库名称，选择您的管道预置存储库。下拉菜单显示提供商账户中的存储库列表。
   d. 对于分支名称，选择存储库分支之一。
5. 选择保存更改。

   将在账户设置页面上显示您的管道存储库。

设置 AWS CLI

要使用 AWS CLI 进行 AWS Proton API 调用，请确认您已安装最新版本的 AWS CLI。有关更多信息，请参阅 AWS Command Line Interface 用户指南中的 AWS CLI 入门。然后，要开始将 AWS CLI 与 AWS Proton 一起使用，请参阅the section called “CLI 入门”。

AWS Proton 入门

在开始之前，请进行设置以使用 AWS Proton，并验证您是否满足入门先决条件。

选择下面的一个或多个路径以开始使用 AWS Proton：

- 通过文档链接执行指导式示例控制台或 CLI 工作流。
- 运行指导式示例控制台工作流。
- 运行指导式示例 AWS CLI 工作流。

主题

先决条件

在开始使用 AWS Proton 之前，请确保满足以下先决条件。有关更多信息，请参阅设置。

- 您使用一个具有管理员权限的 IAM 账户。有关更多信息，请参阅使用 IAM 进行设置。
- 您具有 AWS Proton 服务角色，并将 AWS Proton 管道服务角色附加到您的账户。有关更多信息，请参阅 设置 AWS Proton 服务角色 和 AWS Proton 的服务角色：
- 您具有 AWS CodeStar 连接。有关更多信息，请参阅建立 AWS CodeStar 连接。
- 您熟悉如何创建 AWS CloudFormation 模板和 Jinja 参数化。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的什么是 AWS CloudFormation？和 Jinja 网站。
- 您具有 AWS 基础设施服务的工作经验。
- 您已登录到您的 AWS 账户。

入门工作流

按照示例步骤和链接，了解如何创建模板捆绑包，创建并注册模板以及创建环境和服务。

在开始之前，验证您是否创建了 AWS Proton 服务角色。

如果您的服务模板包含 AWS Proton 服务管道，请验证您是否创建了 AWS CodeStar 连接和 AWS Proton 管道服务角色。

有关更多信息，请参阅 The AWS Proton service API Reference。

示例：入门工作流

1. 请参阅AWS Proton 的工作原理中的图表，以简要了解 AWS Proton 输入和输出。
2. 创建一个环境捆绑包和服务模板捆绑包。

   a. 指定输入参数。

   b. 创建架构文件。

   c. 创建基础设施即代码 (IaC) 文件。

   d. 要打包模板捆绑包，请创建一个清单文件，并将 IaC 文件、清单文件和架构文件放置到不同的目录中。

   e. 使您的模板捆绑包可供 AWS Proton 访问。
3. 使用 AWS Proton 创建并注册一个环境模板版本。

   在您使用控制台创建并注册模板时，将自动创建一个模板版本。

   在您使用 AWS CLI 创建并注册模板时：

   a. 创建一个环境模板。

   b. 创建一个环境模板版本。

   有关更多信息，请参阅 AWS Proton API reference 中的 CreateEnvironmentTemplate 和 CreateEnvironmentTemplateVersion。
4. 发布您的环境模板以使其可供使用。

   有关更多信息，请参阅 AWS Proton API reference 中的 UpdateEnvironmentTemplateVersion。
5. 要创建环境，请选择一个发布的环境模板版本，并提供所需输入的值。

   有关更多信息，请参阅 AWS Proton API reference 中的 CreateEnvironment。
6. 使用 AWS Proton 创建并注册一个服务模板版本。

   在您使用控制台创建并注册模板时，将自动创建一个模板版本。

在您使用 AWS CLI 创建并注册模板时：

a. 创建一个服务模板。

b. 创建一个服务模板版本。

有关更多信息，请参阅 AWS Proton API reference 中的 CreateServiceTemplate 和 CreateServiceTemplateVersion。

7. 发布您的服务模板以使其可供使用。

有关更多信息，请参阅 AWS Proton API reference 中的 UpdateServiceTemplateVersion。

8. 要创建服务，请选择一个发布的服务模板版本，并提供所需输入的值。

有关更多信息，请参阅 AWS Proton API reference 中的 CreateService。

AWS Management Console入门

开始使用 AWS Proton

- 创建并查看环境模板。
- 创建、查看和发布使用您刚创建的环境模板的服务模板。
- 创建环境和服务（可选）。
- 删除服务模板、环境模板、环境和服务（如果已创建）。

步骤 1：打开 AWS Proton 控制台

- 打开 AWS Proton 控制台

步骤 2：准备使用示例模板

1. 创建到 Github 的 CodeStar 连接，并将连接命名为 my-proton-connection。
2. 导航到 https://github.com/aws-samples/aws-proton-cloudformation-sample-templates
3. 在您的 Github 账户中创建存储库的分支。

步骤 3：创建环境模板

在导航窗格中，选择环境模板。

1. 在环境模板页面中，选择创建环境模板。
2. 在创建环境模板页面上的模板选项部分中，选择创建用于预置新环境的模板。
3. 在模板捆绑包源部分中，选择从 Git 同步模板捆绑包。
4. 在模板定义存储库部分中，选择选择链接的 Git 存储库。
5. 从存储库列表中选择 my-proton-connection 。
6. 从分支列表中选择 main。
7. 在 Proton 环境模板详细信息部分中。
   a. 将模板名称输入为 **`fargate-env`**。
   b. 将环境模板显示名称输入为 **`My Fargate Environment`**。
   c. （可选）输入环境模板的描述。
8. （可选）在标签部分中，选择添加新标签，并输入键和值以创建一个客户托管标签。
9. 选择创建环境模板。

   您现在位于一个新页面中，其中显示新环境模板的状态和详细信息。这些详细信息包括 AWS 托管标签和客户托管标签列表。在您创建 AWS Proton 资源时，AWS Proton 自动为您生成 AWS 托管标签。有关更多信息，请参阅AWS Proton 资源和标记。
10. 新环境模板的初始状态为草稿。您和具有 `proton:CreateEnvironment` 权限的其他人可以查看和访问该模板。执行下一步，以使该模板可供其他人使用。
11. 在模板版本部分中，选择刚创建的模板次要版本 (1.0) 左侧的单选按钮。或者，您可以在信息提醒横幅中选择发布并跳过下一步。
12. 在模板版本部分中，选择发布。
13. 模板状态变为已发布。由于它是最新的模板版本，因此，它是推荐版本。
14. 在导航窗格中，选择环境模板。

    新页面将显示环境模板列表以及模板详细信息。

步骤 4：创建服务模板

创建一个服务模板。

1. 在导航窗格中，选择服务模板。
2. 在服务模板页面中，选择创建服务模板。
3. 在创建服务模板页面上的模板捆绑包源部分中，选择从 Git 同步模板捆绑包。
4. 在模板部分中，选择选择链接的 Git 存储库。
5. 从存储库列表中选择 my-proton-connection 。
6. 从分支列表中选择 main。
7. 在 Proton 服务模板详细信息部分中。
   a. 将服务模板名称输入为 **backend-fargate-svc**。
   b. 将服务模板显示名称输入为 **My Fargate Service**。
   c. （可选）输入服务模板的描述。
8. 在兼容的环境模板部分中。
   - 选中 My Fargate Environment 环境模板左侧的复选框，为新服务模板选择兼容的环境模板。
9. 对于加密设置，保留默认值。
10. 在管道定义部分中。
    - 将此模板包括 CI/CD 管道按钮保持选中状态。
11. 选择创建服务模板。

    您现在位于一个新页面中，其中显示新服务模板的状态和详细信息，包括 AWS 和客户托管标签列表。
12. 新服务模板的初始状态为草稿。仅管理员可以查看和访问该状态。要使服务模板可供开发人员使用，请执行下一步。
13. 在模板版本部分中，选择刚创建的模板次要版本 (1.0) 左侧的单选按钮。或者，您可以在信息提醒横幅中选择发布并跳过下一步。
14. 在模板版本部分中，选择发布。
15. 模板状态变为已发布。

    您的服务模板的第一个次要版本已发布并且可供开发人员使用。由于它是最新的模板版本，因此，它是推荐版本。

16. 在导航窗格中，选择服务模板。

    新页面将显示您的服务模板和详细信息列表。

步骤 5：创建环境

在导航窗格中，选择环境。

1. 选择 Create environment（创建环境）。
2. 在选择环境模板页面中，选择刚创建的模板。该模板命名为 My Fargate Environment。然后，选择配置。
3. 在配置环境页面上的预置部分中，选择通过 AWS Proton 进行预置。
4. 在部署账户部分中，选择该 AWS 账户 账户。
5. 在环境设置中，将环境名称输入为 **`my-fargate-environment`**。
6. 在环境角色部分中，选择新服务角色，或者，如果您已创建 AWS Proton 服务角色，请选择现有的服务角色。

   a. 选择新服务角色以创建一个新角色。

      i. 将环境角色名称输入为 **`MyProtonServiceRole`**。

      ii. 选中该复选框以同意为您的账户创建具有管理权限的 AWS Proton 服务角色。

   b. 选择现有的服务角色以使用一个现有的角色。

      - 在环境角色名称下拉字段中选择您的角色。
7. 选择下一步。
8. 在配置自定义设置页面上，使用默认值。
9. 选择下一步并检查您的输入。
10. 选择创建。

    查看环境详细信息和状态，以及您的环境的 AWS 托管标签和客户托管标签。

11. 在导航窗格中，选择环境。

    新页面显示您的环境列表以及状态和其他环境详细信息。

步骤 6：可选 - 创建服务并部署应用程序

1. 打开 AWS Proton 控制台。
2. 在导航窗格中，选择服务。
3. 在服务页面中，选择创建服务。
4. 在选择服务模板页面中，选择模板卡右上角的单选按钮以选择 My Fargate Service 模板。
5. 选择页面右下角的配置。
6. 在配置服务页面上的服务设置部分中，输入服务名称 **my-service**。
7. （可选）输入服务的描述。
8. 在服务存储库设置部分中：
   a. 对于 CodeStar 连接，从列表中选择您的连接。
   b. 对于存储库名称，从列表中选择您的源代码存储库的名称。
   c. 对于分支名称，从列表中选择您的源代码存储库分支的名称。
9. （可选）在标签部分中，选择添加新标签，并输入键和值以创建一个客户托管标签。然后选择下一步。
10. 在配置自定义设置页面上的服务实例部分中，在新实例部分中执行以下步骤，为您的服务实例参数提供自定义值。
    a. 输入实例名称 **my-app-service**。
    b. 为您的服务实例选择 **my-fargate-environment** 环境。
    c. 保留其余实例参数的默认值。
    d. 保留管道输入的默认值。
    e. 选择下一步并检查您的输入。
    f. 选择创建并查看您的服务状态和详细信息。
11. 在服务详细信息页面中，选择概述和管道选项卡以查看您的服务实例和管道的状态。在这些页面上，您还可以查看 AWS 和客户托管标签。AWS Proton 自动为您创建 AWS 托管标签。选择管理标签以创建和修改客户托管标签。有关标记的更多信息，请参阅AWS Proton 资源和标记。
12. 在服务处于活动状态后，在概述选项卡上的服务实例部分中，选择您的服务实例的名称 my-app-service。

    您现在位于服务实例详细信息页面上。
13. 要查看您的应用程序，请在输出部分中将 ServiceEndpoint 链接复制到您的浏览器。

您将在网页中看到 AWS Proton 图形。

14. 在创建服务后，在导航窗格中选择服务以查看您的服务列表。

步骤 7：清理

1. 打开 AWS Proton 控制台。
2. 删除服务（如果已创建）
   a. 在导航窗格中，选择服务。
   b. 在服务页面中，选择服务名称 my-service。

      您现在位于 my-service 的服务详细信息页面上。
   c. 在页面上的右上角，选择操作，然后选择删除。
   d. 一个模态框提示您确认删除操作。
   e. 按照说明进行操作并选择是，删除。
3. 删除环境
   a. 在导航窗格中，选择环境。
   b. 在环境页面中，选择刚创建的环境左侧的单选按钮。
   c. 选择操作，然后选择删除。
   d. 一个模态框提示您确认删除操作。
   e. 按照说明进行操作并选择是，删除。
4. 删除服务模板
   a. 在导航窗格中，选择服务模板。
   b. 在服务模板页面中，选择 my-svc-template 服务模板左侧的单选按钮。
   c. 选择操作，然后选择删除。
   d. 一个模态框提示您确认删除操作。
   e. 按照说明进行操作并选择是，删除。这会删除服务模板及其所有版本。
5. 删除环境模板
   a. 在导航窗格中，选择环境模板。
   b. 在环境模板页面中，选择 my-env-template 左侧的单选按钮。

c. 选择操作，然后选择删除。

d. 一个模态框提示您确认删除操作。

e. 按照说明进行操作并选择是，删除。这会删除环境模板及其所有版本。

6. 删除 CodeStar 连接

AWS CLI入门

要开始通过 AWS CLI 使用 AWS Proton，请按照本教程进行操作。本教程说明了一个基于 AWS Fargate 的面向公众的负载均衡 AWS Proton 服务。本教程还预置了一个 CI/CD 管道，用于部署一个具有显示的图像的静态网站。

在开始之前，请确保您正确进行设置。有关详细信息，请参阅 the section called “先决条件”。

步骤 1：注册环境模板

在该步骤中，您作为管理员注册一个示例环境模板，其中包含 Amazon Elastic Container Service (Amazon ECS) 集群以及具有两个公有/私有子网的 Amazon Virtual Private Cloud (Amazon VPC)。

注册环境模板

1. 将 AWS Proton 示例 CloudFormation 模板存储库复制到您的 GitHub 账户或组织中。该存储库包含我们在本教程中使用的环境和服务模板。

   然后，在 AWS Proton 中注册复制的存储库。有关更多信息，请参阅the section called “创建存储库链接”。

2. 创建一个环境模板。

   环境模板资源跟踪环境模板版本。

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

3. 创建一个模板同步配置。

   AWS Proton 在您的存储库和环境模板之间建立同步关系。然后，它创建处于 DRAFT 状态的模板版本 1.0。

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

4. 等待成功注册环境模板版本。

   如果该命令返回并且退出状态为 0，则说明版本注册完成。这在脚本中是非常有用的，可以确保您在下一步中成功运行命令。

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

5. 发布环境模板版本以使其可用于创建环境。

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

步骤 2：注册服务模板

在该步骤中，您作为管理员注册一个示例服务模板，其中包含在负载均衡器后面预置 Amazon ECS Fargate 服务所需的所有资源以及使用 AWS CodePipeline 的 CI/CD 管道。

注册服务模板

1. 创建一个服务模板。

   服务模板资源跟踪服务模板版本。

```
$ aws proton create-service-template \
  --name "load-balanced-fargate-svc" \
  --display-name "Load balanced Fargate service" \
```

```
  --description "Fargate service with an application load balancer"
```

2. 创建一个模板同步配置。

   AWS Proton 在您的存储库和服务模板之间建立同步关系。然后，它创建处于 DRAFT 状态的模板版本 1.0。

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

3. 等待成功注册服务模板版本。

   如果该命令返回并且退出状态为 `0`，则说明版本注册完成。这在脚本中是非常有用的，可以确保您在下一步中成功运行命令。

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

4. 发布服务模板版本以使其可用于创建服务。

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

步骤 3：部署环境

在该步骤中，您作为管理员通过环境模板实例化一个 AWS Proton 环境。

部署环境

1. 为您注册的环境模板获取一个示例规范文件。

您可以从模板示例存储库中下载 `environment-templates/fargate-env/spec/spec.yaml` 文件。或者，您可以在本地获取整个存储库，并从 `environment-templates/fargate-env` 目录中运行 create-environment 命令。

2. 创建一个环境。

   AWS Proton 从您的环境规范中读取输入值，将它们与您的环境模板合并在一起，并使用您的 AWS Proton 服务角色在您的 AWS 账户中预置环境资源。

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

3. 等待成功部署环境。

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

步骤 4：部署服务 [应用程序开发人员]

在前面的步骤中，管理员注册并发布了一个服务模板，并部署了一个环境。作为应用程序开发人员，您现在可以创建一个 AWS Proton 服务，并将其部署到 AWS Proton 环境中。

部署服务

1. 为管理员注册的服务模板获取一个示例规范文件。

   您可以从模板示例存储库中下载 `service-templates/load-balanced-fargate-svc/spec/spec.yaml` 文件。或者，您可以在本地获取整个存储库，并从 `service-templates/load-balanced-fargate-svc` 目录中运行 create-service 命令。

2. 将 AWS Proton 示例服务存储库复制到您的 GitHub 账户或组织中。该存储库包含我们在本教程中使用的应用程序源代码。

3. 创建服务。

   AWS Proton 从您的服务规范中读取输入值，将其与您的服务模板合并在一起，并在您的 AWS 账户中为规范指定的环境预置服务资源。AWS CodePipeline 管道从您在命令中指定的存储库中部署应用程序代码。

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

4. 等待成功部署服务。

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

5. 检索输出并查看您的新网站。

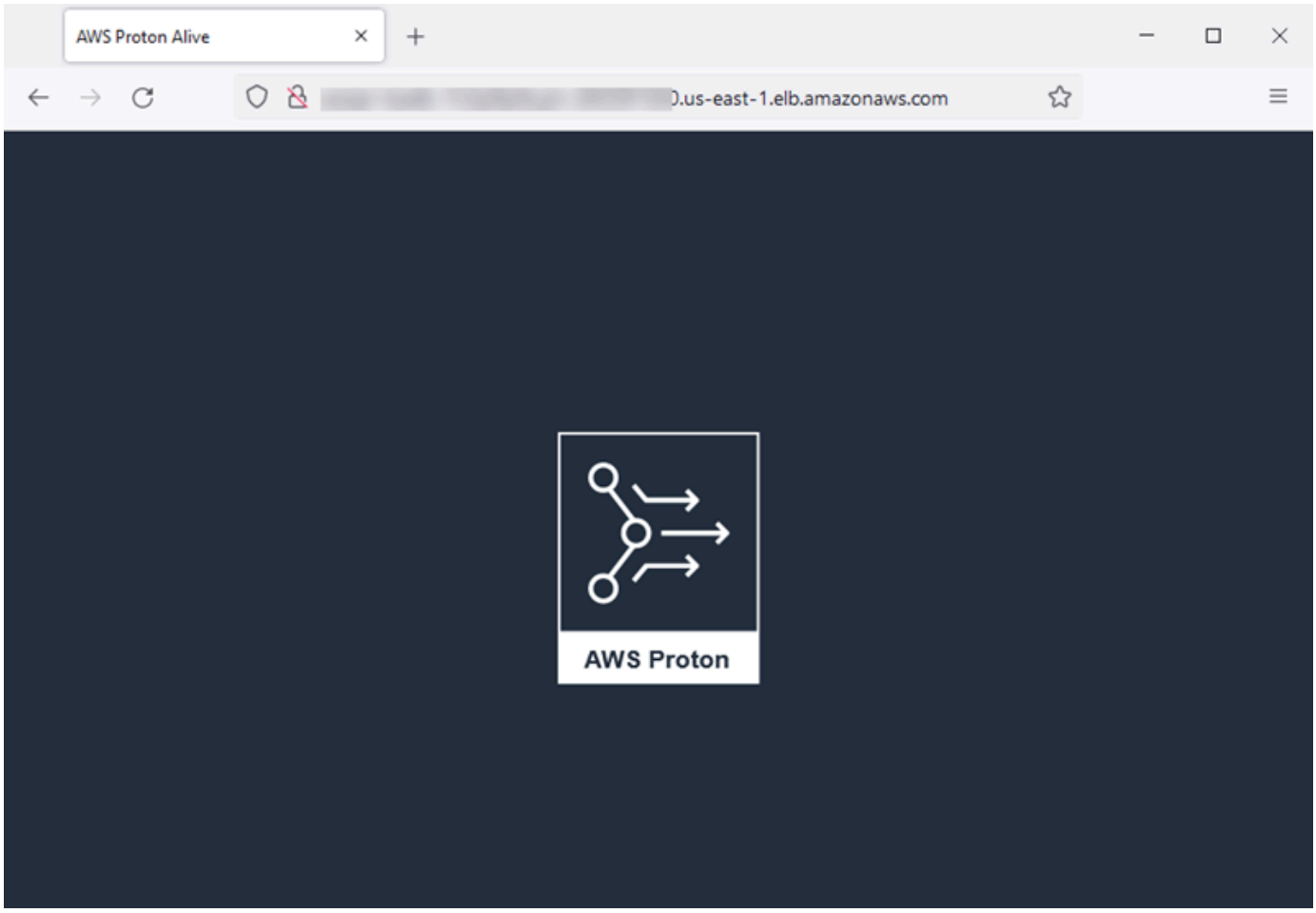
   运行以下命令：

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

   该命令的输出应类似于以下内容：

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

   ServiceURL 实例输出的值是新服务网站的终端节点。使用浏览器导航到该终端节点。您应该会在静态页面上看到以下图形：



步骤 5：清理（可选）

在该步骤中，在您了解了本教程中创建的 AWS 资源后，您可以删除这些资源以节省与它们相关的成本。

删除教程资源

1. 要删除服务，请运行以下命令：

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

2. 要删除环境，请运行以下命令：

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

3. 要删除服务模板，请运行以下命令：

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

4. 要删除环境模板，请运行以下命令：

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

AWS Proton 模板库

AWS Proton 团队在 GitHub 上维护一个模板示例库。该库包含适用于很多常见环境和应用程序基础设施场景的基础设施即代码 (IaC) 文件示例。

该模板库存储在两个 GitHub 存储库中：

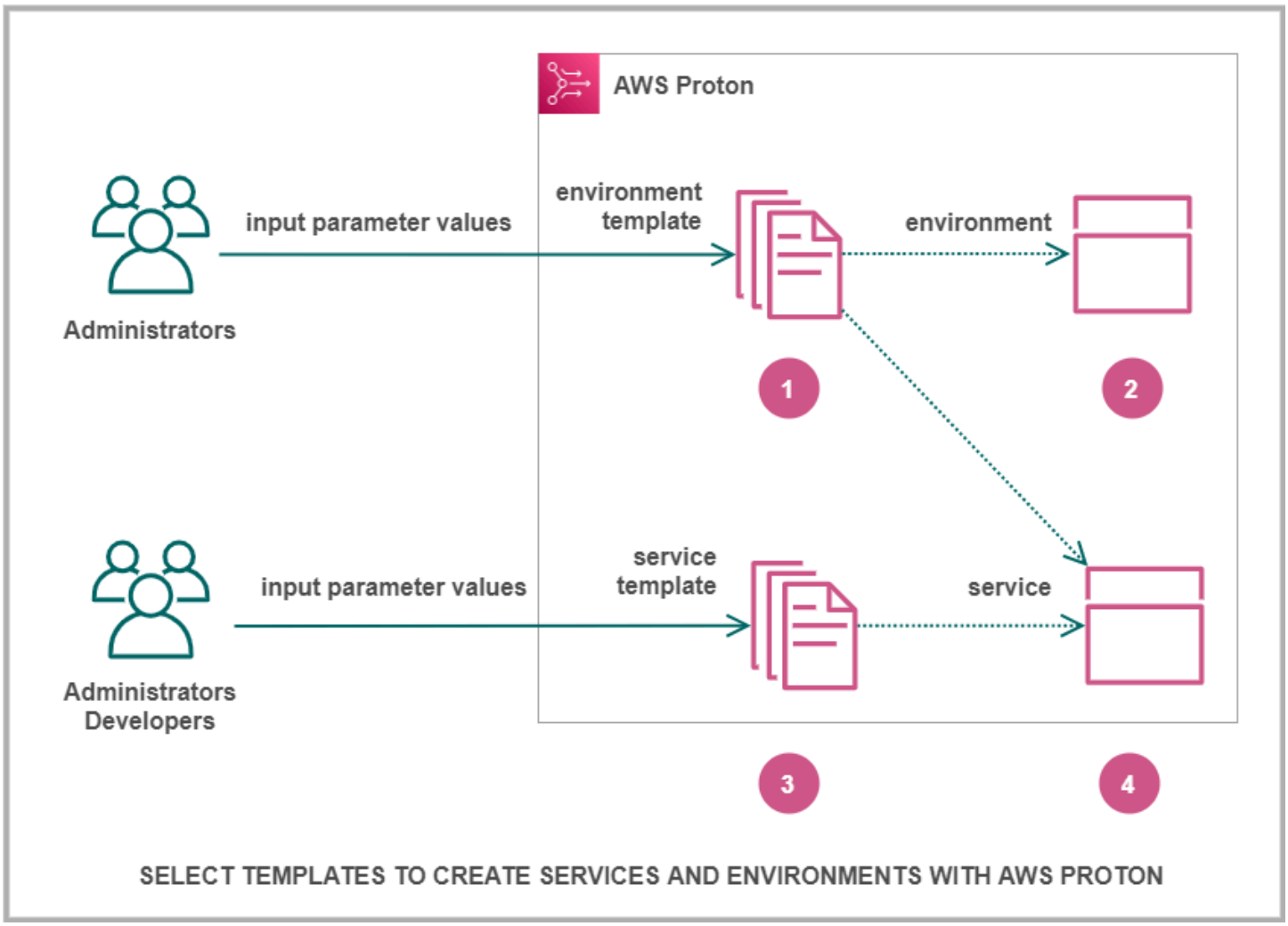
- aws-proton-cloudformation-sample-templates - 使用 AWS CloudFormation 并将 Jinja 作为 IaC 语言的模板捆绑包示例。您可以将这些示例用于 AWS 托管式预置 环境。
- aws-proton-terraform-sample-templates - 将 Terraform 作为 IaC 语言的模板捆绑包示例。您可以将这些示例用于 自托管式预置 环境。

每个存储库具有一个 README 文件，其中包含有关存储库内容和结构的完整信息。每个示例包含有关模板涵盖的使用案例、示例的架构以及模板采用的输入参数的信息。

您可以将该库的存储库之一复制到您的 GitHub 账户，以直接使用该库中的模板。或者，将这些示例作为开发环境和服务模板的起点。

AWS Proton 的工作原理

通过使用 AWS Proton，您可以预置环境，然后预置在这些环境中运行的服务。环境和服务分别基于您在 AWS Proton 版本控制的模板库中选择的环境和服务模板。



1 作为管理员，在您使用 AWS Proton 选择环境模板时，您提供所需的输入参数 值。

2 AWS Proton 使用环境模板和参数值预置您的环境。

3 作为开发人员或管理员，在您使用 AWS Proton 选择服务模板时，您提供所需的输入参数值。您也可以选择一个环境以部署您的应用程序或服务。

4

AWS Proton 使用服务模板以及您的服务和选定环境参数值预置您的服务。

您提供输入参数的值以自定义您的模板，以在多个使用案例、应用程序或服务中重用。

为了实现该目的，您需要创建环境或服务模板捆绑包，并将它们分别上传到注册的环境或服务模板。

模板捆绑包包含 AWS Proton 预置环境或服务所需的所有内容。

在创建环境或服务模板时，您需要上传一个模板捆绑包，其中包含 AWS Proton 用于预置环境或服务的参数化基础设施即代码 (IaC) 文件。

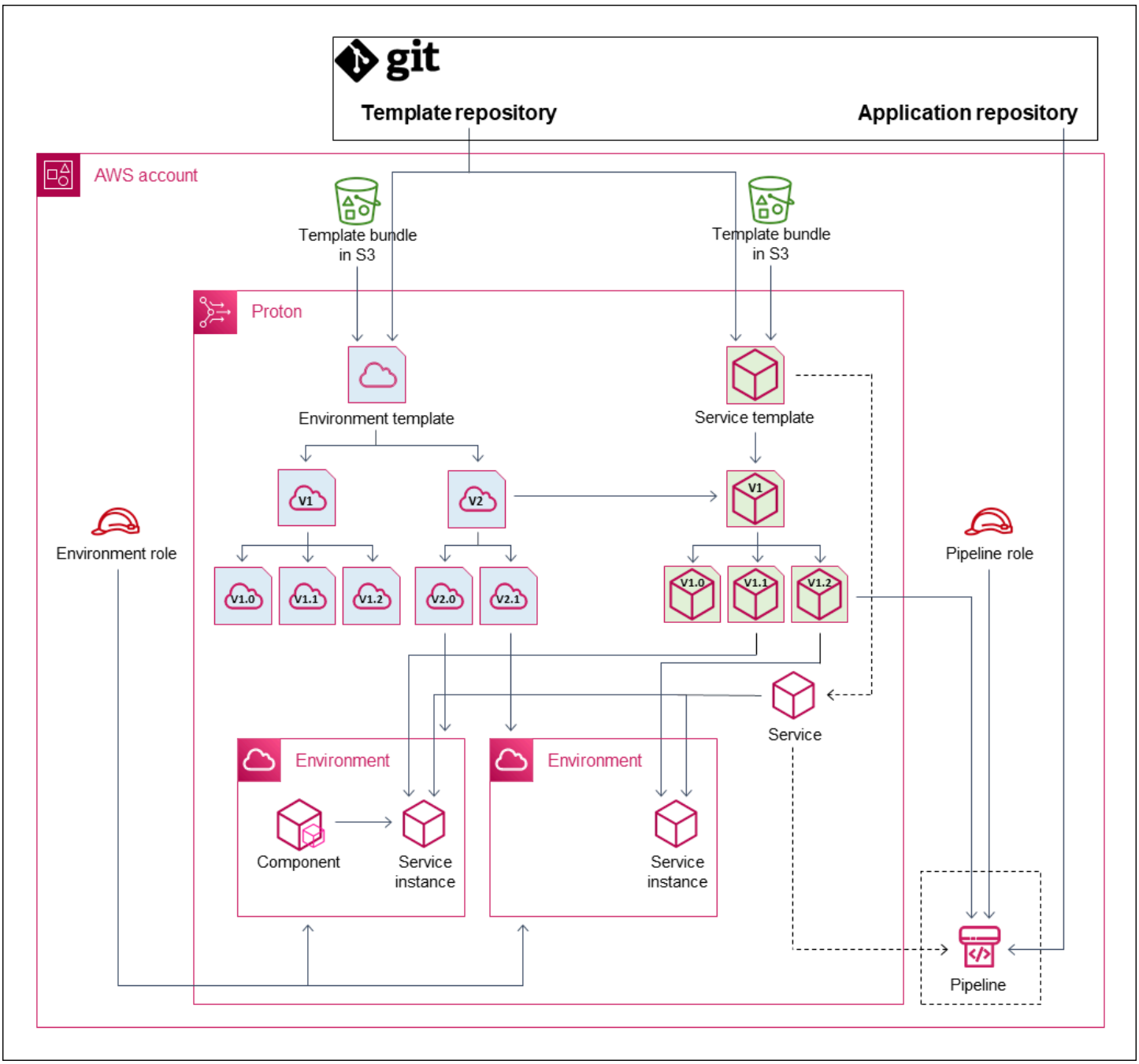
在您选择环境或服务模板以创建或更新环境或服务时，您需要提供模板捆绑包 IaC 文件参数的值。

主题

AWS Proton 对象

下图显示了主要的 AWS Proton 对象及其与其他 AWS 对象和第三方对象的关系。箭头表示数据流方向（依赖项的反方向）。

我们在图表后面提供了这些 AWS Proton 对象的简要说明和参考链接。



- 环境模板 - 可用于创建 AWS Proton 环境的环境模板版本集合。

  有关更多信息，请参阅 模板编写和捆绑包 和 模板：

- 环境模板版本 - 特定的环境模板版本。将 S3 存储桶或 Git 存储库中的模板捆绑包作为输入。该捆绑包为 AWS Proton 环境指定基础设施即代码 (IaC) 和相关的输入参数。

  有关更多信息，请参阅the section called “版本”、the section called “发布”和the section called “模板同步配置”。

- 环境 - 将 AWS Proton 服务部署到的一组共享 AWS 基础设施资源和访问策略。AWS 资源是使用通过特定参数值调用的环境模板版本预置的。访问策略是在服务角色中提供的。

  有关更多信息，请参阅环境。

- 服务模板 - 可用于创建 AWS Proton 服务的服务模板版本集合。

  有关更多信息，请参阅 模板编写和捆绑包 和 模板：

- 服务模板版本 - 特定的服务模板版本。将 S3 存储桶或 Git 存储库中的模板捆绑包作为输入。该捆绑包为 AWS Proton 服务指定基础设施即代码 (IaC) 和相关的输入参数。

  服务模板版本还根据版本指定对服务实例的以下限制：

  - 兼容的环境模板 - 实例只能在基于这些兼容的环境模板的环境中运行。
  - 支持的组件源 - 指定开发人员可以将哪些组件类型与实例相关联。

  有关更多信息，请参阅the section called “版本”、the section called “发布”和the section called “模板同步配置”。

- 服务 - 使用服务模板中指定的资源运行应用程序的服务实例集合（可能具有将应用程序代码部署到这些实例的 CI/CD 管道）。

  在该图中，从服务模板引出的虚线表示服务将模板传送到服务实例和管道。

  有关更多信息，请参阅服务。

- 服务实例 - 在特定 AWS Proton 环境中运行应用程序的 AWS 基础设施资源集。AWS 资源是使用以特定参数值调用的服务模板版本预置的。

  有关更多信息，请参阅 服务 和 the section called “更新实例”：

- 管道 - 将应用程序部署到服务实例的可选 CI/CD 管道，并具有预置该管道的访问策略。访问策略是在服务角色中提供的。服务并非始终具有关联的 AWS Proton 管道 - 您可以选择在 AWS Proton 外部管理应用程序代码部署。

  在该图中，从服务引出的虚线以及管道周围的虚线框意味着，如果您选择自行管理 CI/CD 部署，则可能不会创建 AWS Proton 管道，并且您自己的管道可能不在您的 AWS 账户范围内。

  有关更多信息，请参阅 服务 和 the section called “更新管道”：

- 组件 - 开发人员定义的服务实例扩展。除了环境和服务实例提供的资源以外，指定特定应用程序可能需要的其他 AWS 基础设施资源。平台团队将一个组件角色附加到环境，以控制组件可以预置的基础设施。

有关更多信息，请参阅组件。

AWS Proton 如何预置基础设施

AWS Proton 可以通过以下几种方法之一预置基础设施：

- AWS 托管式预置 - AWS Proton 代表您调用预置引擎。该方法仅支持 AWS CloudFormation 模板捆绑包。有关更多信息，请参阅the section called “AWS CloudFormation IaC 文件”。
- CodeBuild 预置 - AWS Proton 使用 AWS CodeBuild 运行您提供的 Shell 命令。您的命令可以读取 AWS Proton 提供的输入，并负责预置或取消预置基础设施和生成输出值。该方法的模板捆绑包包括清单文件中的命令，以及这些命令可能需要的任何程序、脚本或其他文件。

  作为一个使用 CodeBuild 预置的示例，您可以包含使用 AWS Cloud Development Kit (AWS CDK) 预置 AWS 资源的代码，以及安装 CDK 和运行 CDK 代码的清单。

  有关更多信息，请参阅the section called “CodeBuild 捆绑包”。

  > **Note**
  >
  > 您可以将 CodeBuild 预置与环境和服务一起使用。目前，您无法通过这种方法预置组件。

- 自托管式预置 - AWS Proton 向您提供的存储库发出拉取请求 (PR)，您自己的基础设施部署系统在其中运行预置过程。该方法仅支持 Terraform 模板捆绑包。有关更多信息，请参阅the section called “Terraform IaC 文件”。

AWS Proton 单独确定和设置每个环境和服务的预置方法。在您创建或更新环境或服务时，AWS Proton 检查您提供的模板捆绑包，并确定模板捆绑包指示的预置方法。在环境级别，您提供环境及其潜在服务的预置方法可能需要的参数 - AWS Identity and Access Management (IAM) 角色、环境账户连接或基础设施存储库。

无论预置方法如何，使用 AWS Proton 预置服务的开发人员都会获得相同的体验。开发人员不需要了解预置方法，也不需要在服务预置过程中更改任何内容。服务模板设置预置方法，开发人员部署服务的每个环境提供服务实例预置所需的参数。

下图简要说明了不同预置方法的一些主要特征。表后面的部分提供了有关每种方法的详细信息。

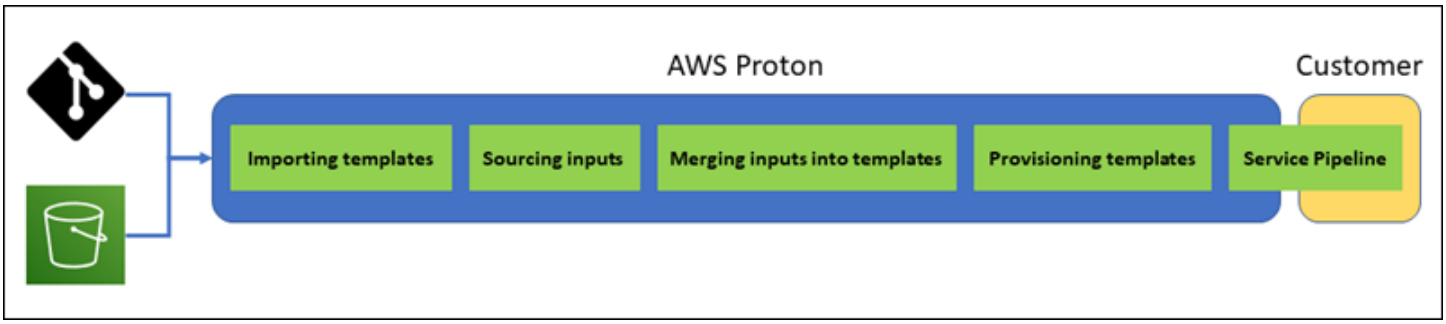
预置方法	模板	预置者	状态跟踪者
AWS 托管式	清单、架构、IaC 文件 (CloudFormation)	AWS Proton（通过 CloudFormation）	AWS Proton（通过 CloudFormation）
CodeBuild	清单（包含命令）、架构、命令依赖项（例如 AWS CDK 代码）	AWS Proton（通过 CodeBuild）	AWS Proton（您的命令通过 CodeBuild 返回状态）
自行管理	清单、架构、IaC 文件 (Terraform)	您的代码（通过 Git 操作）	您的代码（通过 API 调用传递给 AWS）

AWS 托管式预置的工作方式

在环境或服务使用 AWS 托管式预置时，基础设施按以下方式进行预置：

1. AWS Proton 客户（管理员或开发人员）创建 AWS Proton 资源（环境或服务）。客户为资源选择模板，并提供所需的参数。有关更多信息，请参阅the section called “AWS 托管式预置的注意事项”一节。
2. AWS Proton 渲染完整的 AWS CloudFormation 模板以预置资源。
3. AWS Proton 调用 AWS CloudFormation 以使用渲染的模板开始预置。
4. AWS Proton 持续监控 AWS CloudFormation 部署。
5. 在预置完成时，AWS Proton 在失败时报告错误，并在成功时捕获预置输出，例如 Amazon VPC ID。

下图显示 AWS Proton 直接处理其中的大多数步骤。



AWS 托管式预置的注意事项

- 基础设施预置角色 - 在环境或其中运行的任何服务实例可能使用 AWS 托管式预置时，管理员需要配置一个 IAM 角色（直接配置或作为 AWS Proton 环境账户连接的一部分）。AWS Proton 使用该角色预置这些 AWS 托管式预置资源的基础设施。该角色应有权使用 AWS CloudFormation 创建这些资源的模板包含的所有资源。

  有关更多信息，请参阅 the section called “IAM 角色” 和 the section called “服务角色策略示例”：
- 服务预置 - 在开发人员将使用 AWS 托管式预置的服务实例部署到环境时，AWS Proton 使用为该环境提供的角色预置服务实例的基础设施。开发人员看不到该角色，也无法更改该角色。
- 具有管道的服务 - 使用 AWS 托管式预置的服务模板可能包含以 AWS CloudFormation YAML 架构编写的管道定义。AWS Proton 还可以调用 AWS CloudFormation 以创建管道。AWS Proton 用于创建管道的角色与每个单独环境的角色是分开的。该角色单独提供给 AWS Proton，仅在 AWS 账户级别提供一次，用于预置和管理所有 AWS 托管的管道。该角色应有权创建管道以及管道所需的其他资源。

  以下过程说明了如何向 AWS Proton 提供管道角色。

  AWS Proton console

  提供管道角色

  1. 在 AWS Proton 控制台的导航窗格中，选择设置 > 账户设置，然后选择配置。
  2. 使用管道 AWS 托管式角色部分为 AWS 托管式预置配置新的或现有的管道角色。

  AWS Proton API

  提供管道角色

  1. 使用 UpdateAccountSettings API 操作。
  2. 在 `pipelineServiceRoleArn` 参数中提供您的管道服务角色的 Amazon 资源名称 (ARN)。

  AWS CLI

  提供管道角色

  运行以下命令：

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

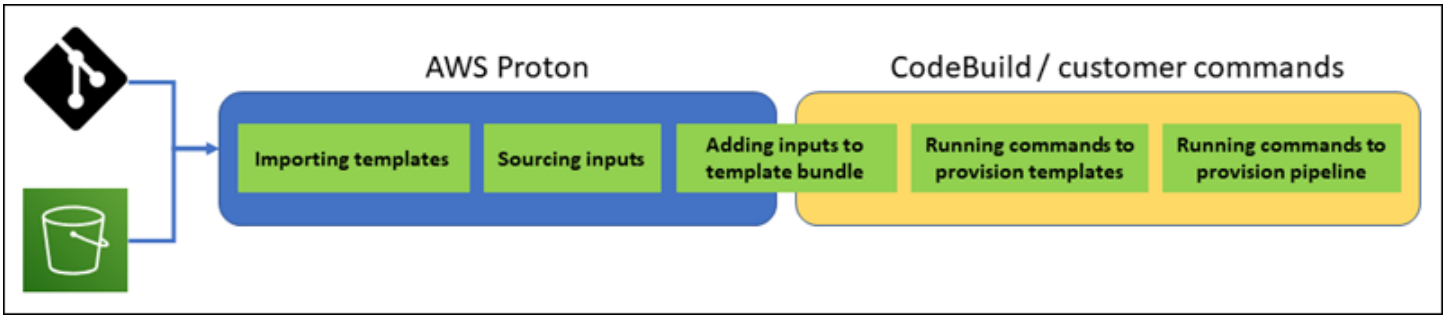
CodeBuild 预置的工作方式

在环境或服务使用 CodeBuild 预置时，基础设施按以下方式进行预置：

1. AWS Proton 客户（管理员或开发人员）创建 AWS Proton 资源（环境或服务）。客户为资源选择模板，并提供所需的参数。有关更多信息，请参阅the section called “CodeBuild 预置注意事项”一节。
2. AWS Proton 使用输入参数值渲染输入文件以预置资源。
3. AWS Proton 调用 CodeBuild 以启动一个作业。CodeBuild 作业运行模板中指定的客户 Shell 命令。这些命令预置所需的基础设施，同时可以选择读取输入值。
4. 在预置完成时，最终客户命令将预置状态返回到 CodeBuild，并调用 NotifyResourceDeploymentStatusChange AWS Proton API 操作以提供输出，例如 Amazon VPC ID（如果存在）。

> **Important**
>
> 确保您的命令正确将预置状态返回到 CodeBuild 并提供输出。否则，AWS Proton 无法正确跟踪预置状态，也无法向服务实例提供正确的输出。

下图说明了 AWS Proton 执行的步骤以及您的命令在 CodeBuild 作业中执行的步骤。



CodeBuild 预置注意事项

- 基础设施预置角色 - 在环境或其中运行的任何服务实例可能使用基于 CodeBuild 的预置时，管理员需要配置一个 IAM 角色（直接配置或作为 AWS Proton 环境账户连接的一部分）。AWS Proton 使用该角色预置这些 CodeBuild 预置资源的基础设施。该角色应有权使用 CodeBuild 创建这些资源的模板中的命令预置的所有资源。

  有关更多信息，请参阅 the section called “IAM 角色” 和 the section called “服务角色策略示例”：
- 服务预置 - 在开发人员将使用 CodeBuild 预置的服务实例部署到环境时，AWS Proton 使用为该环境提供的角色预置服务实例的基础设施。开发人员看不到该角色，也无法更改该角色。
- 具有管道的服务 - 使用 CodeBuild 预置的服务模板可能包含预置管道的命令。AWS Proton 还可以调用 CodeBuild 以创建管道。AWS Proton 用于创建管道的角色与每个单独环境的角色是分开的。该角色单独提供给 AWS Proton，仅在 AWS 账户级别提供一次，用于预置和管理所有基于 CodeBuild 的管道。该角色应有权创建管道以及管道所需的其他资源。

  以下过程说明了如何向 AWS Proton 提供管道角色。

  AWS Proton console

  提供管道角色

  1. 在 AWS Proton 控制台的导航窗格中，选择设置 > 账户设置，然后选择配置。
  2. 使用 CodeBuild 管道预置角色部分为 CodeBuild 预置配置新的或现有的管道角色。

  AWS Proton API

  提供管道角色

  1. 使用 UpdateAccountSettings API 操作。
  2. 在 `pipelineCodebuildRoleArn` 参数中提供您的管道服务角色的 Amazon 资源名称 (ARN)。

  AWS CLI

  提供管道角色

  运行以下命令：

```
$ aws proton update-account-settings \
```

```
    --pipeline-codebuild-role-arn \
        "arn:aws:iam::123456789012:role/my-pipeline-role"
```

自托管式预置的工作方式

在环境或服务配置为使用自托管式预置时，基础设施按以下方式进行预置：

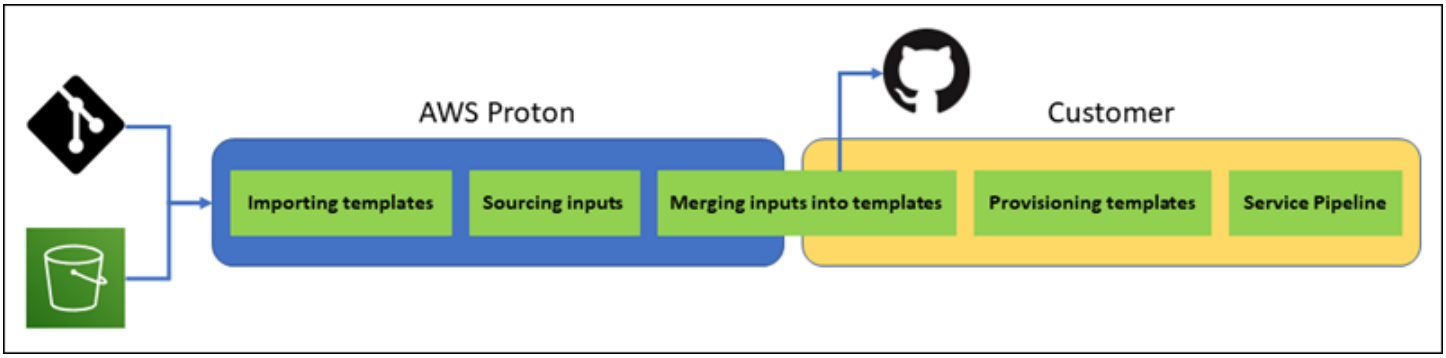
1. AWS Proton 客户（管理员或开发人员）创建 AWS Proton 资源（环境或服务）。客户为资源选择模板，并提供所需的参数。对于环境，客户还会提供链接的基础设施存储库。有关更多信息，请参阅the section called “自托管式预置的注意事项”一节。
2. AWS Proton 渲染完整的 Terraform 模板。它由一个或多个 Terraform 文件（可能位于多个文件夹中）和一个 `.tfvars` 变量文件组成。AWS Proton 将资源创建调用中提供的参数值写入到该变量文件。
3. AWS Proton 使用渲染的 Terraform 模板向基础设施存储库提交 PR。
4. 在客户（管理员或开发人员）合并 PR 时，客户的自动化代码触发预置引擎，以使用合并的模板开始预置基础设施。

> **Note**
>
> 如果客户（管理员或开发人员）关闭 PR，则 AWS Proton 将 PR 识别为已关闭并将部署标记为已取消。

5. 在预置完成时，客户的自动化代码调用 NotifyResourceDeploymentStatusChange AWS Proton API 操作以指示完成，提供状态（成功或失败）并提供输出，例如 Amazon VPC ID（如果存在）。

> **Important**
>
> 确保您的自动化代码回调 AWS Proton 并提供预置状态和输出。否则，AWS Proton 将预置视为待处理的时间可能比应有的时间长，并继续显示进行中状态。

下图说明了 AWS Proton 执行的步骤以及您自己的预置系统执行的步骤。



自托管式预置的注意事项

- 基础设施存储库 - 在管理员为环境配置自托管式预置时，他们需要提供链接的基础设施存储库。AWS Proton 向该存储库提交 PR，以预置环境的基础设施以及部署到环境中的所有服务实例。存储库中的客户拥有的自动化操作应担任一个 IAM 角色，该角色有权创建您的环境和服务模板包含的所有资源并具有反映目标 AWS 账户的身份。有关担任角色的示例 GitHub 操作，请参阅 "Configure AWS Credentials" Action For GitHub Actions 文档中的 Assuming a Role。
- 权限 - 您的预置代码必须根据需要对账户进行身份验证（例如，对 AWS 账户进行身份验证），并提供资源预置授权（例如，提供一个角色）。
- 服务预置 - 在开发人员将使用自托管式预置的服务实例部署到环境时，AWS Proton 向与环境关联的存储库提交 PR 以预置服务实例的基础设施。开发人员看不到存储库，也无法更改存储库。

> **Note**
>
> 无论预置方法如何，创建服务的开发人员使用相同的过程，并对它们的差异进行抽象处理。不过，对于自托管式预置，开发人员可能会遇到响应速度较慢的情况，因为他们需要等待某人（可能不是他们自己）将 PR 合并到基础设施存储库中，然后才能开始预置。

- 具有管道的服务 - 具有自托管式预置的环境的服务模板可能包含使用 Terraform HCL 编写的管道定义（例如 AWS CodePipeline 管道）。要使 AWS Proton 能够预置这些管道，管理员向 AWS Proton 提供一个链接的管道存储库。在预置管道时，存储库中的客户拥有的自动化操作应担任一个 IAM 角色，该角色有权预置管道并具有反映目标 AWS 账户的身份。管道存储库和角色与用于每个单独环境的存储库和角色是分开的。链接的存储库单独提供给 AWS Proton，仅在 AWS 账户级别提供一次，用于预置和管理所有管道。该角色应有权创建管道以及管道所需的其他资源。

  以下过程说明了如何向 AWS Proton 提供管道存储库和角色。

AWS Proton console

提供管道角色

1. 在 AWS Proton 控制台的导航窗格中，选择设置 > 账户设置，然后选择配置。
2. 使用 CI/CD 管道存储库部分配置新的或现有的存储库链接。

AWS Proton API

提供管道角色

1. 使用 UpdateAccountSettings API 操作。
2. 在 `pipelineProvisioningRepository` 参数中提供管道存储库的提供商、名称和分支。

AWS CLI

提供管道角色

运行以下命令：

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

- 删除自托管式预置资源 - 除了资源定义以外，Terraform 模块可能还包含 Terraform 操作所需的配置元素。因此，AWS Proton 不能删除环境或服务实例的所有 Terraform 文件。相反，AWS Proton 将文件标记为删除，并在 PR 元数据中更新一个标记。您的自动化代码可以读取该标记，并使用它触发 Terraform 销毁命令。

AWS Proton 术语

环境模板

定义由多个应用程序或资源使用的共享基础设施，例如 VPC 或集群。

环境模板捆绑包

您上传的一组文件，用于在 AWS Proton 中创建和注册环境模板。环境模板捆绑包包含以下内容：

1. 定义基础设施即代码输入参数的架构文件。
2. 基础设施即代码 (IaC) 文件，用于定义由多个应用程序或资源使用的共享基础设施，例如 VPC 或集群。
3. 列出 IaC 文件的清单文件。

环境

由多个应用程序或资源使用的预置共享基础设施，例如 VPC 或集群。

服务模板

定义在环境中部署和维护应用程序或微服务所需的基础设施类型。

服务模板捆绑包

您上传的一组文件，用于在 AWS Proton 中创建和注册服务模板。服务模板捆绑包包含以下内容：

1. 定义基础设施即代码 (IaC) 输入参数的架构文件。
2. 一个 IaC 文件，用于定义在环境中部署和维护应用程序或微服务所需的基础设施。
3. 列出 IaC 文件的清单文件。
4. 可选
   a. 定义服务管道基础设施的 IaC 文件。
   b. 列出 IaC 文件的清单文件。

服务

定义在环境中部署和维护应用程序或微服务所需的预置基础设施。

服务实例

支持环境中的应用程序或微服务的预置基础设施。

服务管道

支持管道的预置基础设施。

模板版本

模板的主要或次要版本。有关更多信息，请参阅版本控制的模板。

输入参数

在架构文件中定义并在基础设施即代码 (IaC) 文件中使用，以便重复使用 IaC 文件以及将其用于各种使用案例。

架构文件

定义基础设施即代码输入参数。

规范文件

指定架构文件中定义的基础设施即代码文件输入参数的值。

清单文件

列出基础设施即代码文件。

为创作模板和创建捆绑包 AWS Proton

AWS Proton 根据基础设施即代码 (IaC) 文件为您配置资源。您可以在可重用的 IaC 文件中描述基础设施。要使文件可以在不同的环境和应用程序中重复使用，您可以将它们编写为模板，定义输入参数，并在 IaC 定义中使用这些参数。稍后创建配置资源（环境、服务实例或组件）时，会 AWS Proton 使用渲染引擎，该引擎将输入值与模板组合在一起，以创建随时可以置备的 IaC 文件。

管理员将大多数模板创作为模板包，然后将其上传并注册到 AWS Proton。本页的其余部分将讨论这些 AWS Proton 模板包。直接定义的组件是一个例外；开发人员创建这些组件并直接提供 IaC 模板文件。有关组件的更多信息，请参阅组件。

主题

模板捆绑包

作为管理员，您可以创建和注册模板 AWS Proton。您可以使用这些模板创建环境和服务。创建服务时，会将服务实例置 AWS Proton 备并部署到选定的环境。有关更多信息，请参阅适用于平台团队的 AWS Proton。

要在中创建和注册模板 AWS Proton，您需要上传一个模板包，其中包含 AWS Proton 需要置备的基础架构即代码 (IaC) 文件以及环境或服务。

模板捆绑包 包含以下内容：

- 基础设施即代码 (IaC) 文件，其中包含列出 IaC 文件的清单 YAML 文件。
- IaC 文件输入参数定义的架构 YAML 文件。

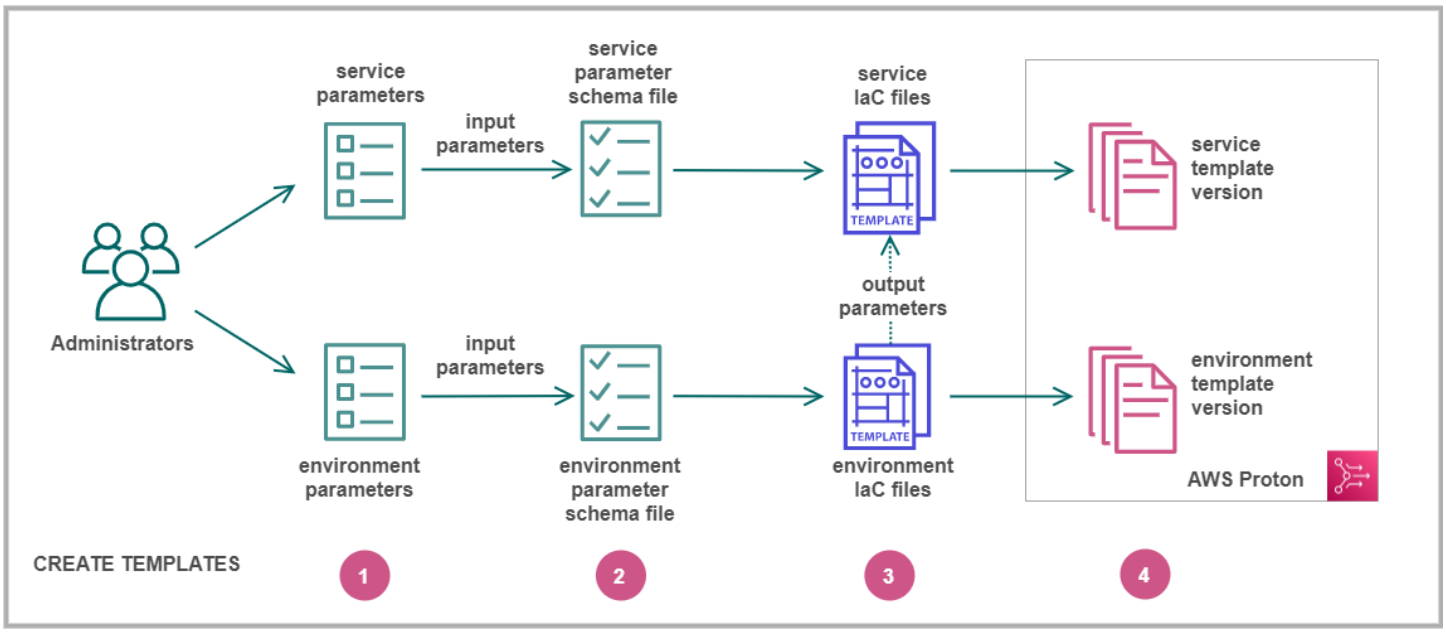
CloudFormation 环境模板包包含一个 IaC 文件。

CloudFormation 服务模板包包含一个用于服务实例定义的 IaC 文件和另一个用于管道定义的可选 IaC 文件。

Terraform 环境和服务模板捆绑包可以分别包含多个 IaC 文件。

AWS Proton 需要输入参数架构文件。当你使用 AWS CloudFormation 创建 IaC 文件时，你可以使用 Jinja 语法来引用你的输入参数。 AWS Proton 提供了可用于引用 IaC 文件中的参数的参数命名空间。

下图显示了为创建模板可以采取的步骤示例 AWS Proton。



1

指定输入参数。

2

创建架构文件以定义您的输入参数。

3

创建引用您的输入参数的 IaC 文件。您可以引用环境 IaC 文件输出 以作为服务 IaC 文件的输入。

4

注册模板版本 AWS Proton 并上传您的模板包。

向

AWS Proton 参数

您可以在基础设施即代码 (IaC) 文件中定义和使用参数，以使其灵活且可重用。您可以通过引用参数命名空间中的参数名称来读取 IaC 文件中的 AWS Proton 参数值。 AWS Proton 将参数值注入其在资源配置期间生成的渲染的 IaC 文件中。要处理 AWS CloudFormation IaC 参数，请 AWS Proton 使用 Jinja。要处理 Terraform iaC 参数，需要 AWS Proton 生成 Terraform 参数值文件，并依赖 HCL 内置的参数化功能。

使用CodeBuild 预置， AWS Proton 生成您的代码可以导入的输入文件。该文件是 JSON 或 HCL 文件，具体取决于模板清单中的属性。有关更多信息，请参阅the section called “CodeBuild 配置参数”。

您可以引用环境、服务和组件 IaC 文件或预置代码中的参数，并具有以下要求：

- 每个参数名称的长度不超过 100 个字符。
- 参数命名空间和资源名称的总长度不超过资源名称的字符限制。

AWS Proton 如果超过这些配额，则配置将失败。

参数类型

在 AWS Proton iaC 文件中，以下参数类型可供您参考：

输入参数

环境和服务实例可以使用与环境或服务模板关联的架构文件中定义的输入参数。您可以在资源的 IaC 文件中引用资源的输入参数。组件 IaC 文件可以引用组件附加到的服务实例的输入参数。

AWS Proton 根据架构文件检查输入参数名称，并将它们与 IaC 文件中引用的参数进行匹配，以注入您在资源配置期间在规范文件中提供的输入值。

输出参数

您可以在任何 IaC 文件中定义输出。例如，输出可以是模板预置的资源之一的名称、ID 或 ARN，也可以是传递模板输入之一的方式。您可以在其他资源的 IaC 文件中引用这些输出。

在 CloudFormation IaC 文件中，在`Outputs:`模块中定义输出参数。在 Terraform IaC 文件中，使用 `output` 语句定义每个输出参数。

资源参数

AWS Proton 自动创建 AWS Proton 资源参数。这些参数公开了 AWS Proton 资源对象的属性。一个资源参数示例是 `environment.name`。

在 IaC 文件中使用 AWS Proton 参数

要读取 IaC 文件中的参数值，请在参数命名空间中引用该 AWS Proton 参数的名称。对于 AWS CloudFormation IaC 文件，您可以使用 Jinja 语法，并用成对的大括号和引号将参数括起来。

下表显示了每种支持的模板语言的参考语法以及示例。

模板语言	语法	示例：名为“VPC”的环境输入
CloudFormation	`"{{ parameter-name }}"`	`"{{ environment.inputs.VPC }}"`
Terraform	`var.parameter-name`	`var.environment.inputs.VPC` 生成的 Terraform 变量定义

> **Note**
>
> 如果您在 IaC 文件中使用CloudFormation 动态参数，则必须对其进行转义以防止 Jinja 误解错误。有关更多信息，请参阅 排除 AWS Proton 的故障。

下表列出了所有 AWS Proton 资源参数的命名空间名称。每个模板文件类型可以使用参数命名空间的不同子集。

模板文件	参数类型	参数名称	描述
环境	资源	`environment.` **`name`**	环境名称
	input	`environment.inputs.` *`input-name`*	架构定义的环境输入
服务	资源	`environment.` **`name`** `environment.` **`account_id`**	环境名称和 AWS 账户 ID
	output	`environment.outputs.` *`output-name`*	环境 IaC 文件输出
	资源	`service.`**`branch_name`**	服务名称和代码存储库

模板文件	参数类型	参数名称	描述
		service.**name** service.**repository_connection_arn** service.**repository_id**	
	资源	service_instance. **name**	服务实例名称
	input	service_instance.inputs. *input-name*	架构定义的服务实例输入
	资源	service_instance.components.default. **name**	附加的默认组件名称
	output	service_instance.components.default.outputs. *output-name*	附加的默认组件 IaC 文件输出
管道	资源	service_instance.environment. **name** service_instance.environment. **account_id**	服务实例环境名称和 AWS 账户 ID
	output	service_instance.environment.outputs. *output-name*	服务实例环境 IaC 文件输出
	input	pipeline.inputs. *input-name*	架构定义的管道输入
	资源	service.**branch_name** service.**name** service.**repository_connection_arn** service.**repository_id**	服务名称和代码存储库

模板文件	参数类型	参数名称	描述
	input	`service_instance.inputs.` *`input-name`*	架构定义的服务实例输入
	collection	`{% for service_instance in` **`service_instances`** `%}...{% endfor %}`	您可以循环访问的服务实例的集合
组件	资源	`environment.` **`name`** `environment.` **`account_id`**	环境名称和 AWS 账户账户 ID
	output	`environment.outputs.` *`output-name`*	环境 IaC 文件输出
	资源	`service.`**`branch_name`** `service.`**`name`** `service.`**`repository_connection_arn`** `service.`**`repository_id`**	服务名称和代码存储库（附加的组件）
	资源	`service_instance.` **`name`**	服务实例名称（附加的组件）
	input	`service_instance.inputs.` *`input-name`*	架构定义的服务实例输入（附加的组件）
	资源	`component.` **`name`**	组件名称

有关更多信息和示例，请参阅有关不同资源类型和模板语言的 IaC 模板文件中的参数的子主题。

主题

- 环境 CloudFormation IaC 文件参数详细信息和示例
- 服务 CloudFormation IaC 文件参数详细信息和示例
- 组件 CloudFormation IaC 文件参数详细信息和示例

- CloudFormation IaC 文件的参数过滤器
- CodeBuild 配置参数详细信息和示例
- Terraform 基础设施即代码 (IaC) 文件参数详细信息和示例

环境 CloudFormation IaC 文件参数详细信息和示例

您可以在环境基础设施即代码 (IaC) 文件中定义和引用参数。有关参数、参数类型、 AWS Proton 参数命名空间以及如何在 IaC 文件中使用参数的详细说明，请参阅the section called “参数”。

定义环境参数

您可以为环境 IaC 文件定义输入和输出参数。

- 输入参数 - 在架构文件中定义环境输入参数。

  以下列表包括典型使用案例的环境输入参数示例。

  - VPC CIDR 值
  - 负载均衡器设置
  - 数据库设置
  - 运行状况检查超时

  作为管理员，您可以在创建环境时提供输入参数的值：

  - 使用控制台填写基于架构的表单，该 AWS Proton 表单提供了。
  - 使用 CLI 提供包含这些值的规范。
- 输出参数 - 在环境 IaC 文件中定义环境输出。然后，您可以在其他资源的 IaC 文件中引用这些输出。

读取环境 IaC 文件中的参数值

您可以在环境 IaC 文件中读取与环境相关的参数。您可以引用 AWS Proton 参数命名空间中的参数名称以读取参数值。

- 输入参数 - 引用 `environment.inputs.`*`input-name`* 以读取环境输入值。
- 资源参数-通过引用名称来读取 AWS Proton 资源参数，`environment.name`例如。

> **Note**
>
> 无法在环境 IaC 文件中使用其他资源的输出参数。

包含参数的示例环境和服务 IaC 文件

以下示例说明了环境 IaC 文件中的参数定义和引用。然后，该示例说明了如何在服务 IaC 文件中引用环境 IaC 文件中定义的环境输出参数。

Example 环境 CloudFormation IaC 文件

在该示例中，请注意以下事项：

- `environment.inputs.` 命名空间引用环境输入参数。
- Amazon EC2 Systems Manager (SSM) 参数 `StoreInputValue` 串联环境输入。
- `MyEnvParameterValue` 输出公开与输出参数相同的输入参数串联。三个额外的输出参数也单独公开输入参数。
- 6 个额外的输出参数公开环境预置的资源。

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

# These output values are available to service infrastructure as code files as outputs,
 when given the
# the 'environment.outputs' namespace, for example,
 service_instance.environment.outputs.ClusterName.
Outputs:
  MyEnvParameterValue:                                          # output definition
    Value: !GetAtt StoreInputValue.Value
  MySampleInputValue:                                           # output definition
    Value: "{{ environment.inputs.my_sample_input }}"           #   input parameter
 reference
```

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

Example 服务 CloudFormation IaC 文件

environment.outputs. 命名空间引用环境 IaC 文件的环境输出。例如，名称 environment.outputs.ClusterName 读取 ClusterName 环境输出参数的值。

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

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

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

  # The service_instance. The service is a resource which allows you to run multiple
```

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

服务 CloudFormation IaC 文件参数详细信息和示例

您可以在服务和管道基础设施即代码 (IaC) 文件中定义和引用参数。有关 AWS Proton 参数、参数类型、参数命名空间以及如何在 IaC 文件中使用参数的详细说明，请参阅the section called “参数”。

定义服务参数

您可以为服务 IaC 文件定义输入和输出参数。

- 输入参数 - 在架构文件中定义服务输入参数。

以下列表包括典型使用案例的服务输入参数示例。

- 端口
- 任务大小
- 图像
- 预期数量
- Docker 文件
- 单元测试命令

您可以在创建服务时提供输入参数的值：

- 使用控制台填写基于架构的表单，该 AWS Proton 表单提供了。
- 使用 CLI 提供包含这些值的规范。

- 输出参数 - 在服务 IaC 文件中定义服务实例输出。然后，您可以在其他资源的 IaC 文件中引用这些输出。

读取服务 IaC 文件中的参数值

您可以在服务 IaC 文件中读取与服务和其他资源相关的参数。您可以通过在参数命名空间中引用参数的名称来读取 AWS Proton 参数值。

- 输入参数 - 引用 `service_instance.inputs.`*`input-name`* 以读取服务实例输入值。
- 资源参数-通过引用`service.name`、`service_instance.name`和`environment.name`等名称来读取 AWS Proton 资源参数。
- 输出参数 - 引用 `environment.outputs.`*`output-name`* 或 `service_instance.components.default.outputs.`*`output-name`* 以读取其他资源的输出。

包含参数的示例服务 IaC 文件

以下示例是服务 CloudFormation IaC 文件中的片段。`environment.outputs.` 命名空间引用环境 IaC 文件的输出。`service_instance.inputs.` 命名空间引用服务实例输入参数。该`service_instance.name`属性指的是 AWS Proton 资源参数。

```
Resources:
  StoreServiceInstanceInputValue:
```

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

组件 CloudFormation IaC 文件参数详细信息和示例

您可以在组件基础设施即代码 (IaC) 文件中定义和引用参数。有关参数、参数类型、 AWS Proton 参数命名空间以及如何在 IaC 文件中使用参数的详细说明，请参阅the section called “参数”。有关组件的更多信息，请参阅组件。

定义组件输出参数

您可以在组件 IaC 文件中定义输出参数。然后，您可以在服务 IaC 文件中引用这些输出。

> **Note**
>
> 您无法为组件 IaC 文件定义输入。附加的组件可以从它们附加到的服务实例中获取输入。分离的组件没有输入。

读取组件 IaC 文件中的参数值

您可以在组件 IaC 文件中读取与组件和其他资源相关的参数。您可以通过在参数命名空间中引用参数的名称来读取 AWS Proton 参数值。

- 输入参数 - 引用 `service_instance.inputs.input-name` 以读取附加的服务实例输入值。
- 资源参数-通过引用`component.name`、`service.name`、`service_instance.name`、和`environment.name`等名称来读取 AWS Proton 资源参数。
- 输出参数 - 引用 `environment.outputs.output-name` 以读取环境输出。

包含参数的示例组件和服务 IaC 文件

以下示例显示一个组件，该组件预置 Amazon Simple Storage Service (Amazon S3) 存储桶和相关的访问策略，并将两种资源的 Amazon 资源名称 (ARN) 公开为组件输出。服务 IaC 模板将组件输出添加为 Amazon Elastic Container Service (Amazon ECS) 任务的容器环境变量以使输出可供容器中运行的代码使用，并将存储桶访问策略添加到任务的角色中。存储桶名称基于环境、服务、服务实例和组件的名称，这意味着存储桶与扩展特定服务实例的特定组件模板实例结合使用。开发人员可以根据该组件模板创建多个自定义组件，以针对不同的服务实例和功能需求预置 Amazon S3 存储桶。

该示例说明了如何使用 Jinja `{{ ... }}` 语法引用服务 IaC 文件中的组件和其他资源参数。只有在组件附加到服务实例时，您才能使用 `{% if ... %}` 语句添加语句块。`proton_cfn_*` 关键字是可用于清理输出参数值和设置参数值格式的筛选条件。有关筛选条件的更多信息，请参阅the section called “CloudFormation 参数过滤器”。

作为管理员，您编写服务 IaC 模板文件。

Example 使用组件的服务 CloudFormation IaC 文件

```
# service/instance_infrastructure/cloudformation.yaml

Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
```

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

  # ...

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

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

作为开发人员，您编写组件 IaC 模板文件。

Example 组件 CloudFormation iaC 文件

```
# cloudformation.yaml

# A component that defines an S3 bucket and a policy for accessing the bucket.
Resources:
  S3Bucket:
    Type: 'AWS::S3::Bucket'
    Properties:
      BucketName: '{{environment.name}}-{{service.name}}-{{service_instance.name}}-
{{component.name}}'
  S3BucketAccessPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
```

```
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Action:
              - 's3:Get*'
              - 's3:List*'
              - 's3:PutObject'
            Resource: !GetAtt S3Bucket.Arn
Outputs:
  BucketName:
    Description: "Bucket to access"
    Value: !GetAtt S3Bucket.Arn
  BucketAccessPolicyArn:
    Value: !Ref S3BucketAccessPolicy
```

为服务实例 AWS Proton 渲染 AWS CloudFormation 模板并将所有参数替换为实际值时，模板可能如下所示。

Example 服务实例 CloudFormation 呈现的 iaC 文件

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

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
```

```
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name

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

CloudFormation IaC 文件的参数过滤器

当您引用 AWS CloudFormation IaC 文件中的AWS Proton 参数时，您可以使用称为过滤器的 Jinja 修饰符来验证、筛选和格式化参数值，然后再将其插入到渲染的模板中。在引用组件输出参数时，筛选条件验证是特别有用的，因为组件创建和附加是由开发人员完成的，并且在服务实例模板中使用组件输出的管理员可能希望验证它们是否存在和有效。不过，您可以在任何 Jinja IaC 文件中使用筛选条件。

以下各节描述和定义了可用的参数筛选器，并提供了示例。 AWS Proton 定义了其中的大多数过滤器。`default` 筛选条件是 Jinja 内置筛选条件。

为 Amazon ECS 任务设置环境属性格式

声明

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

说明

该筛选条件为 Amazon Elastic Container Service (Amazon ECS) 任务定义 `ContainerDefinition` 部分的 Environment 属性中使用的输出列表设置格式。

还可以将 `raw` 设置为 `False` 以验证参数值。在这种情况下，该值需要与正则表达式 `^[a-zA-Z0-9_-]*$` 匹配。如果该值未通过验证，模板渲染将失败。

示例

如果使用以下自定义组件模板：

```
Resources:
  # ...
Outputs:
  Output1:
```

```
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
```

以及以下服务模板：

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      # ...
      ContainerDefinitions:
        - Name: MyServiceName
          # ...
          Environment:
            {{ service_instance.components.default.outputs
              | proton_cfn_ecs_task_definition_formatted_env_vars }}
```

渲染的服务模板如下所示：

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

设置 Lambda 函数的环境属性格式

声明

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

说明

此过滤器格式化了要在 AWS Lambda 函数定义Properties部分的 En vironment 属性中使用的输出列表。

还可以将 raw 设置为 False 以验证参数值。在这种情况下，该值需要与正则表达式 ^[a-zA-Z0-9_-]*$ 匹配。如果该值未通过验证，模板渲染将失败。

示例

如果使用以下自定义组件模板：

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

以及以下服务模板：

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

渲染的服务模板如下所示：

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

```
          Output2: world
```

提取 IAM policy ARN 以包含在 IAM 角色中

声明

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

说明

此筛选器格式化了要在 AWS Identity and Access Management (IAM) 角色定义`Properties`部分的ManagedPolicyArns 属性中使用的输出列表。该筛选条件使用正则表达式 `^arn:[a-zA-Z-]+:iam::\d{12}:policy/` 从输出参数列表中提取有效的 IAM policy ARN。您可以使用该筛选条件，将输出参数值中的策略附加到服务模板中的 IAM 角色定义。

示例

如果使用以下自定义组件模板：

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

  # ...

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

```
    Description: "ARN of policy 2"
    Value: !Ref ExamplePolicy2
```

以及以下服务模板：

```
Resources:

  # ...

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

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

渲染的服务模板如下所示：

```
Resources:

  # ...

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

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

```
      # ...
```

清理属性值

声明

```
string # proton_cfn_sanitize # string
```

说明

这是一个通用的筛选条件。可以使用该筛选条件验证参数值的安全性。该筛选条件验证值是否与正则表达式 ^[a-zA-Z0-9_-]*$ 匹配或者是有效的 Amazon 资源名称 (ARN)。如果该值未通过验证，模板渲染将失败。

示例

如果使用以下自定义组件模板：

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

- 服务模板中的以下引用：

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

  渲染如下所示：

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

- 服务模板中的以下引用：

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

包含以下渲染错误的结果：

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

- 服务模板中的以下引用：

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

渲染如下所示：

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

为不存在的引用提供默认值

描述

在命名空间引用不存在时，default 筛选条件提供默认值。可以使用它编写强大的模板，即使您引用的参数丢失，也可以成功进行渲染。

示例

如果服务实例没有附加直接定义的（默认）组件，或者附加的组件没有名为 test 的输出，服务模板中的以下引用将导致模板渲染失败。

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

要避免该问题，请添加 default 筛选条件。

```
# ...
```

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

CodeBuild 配置参数详细信息和示例

您可以在模板中为 CodeBuild基于 AWS Proton 资源的定义参数，并在配置代码中引用这些参数。有关参数、参数类型、 AWS Proton 参数命名空间以及如何在 IaC 文件中使用参数的详细说明，请参阅the section called “参数”。

> **Note**
>
> 您可以对环境和服务使用 CodeBuild 置备。目前，您无法通过这种方法预置组件。

输入参数

创建 AWS Proton 资源（如环境或服务）时，需要为模板架构文件中定义的输入参数提供值。当您创建的资源使用时CodeBuild 预置，会将这些输入值 AWS Proton 呈现到输入文件中。您的预置代码可以从该文件中导入和获取参数值。

有关 CodeBuild 模板的示例，请参见the section called “CodeBuild 捆绑包”。有关清单文件的更多信息，请参阅the section called “清单和打包”。

以下示例是在 CodeBuild基于服务实例的配置期间生成的 JSON 输入文件。

示例：使用 AWS CDK 带 CodeBuild 配置的

```
{
  "service_instance": {
    "name": "my-service-staging",
    "inputs": {
      "port": "8080",
      "task_size": "medium"
    }
  },
  "service": {
    "name": "my-service"
  },
  "environment": {
    "account_id": "123456789012",
    "name": "my-env-staging",
    "outputs": {
      "vpc-id": "hdh2323423"
```

```
    }
  }
}
```

输出参数

要将资源配置输出传回给 AWS Proton，您的配置代码可以生成一个名为的 JSON 文件，该文件名`proton-outputs.json`为模板架构文件中定义的输出参数的值。例如，该`cdk deploy`命令的`--outputs-file`参数指示生成包含配置输出的 AWS CDK JSON 文件。如果您的资源使用 AWS CDK，请在 CodeBuild 模板清单中指定以下命令：

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

AWS Proton 正在寻找这个 JSON 文件。如果配置代码成功完成后文件存在，则从中 AWS Proton 读取输出参数值。

Terraform 基础设施即代码 (IaC) 文件参数详细信息和示例

您可以将 Terraform 输入变量包含在模板捆绑包的 `variable.tf` 文件中。您也可以创建架构来创建 AWS Proton 托管变量。 AWS Proton `.tf files`从您的架构文件中创建变量。有关更多信息，请参阅the section called “Terraform IaC 文件”。

要在基础架构中引用架构定义的 AWS Proton 变量`.tf files`，请使用 Terrafor
AWS Proton m iaC 的参数和命名空间表中显示的命名空间。例如，您可以使用 `var.environment.inputs.vpc_cidr`。在引号内，用单大括号将这些变量括起来，并在第一个大括号前面添加一个美元符号（例如“`${var.environment.inputs.vpc_cidr}`”）。

以下示例说明如何使用命名空间在环境中包含 AWS Proton 参数。`.tf file`

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

```
    region = "us-east-1"
  }
}

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

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

AWS Proton 基础架构即代码文件

模板包的主要部分是基础设施即代码 (IaC) 文件，用于定义要置备的基础架构资源和属性。 AWS CloudFormation 以及其他基础设施即代码引擎使用这些类型的文件来配置基础架构资源。

> **Note**
>
> 也可以独立于模板捆绑包使用 IaC 文件，以作为直接定义的组件 的直接输入。有关组件的更多信息，请参阅组件。

AWS Proton 目前支持两种类型的 IaC 文件：

- CloudFormation文件-用于AWS托管配置。 AWS Proton 在 CloudFormation 模板文件格式之上使用 Jinja 进行参数化。
- Terraform HCL 文件 - 用于自托管式预置。HCL 本身支持参数化。

您不能使用多种配置方法来配置 AWS Proton 资源。您必须使用其中的一种方法。您无法将 AWS托管配置服务部署到自我管理的配置环境，反之亦然。

有关更多信息，请参阅 the section called “预置方法”、环境、服务 和 组件。

AWS CloudFormation IaC 文件

通过学习如何使用 AWS CloudFormation 基础架构作为代码文件 AWS Proton。 AWS CloudFormation 是一项基础设施即代码 (IaC) 服务，可帮助您对 AWS 资源进行建模和设置。您可以在模板中定义基础架构资源，在 CloudFormation 模板文件格式之上使用 Jinja 进行参数化。 AWS Proton 展开参数并呈现完整 CloudFormation 模板。 CloudFormation 将定义的资源配置为 CloudFormation 堆栈。有关更多信息，请参阅《 AWS CloudFormation 用户指南》中的什么是 AWS CloudFormation？

AWS Proton 支持 AWS CloudFormation IaC 的托管配置。

从您自己的现有基础设施即代码文件开始

您可以调整自己现有的基础设施即代码 (IaC) 文件以供使用 AWS Proton。

以下 AWS CloudFormation 示例（示例 1 和示例 2）代表您自己的现有 CloudFormation IaC 文件。CloudFormation 可以使用这些文件创建两个不同的 CloudFormation堆栈。

在示例 1 中，将 CloudFormation IaC 文件配置为配置基础架构，以便在容器应用程序之间共享。在该示例中，添加了输入参数，以便使用相同的 IaC 文件创建多组预置的基础设施。每个集合可以具有不同的名称以及不同的 VPC 和子网 CIDR 值集。作为管理员或开发人员，在使用 IaC 文件配置基础设施资源时，您需要为这些参数提供值。 CloudFormation为了方便起见，这些输入参数标有注释，并在该示例中多次引用。输出是在模板末尾定义的。它们可以在其他 CloudFormation IaC 文件中引用。

在示例 2 中，将 CloudFormation IaC 文件配置为将应用程序部署到示例 1 中配置的基础架构。为了方便起见，对这些参数进行了注释。

示例 1： CloudFormation IaC 文件

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

```
        Type: String
        Default: "10.0.1.0/24"
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: true
      CidrBlock:
        Ref: 'VpcCIDR'

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

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

  # Setup networking resources for the public subnets. Containers
  # in the public subnets have public IP addresses and the routing table
  # sends network traffic via the internet gateway.
  InternetGateway:
    Type: AWS::EC2::InternetGateway
  GatewayAttachement:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
```

```
      VpcId: !Ref 'VPC'
      InternetGatewayId: !Ref 'InternetGateway'
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref 'VPC'
  PublicRoute:
    Type: AWS::EC2::Route
    DependsOn: GatewayAttachement
    Properties:
      RouteTableId: !Ref 'PublicRouteTable'
      DestinationCidrBlock: '0.0.0.0/0'
      GatewayId: !Ref 'InternetGateway'
  PublicSubnetOneRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetOne
      RouteTableId: !Ref PublicRouteTable
  PublicSubnetTwoRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetTwo
      RouteTableId: !Ref PublicRouteTable

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

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

  # This is a role which is used by the ECS tasks themselves.
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Effect: Allow
```

```
          Principal:
            Service: [ecs-tasks.amazonaws.com]
          Action: ['sts:AssumeRole']
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy'

# These output values will be available to other templates to use.
Outputs:
  ClusterName:                                              # output
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ECSCluster"
  ECSTaskExecutionRole:                                      # output
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ECSTaskExecutionRole"
  VpcId:                                                     # output
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-VPC"
  PublicSubnetOne:                                           # output
    Description: Public subnet one
    Value: !Ref 'PublicSubnetOne'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-PublicSubnetOne"
  PublicSubnetTwo:                                           # output
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-PublicSubnetTwo"
  ContainerSecurityGroup:                                    # output
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'
    Export:
      Name:
```

```
        Fn::Sub: "${AWS::StackName}-ContainerSecurityGroup"
```

示例 2： CloudFormation IaC 文件

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Parameters:
    ContainerPortInput:  # input parameter
        Description: The port to route traffic to
        Type: Number
        Default: 80
    TaskCountInput:      # input parameter
        Description: The default number of Fargate tasks you want running
        Type: Number
        Default: 1
    TaskSizeInput:       # input parameter
        Description: The size of the task you want to run
        Type: String
        Default: x-small
    ContainerImageInput: # input parameter
        Description: The name/url of the container image
        Type: String
        Default: "public.ecr.aws/z9d2n7e1/nginx:1.19.5"
    TaskNameInput:       # input parameter
        Description: Name for your task
        Type: String
        Default: "my-fargate-instance"
    StackName:           # input parameter
        Description: Name of the environment stack to deploy to
        Type: String
        Default: "my-fargate-environment"
Mappings:
  TaskSizeMap:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
```

```
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName:
        Ref: 'TaskNameInput' # input parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Ref 'TaskNameInput'
      Cpu: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', cpu]
      Memory: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', memory]
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn:
        Fn::ImportValue:
          !Sub "${StackName}-ECSTaskExecutionRole"    # output parameter from another
 CloudFormation template
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: !Ref 'TaskNameInput'


  # The service_instance. The service is a resource which allows you to run multiple
  # copies of a type of task, and gather up their logs and metrics, as well
  # as monitor the number of running tasks and replace any that have crashed
  Service:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerRule
    Properties:
      ServiceName: !Ref 'TaskNameInput'
      Cluster:
        Fn::ImportValue:
```

```
          !Sub "${StackName}-ECSCluster"  # output parameter from another
CloudFormation template
     LaunchType: FARGATE
     DeploymentConfiguration:
       MaximumPercent: 200
       MinimumHealthyPercent: 75
     DesiredCount: !Ref 'TaskCountInput'
     NetworkConfiguration:
       AwsvpcConfiguration:
         AssignPublicIp: ENABLED
         SecurityGroups:
           - Fn::ImportValue:
               !Sub "${StackName}-ContainerSecurityGroup"    # output parameter from
another CloudFormation template
         Subnets:
           - Fn::ImportValue:r CloudFormation template
     TaskRoleArn: !Ref "AWS::NoValue"
     ContainerDefinitions:
       - Name: !Ref 'TaskNameInput'
         Cpu: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', cpu]
         Memory: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', memory]
         Image: !Ref 'ContainerImageInput'             # input parameter
         PortMappings:
           - ContainerPort: !Ref 'ContainerPortInput'  # input parameter

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

 # A target group. This is used for keeping track of all the tasks, and
 # what IP addresses / port numbers they have. You can query it yourself,
 # to use the addresses yourself, but most often this target group is just
 # connected to an application load balancer, or network load balancer, so
```

```
  # it can automatically distribute traffic across all the targets.
  TargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      HealthCheckIntervalSeconds: 6
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      TargetType: ip
      Name: !Ref 'TaskNameInput'
      Port: !Ref 'ContainerPortInput'
      Protocol: HTTP
      UnhealthyThresholdCount: 2
      VpcId:
        Fn::ImportValue:
          !Sub "${StackName}-VPC"    # output parameter from another CloudFormation
template

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

  # Enable autoscaling for this service
  ScalableTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    DependsOn: Service
    Properties:
      ServiceNamespace: 'ecs'
      ScalableDimension: 'ecs:service:DesiredCount'
      ResourceId:
        Fn::Join:
          - '/'
          - - service
```

```
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"
            - !Ref 'TaskNameInput'
      MinCapacity: 1
      MaxCapacity: 10
      RoleARN: !Sub arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService

  # Create scaling policies for the service
  ScaleDownPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - !Ref 'TaskNameInput'
            - down
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"  # output parameter from another
 CloudFormation template
            - !Ref 'TaskNameInput'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalUpperBound: 0
            ScalingAdjustment: -1
        MetricAggregationType: 'Average'
        Cooldown: 60

  ScaleUpPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
```

```
        Fn::Join:
          - '/'
          - - scale
            - !Ref 'TaskNameInput'
            - up
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"
            - !Ref 'TaskNameInput'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalLowerBound: 0
            MetricIntervalUpperBound: 15
            ScalingAdjustment: 1
          - MetricIntervalLowerBound: 15
            MetricIntervalUpperBound: 25
            ScalingAdjustment: 2
          - MetricIntervalLowerBound: 25
            ScalingAdjustment: 3
        MetricAggregationType: 'Average'
        Cooldown: 60

  # Create alarms to trigger these policies
  LowCpuUsageAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName:
        Fn::Join:
          - '-'
          - - low-cpu
            - !Ref 'TaskNameInput'
      AlarmDescription:
        Fn::Join:
          - ' '
          - - "Low CPU utilization for service"
            - !Ref 'TaskNameInput'
      MetricName: CPUUtilization
```

```
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: !Ref 'TaskNameInput'
        - Name: ClusterName
          Value:
            Fn::ImportValue:
              !Sub "${StackName}-ECSCluster"
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 20
      ComparisonOperator: LessThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleDownPolicy

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

```
        - !Ref ScaleUpPolicy

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

  # Public load balancer, hosted in public subnets that is accessible
  # to the public, and is intended to route traffic to one or more public
  # facing services. This is used for accepting traffic from the public
  # internet and directing it to public facing microservices
  PublicLoadBalancerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the public facing load balancer
      VpcId:
        Fn::ImportValue:
          !Sub "${StackName}-VPC"
      SecurityGroupIngress:
          # Allow access to ALB from anywhere on the internet
          - CidrIp: 0.0.0.0/0
            IpProtocol: -1

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

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

您可以调整这些文件以供使用 AWS Proton。

将您的基础架构即代码带到 AWS Proton

只需稍作修改，即可将示例 1 用作用于部署环境的环境模板包的基础架构即 AWS Proton 代码 (IaC) 文件（如示例 3 所示）。

您可以使用 Jinja 语法来引用在基于 Ope n API 的架构文件中定义的参数，而不是使用参数。CloudFormation 为了方便起见，对这些输入参数进行了注释，并在 IaC 文件中多次引用这些参数。这样， AWS Proton 就可以审计和检查参数值。它也可以将一个 IaC 文件中的输出参数值与另一个 IaC 文件中的参数进行匹配和插入。

作为管理员，您可以将 AWS Proton `environment.inputs.`命名空间添加到输入参数中。在服务 IaC 文件中引用环境 IaC 文件输出时，您可以将 `environment.outputs.` 命名空间添加到输出中（例如 `environment.outputs.ClusterName`）。最后，用大括号和引号将它们括起来和引起来。

通过这些修改，您的 CloudFormation IaC 文件就可以被使用了。 AWS Proton

示例 3： AWS Proton 环境基础架构作为代码文件

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS Fargate cluster running containers in a public subnet. Only supports
             public facing load balancer, and public service discovery prefixes.
Mappings:
  # The VPC and subnet configuration is passed in via the environment spec.
```

```
  SubnetConfig:
    VPC:
      CIDR: '{{ environment.inputs.vpc_cidr}}'        # input parameter
    PublicOne:
      CIDR: '{{ environment.inputs.subnet_one_cidr}}' # input parameter
    PublicTwo:
      CIDR: '{{ environment.inputs.subnet_two_cidr}}' # input parameter
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: true
      CidrBlock: !FindInMap ['SubnetConfig', 'VPC', 'CIDR']

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

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

  # Setup networking resources for the public subnets. Containers
  # in the public subnets have public IP addresses and the routing table
  # sends network traffic via the internet gateway.
  InternetGateway:
    Type: AWS::EC2::InternetGateway
  GatewayAttachement:
```

```
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref 'VPC'
      InternetGatewayId: !Ref 'InternetGateway'
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref 'VPC'
  PublicRoute:
    Type: AWS::EC2::Route
    DependsOn: GatewayAttachement
    Properties:
      RouteTableId: !Ref 'PublicRouteTable'
      DestinationCidrBlock: '0.0.0.0/0'
      GatewayId: !Ref 'InternetGateway'
  PublicSubnetOneRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetOne
      RouteTableId: !Ref PublicRouteTable
  PublicSubnetTwoRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetTwo
      RouteTableId: !Ref PublicRouteTable

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

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

  # This is a role which is used by the ECS tasks themselves.
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
```

```
        Statement:
        - Effect: Allow
          Principal:
            Service: [ecs-tasks.amazonaws.com]
          Action: ['sts:AssumeRole']
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy'

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

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

示例 1 和示例 3 中的 IaC 文件生成的 CloudFormation 堆栈略有不同。参数是在堆栈模板文件中以不同方式显示的。示例 1 CloudFormation 堆栈模板文件在堆栈模板视图中显示参数标签（密钥）。示例 3 AWS Proton CloudFormation 基础设施堆栈模板文件显示了参数值。 AWS Proton 输入参数不会出现在控制台 CloudFormation堆栈参数视图中。

在示例 4 中， AWS Proton 服务 IaC 文件与示例 2 相对应。

示例 4： AWS Proton 服务实例 IaC 文件

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Mappings:
  TaskSize:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: '{{service_instance.name}}' # resource parameter

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

```
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu]
          Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
          Image: '{{service_instance.inputs.image}}'
          PortMappings:
            - ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          LogConfiguration:
            LogDriver: 'awslogs'
            Options:
              awslogs-group: '{{service_instance.name}}'
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: '{{service_instance.name}}'

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

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

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

  # Enable autoscaling for this service
  ScalableTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    DependsOn: Service
    Properties:
      ServiceNamespace: 'ecs'
      ScalableDimension: 'ecs:service:DesiredCount'
      ResourceId:
```

```
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}' # output from an environment
 infrastructure as code file
            - '{{service_instance.name}}'
      MinCapacity: 1
      MaxCapacity: 10
      RoleARN: !Sub arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService

  # Create scaling policies for the service
  ScaleDownPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - '{{service_instance.name}}'
            - down
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}'
            - '{{service_instance.name}}'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalUpperBound: 0
            ScalingAdjustment: -1
        MetricAggregationType: 'Average'
        Cooldown: 60

  ScaleUpPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
```

```
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - '{{service_instance.name}}'
            - up
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}'
            - '{{service_instance.name}}'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalLowerBound: 0
            MetricIntervalUpperBound: 15
            ScalingAdjustment: 1
          - MetricIntervalLowerBound: 15
            MetricIntervalUpperBound: 25
            ScalingAdjustment: 2
          - MetricIntervalLowerBound: 25
            ScalingAdjustment: 3
        MetricAggregationType: 'Average'
        Cooldown: 60

  # Create alarms to trigger these policies
  LowCpuUsageAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName:
        Fn::Join:
          - '-'
          - - low-cpu
            - '{{service_instance.name}}'
      AlarmDescription:
        Fn::Join:
          - ' '
          - - "Low CPU utilization for service"
            - '{{service_instance.name}}'
      MetricName: CPUUtilization
```

```
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: '{{service_instance.name}}'
        - Name: ClusterName
          Value:
            '{{environment.outputs.ClusterName}}'
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 20
      ComparisonOperator: LessThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleDownPolicy

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

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

  # Public load balancer, hosted in public subnets that is accessible
  # to the public, and is intended to route traffic to one or more public
  # facing services. This is used for accepting traffic from the public
  # internet and directing it to public facing microservices
  PublicLoadBalancerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the public facing load balancer
      VpcId: '{{environment.outputs.VpcId}}'
      SecurityGroupIngress:
          # Allow access to ALB from anywhere on the internet
          - CidrIp: 0.0.0.0/0
            IpProtocol: -1

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

  PublicLoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    DependsOn:
      - PublicLoadBalancer
    Properties:
      DefaultActions:
        - TargetGroupArn: !Ref 'TargetGroup'
```

```
          Type: 'forward'
      LoadBalancerArn: !Ref 'PublicLoadBalancer'
      Port: 80
      Protocol: HTTP
Outputs:
  ServiceEndpoint:         # output
    Description: The URL to access the service
    Value: !Sub "http://${PublicLoadBalancer.DNSName}"
```

在示例 5 中，AWS Proton 管道 IaC 文件配置管道基础设施以支持示例 4 中配置的服务实例。

示例 5：AWS Proton 服务管道 IaC 文件

```
Resources:
  ECRRepo:
    Type: AWS::ECR::Repository
    DeletionPolicy: Retain
  BuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:
        Type: CODEPIPELINE
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux2-x86_64-standard:3.0
        PrivilegedMode: true
        Type: LINUX_CONTAINER
        EnvironmentVariables:
        - Name: repo_name
          Type: PLAINTEXT
          Value: !Ref ECRRepo
        - Name: service_name
          Type: PLAINTEXT
          Value: '{{ service.name }}'    # resource parameter
      ServiceRole:
        Fn::GetAtt:
          - PublishRole
          - Arn
      Source:
        BuildSpec:
          Fn::Join:
            - ""
            - - >-
                {
```

```
                  "version": "0.2",
                  "phases": {
                    "install": {
                      "runtime-versions": {
                        "docker": 18
                      },
                      "commands": [
                        "pip3 install --upgrade --user awscli",
                        "echo
 'f6bd1536a743ab170b35c94ed4c7c4479763356bd543af5d391122f4af852460  yq_linux_amd64' >
 yq_linux_amd64.sha",
                        "wget https://github.com/mikefarah/yq/releases/download/3.4.0/
yq_linux_amd64",
                        "sha256sum -c yq_linux_amd64.sha",
                        "mv yq_linux_amd64 /usr/bin/yq",
                        "chmod +x /usr/bin/yq"
                      ]
                    },
                    "pre_build": {
                      "commands": [
                        "cd $CODEBUILD_SRC_DIR",
                        "$(aws ecr get-login --no-include-email --region
 $AWS_DEFAULT_REGION)",
                        "{{ pipeline.inputs.unit_test_command }}",    # input parameter
                      ]
                    },
                    "build": {
                      "commands": [
                        "IMAGE_REPO_NAME=$repo_name",
                        "IMAGE_TAG=$CODEBUILD_BUILD_NUMBER",
                        "IMAGE_ID=
              - Ref: AWS::AccountId
              - >-
                .dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:
$IMAGE_TAG",
                        "docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG -f
 {{ pipeline.inputs.dockerfile }} .",     # input parameter
                        "docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $IMAGE_ID;",
                        "docker push $IMAGE_ID"
                      ]
                    },
                    "post_build": {
                      "commands": [
```

```
                        "aws proton --region $AWS_DEFAULT_REGION get-service --name
 $service_name | jq -r .service.spec > service.yaml",
                        "yq w service.yaml 'instances[*].spec.image' \"$IMAGE_ID\" >
 rendered_service.yaml"
                      ]
                    }
                  },
                  "artifacts": {
                    "files": [
                      "rendered_service.yaml"
                    ]
                  }
                }
        Type: CODEPIPELINE
      EncryptionKey:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% for service_instance in service_instances %}
  Deploy{{loop.index}}Project:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:
        Type: CODEPIPELINE
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux2-x86_64-standard:3.0
        PrivilegedMode: false
        Type: LINUX_CONTAINER
        EnvironmentVariables:
        - Name: service_name
          Type: PLAINTEXT
          Value:  '{{service.name}}'          # resource parameter
        - Name: service_instance_name
          Type: PLAINTEXT
          Value: '{{service_instance.name}}'  # resource parameter
      ServiceRole:
        Fn::GetAtt:
          - DeploymentRole
          - Arn
      Source:
        BuildSpec: >-
          {
            "version": "0.2",
```

```
            "phases": {
              "build": {
                "commands": [
                  "pip3 install --upgrade --user awscli",
                  "aws proton --region $AWS_DEFAULT_REGION update-service-instance
 --deployment-type CURRENT_VERSION --name $service_instance_name --service-name
 $service_name --spec file://rendered_service.yaml",
                  "aws proton --region $AWS_DEFAULT_REGION wait service-instance-
deployed --name $service_instance_name --service-name $service_name"
                ]
              }
            }
          }
        Type: CODEPIPELINE
      EncryptionKey:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% endfor %}
  # This role is used to build and publish an image to ECR
  PublishRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  PublishRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
```

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

```
            - ecr:InitiateLayerUpload
            - ecr:PutImage
            - ecr:UploadLayerPart
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - ECRRepo
              - Arn
        - Action:
            - proton:GetService
          Effect: Allow
          Resource: "*"
        - Action:
            - s3:GetObject*
            - s3:GetBucket*
            - s3:List*
            - s3:DeleteObject*
            - s3:PutObject*
            - s3:Abort*
          Effect: Allow
          Resource:
            - Fn::GetAtt:
                - PipelineArtifactsBucket
                - Arn
            - Fn::Join:
                - ""
                - - Fn::GetAtt:
                      - PipelineArtifactsBucket
                      - Arn
                  - /*
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
        - Action:
            - kms:Decrypt
            - kms:Encrypt
```

```
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
      Version: "2012-10-17"
    PolicyName: PublishRoleDefaultPolicy
    Roles:
      - Ref: PublishRole

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

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

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

```
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - PipelineRole
                  - Arn
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - PublishRole
                  - Arn
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
```

```
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
      Version: "2012-10-17"
  UpdateReplacePolicy: Delete
  DeletionPolicy: Delete
PipelineArtifactsBucket:
  Type: AWS::S3::Bucket
  Properties:
    VersioningConfiguration:
      Status: Enabled
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            KMSMasterKeyID:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
            SSEAlgorithm: aws:kms
    PublicAccessBlockConfiguration:
      BlockPublicAcls: true
```

```
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  PipelineArtifactsBucketEncryptionKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
      AliasName: 'alias/codepipeline-encryption-key-{{ service.name }}'
      TargetKeyId:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codepipeline.amazonaws.com
        Version: "2012-10-17"
  PipelineRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
```

```
                - - Fn::GetAtt:
                      - PipelineArtifactsBucket
                      - Arn
                  - /*
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
        - Action: codestar-connections:*
          Effect: Allow
          Resource: "*"
        - Action: sts:AssumeRole
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - PipelineBuildCodePipelineActionRole
              - Arn
        - Action: sts:AssumeRole
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - PipelineDeployCodePipelineActionRole
              - Arn
      Version: "2012-10-17"
    PolicyName: PipelineRoleDefaultPolicy
    Roles:
      - Ref: PipelineRole
Pipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn:
      Fn::GetAtt:
        - PipelineRole
        - Arn
    Stages:
      - Actions:
          - ActionTypeId:
```

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

```
              Name: Deploy
              RoleArn:
                Fn::GetAtt:
                  - PipelineDeployCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: 'Deploy{{service_instance.name}}'
{%- endfor %}
      ArtifactStore:
        EncryptionKey:
          Id:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
          Type: KMS
        Location:
          Ref: PipelineArtifactsBucket
        Type: S3
    DependsOn:
      - PipelineRoleDefaultPolicy
      - PipelineRole
  PipelineBuildCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineBuildCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
```

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

```
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - ":project/Deploy*"
        Version: "2012-10-17"
      PolicyName: PipelineDeployCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineDeployCodePipelineActionRole
Outputs:
  PipelineEndpoint:
    Description: The URL to access the pipeline
    Value: !Sub "https://${AWS::Region}.console.aws.amazon.com/codesuite/codepipeline/
pipelines/${Pipeline}/view?region=${AWS::Region}"

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

```
            - logs:PutLogEvents
          Effect: Allow
          Resource:
            - Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":logs:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :log-group:/aws/codebuild/
                  - Ref: BuildProject
            - Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":logs:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :log-group:/aws/codebuild/
                  - Ref: BuildProject
                  - :*
        - Action:
            - codebuild:CreateReportGroup
            - codebuild:CreateReport
            - codebuild:UpdateReport
            - codebuild:BatchPutTestCases
          Effect: Allow
          Resource:
            Fn::Join:
              - ""
              - - "arn:"
                - Ref: AWS::Partition
                - ":codebuild:"
                - Ref: AWS::Region
                - ":"
                - Ref: AWS::AccountId
                - :report-group/
                - Ref: BuildProject
                - -*
        - Action:
            - ecr:GetAuthorizationToken
```

```
      Effect: Allow
      Resource: "*"
    - Action:
        - ecr:BatchCheckLayerAvailability
        - ecr:CompleteLayerUpload
        - ecr:GetAuthorizationToken
        - ecr:InitiateLayerUpload
        - ecr:PutImage
        - ecr:UploadLayerPart
      Effect: Allow
      Resource:
        Fn::GetAtt:
          - ECRRepo
          - Arn
    - Action:
        - proton:GetService
      Effect: Allow
      Resource: "*"
    - Action:
        - s3:GetObject*
        - s3:GetBucket*
        - s3:List*
        - s3:DeleteObject*
        - s3:PutObject*
        - s3:Abort*
      Effect: Allow
      Resource:
        - Fn::GetAtt:
            - PipelineArtifactsBucket
            - Arn
        - Fn::Join:
            - ""
            - - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - /*
    - Action:
        - kms:Decrypt
        - kms:DescribeKey
        - kms:Encrypt
        - kms:ReEncrypt*
        - kms:GenerateDataKey*
      Effect: Allow
      Resource:
```

```
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: PublishRoleDefaultPolicy
      Roles:
        - Ref: PublishRole

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

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

```
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: DeploymentRoleDefaultPolicy
      Roles:
        - Ref: DeploymentRole
  PipelineArtifactsBucketEncryptionKey:
    Type: AWS::KMS::Key
    Properties:
      KeyPolicy:
        Statement:
          - Action:
              - kms:Create*
              - kms:Describe*
              - kms:Enable*
              - kms:List*
              - kms:Put*
              - kms:Update*
              - kms:Revoke*
              - kms:Disable*
              - kms:Get*
              - kms:Delete*
```

```
            - kms:ScheduleKeyDeletion
            - kms:CancelKeyDeletion
            - kms:GenerateDataKey
            - kms:TagResource
            - kms:UntagResource
          Effect: Allow
          Principal:
            AWS:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":iam::"
                  - Ref: AWS::AccountId
                  - :root
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PipelineRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
```

```
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
      Version: "2012-10-17"
  UpdateReplacePolicy: Delete
  DeletionPolicy: Delete
PipelineArtifactsBucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            KMSMasterKeyID:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
```

```
                - Arn
            SSEAlgorithm: aws:kms
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  PipelineArtifactsBucketEncryptionKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
      AliasName: 'alias/codepipeline-encryption-key-{{ service.name }}'     # resource
parameter
      TargetKeyId:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codepipeline.amazonaws.com
        Version: "2012-10-17"
  PipelineRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
            Resource:
```

```
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action: codestar-connections:*
            Effect: Allow
            Resource: "*"
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineBuildCodePipelineActionRole
                - Arn
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineDeployCodePipelineActionRole
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineRoleDefaultPolicy
      Roles:
        - Ref: PipelineRole
  Pipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      RoleArn:
        Fn::GetAtt:
```

```
          - PipelineRole
          - Arn
      Stages:
        - Actions:
            - ActionTypeId:
                Category: Source
                Owner: AWS
                Provider: CodeStarSourceConnection
                Version: "1"
              Configuration:
                ConnectionArn: '{{ service.repository_connection_arn }}'   # resource
parameter
                FullRepositoryId: '{{ service.repository_id }}'            # resource
parameter
                BranchName: '{{ service.branch_name }}'                    # resource
parameter
              Name: Checkout
              OutputArtifacts:
                - Name: Artifact_Source_Checkout
              RunOrder: 1
          Name: Source
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: BuildProject
              InputArtifacts:
                - Name: Artifact_Source_Checkout
              Name: Build
              OutputArtifacts:
                - Name: BuildOutput
              RoleArn:
                Fn::GetAtt:
                  - PipelineBuildCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: Build {%- for service_instance in service_instances %}
        - Actions:
            - ActionTypeId:
                Category: Build
```

```
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: Deploy{{loop.index}}Project
              InputArtifacts:
                - Name: BuildOutput
              Name: Deploy
              RoleArn:
                Fn::GetAtt:
                  - PipelineDeployCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: 'Deploy{{service_instance.name}}'         # resource parameter
{%- endfor %}
      ArtifactStore:
        EncryptionKey:
          Id:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
          Type: KMS
        Location:
          Ref: PipelineArtifactsBucket
        Type: S3
    DependsOn:
      - PipelineRoleDefaultPolicy
      - PipelineRole
  PipelineBuildCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
```

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

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

CodeBuild 配置模板包

通过 CodeBuild 配置，无需使用 IaC 模板来渲染 IaC 文件并使用 IaC 配置引擎运行它们， AWS Proton 只需运行 shell 命令即可。为此，请在环境帐户中为环境 AWS Proton 创建一个 AWS CodeBuild 项目，然后启动一个作业，在每次创建或更新 AWS Proton 资源时运行您的命令。在您编写模板捆绑包时，您需要提供一个清单，以指定基础设施预置和取消预置命令，以及这些命令可能需要的任何程序、脚本和其他文件。您的命令可以读取 AWS Proton 提供的输入，并负责预置或取消预置基础设施和生成输出值。

清单还指定了 AWS Proton 应如何呈现您的代码可以输入并从中获取输入值的输入文件。可以将该文件渲染为 JSON 或 HCL。有关输入参数的更多信息，请参阅the section called “CodeBuild 配置参数”。有关清单文件的更多信息，请参阅the section called “清单和打包”。

> **Note**
>
> 您可以对环境和服务使用 CodeBuild 置备。目前，您无法通过这种方法预置组件。

示例：使用 AWS CDK 带 CodeBuild 配置的

作为使用 CodeBuild 预配的示例，您可以包括使用预配置（部署）和取消置备（销毁） AWS 资源的代码，以及用于安装 CDK 并运行 CDK 代码的清单。 AWS Cloud Development Kit (AWS CDK)

以下各节列出了您可以包含在 CodeBuild 配置模板包中的示例文件，该模板包使用置备环境 AWS CDK。

清单

以下清单文件指定了 CodeBuild 配置，并包括安装和使用 AWS CDK、输出文件处理和报告输出所需的命令 AWS Proton。

Example infrastructure/manifest.yaml

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

架构

以下架构文件为环境定义参数。您的 AWS CDK 代码可以在部署期间引用这些参数的值。

Example schema/schema.yaml

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

AWS CDK 文件

以下文件是一个 Node.js CDK 项目示例。

Example infrastructure/package.json

```
{
  "name": "ProtonEnvironment",
  "version": "0.1.0",
  "bin": {
    "ProtonEnvironmente": "bin/ProtonEnvironment.js"
  },
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk"
  },
```

```
  "devDependencies": {
    "@types/jest": "^28.1.7",
    "@types/node": "18.7.6",
    "jest": "^28.1.3",
    "ts-jest": "^28.0.8",
    "aws-cdk": "2.37.1",
    "ts-node": "^10.9.1",
    "typescript": "~4.7.4"
  },
  "dependencies": {
    "aws-cdk-lib": "2.37.1",
    "constructs": "^10.1.77",
    "source-map-support": "^0.5.21"
  }
}
```

Example infrastructure/tsconfig.json

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

```
    ]
  },
  "exclude": [
    "node_modules",
    "cdk.out"
  ]
}
```

Example infrastructure/cdk.json

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

Example 基础设施/bin/. ProtonEnvironment ts

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

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

Example 基础设施/lib/. ProtonEnvironmentStack ts

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

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

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

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

渲染的输入文件

当您使用 CodeBuild基于的配置模板创建环境时，会使用您提供的输入参数值 AWS Proton 呈现一个输入文件。您的代码可以引用这些值。以下文件是一个渲染的输入文件示例。

Example infrastructure/proton-inputs.json

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

Terraform IaC 文件

学习如何使用 Terraform 基础设施即代码 (IaC) 文件。 AWS ProtonTerraform 是一个广泛使用的开源 IaC 引擎，由开发。HashiCorpTerraform 模块是用 HCL 语言开发 HashiCorp的，支持包括亚马逊 Web Services 在内的多个后端基础设施提供商。

AWS Proton 支持 Terraform IaC 的自我管理配置。

有关响应拉取请求并实现基础架构配置的配置存储库的完整示例，请参阅上的 Terraform Acti OpenSource GitHub ons 自动化模板。 AWS Proton GitHub

自托管式预置如何使用 Terraform IaC 模板捆绑包文件：

1. 使用 Terraform 模板包创建环境时，使用控制台或输入参数 AWS Proton 编译`.tf`文件。`spec file`
2. 它发出拉取请求，将编译的 IaC 文件合并到您在 AWS Proton中注册的存储库。
3. 如果请求获得批准，则 AWS Proton 等待您提供的配置状态。
4. 如果拒绝了该请求，则取消创建环境。
5. 如果拉取请求超时，则不会 完成创建环境。

AWS Proton 考虑到 Terraform IaC 的注意事项：

- AWS Proton 无法管理你的 Terraform 配置。
- 您必须向注册配置存储库 AWS Proton。 AWS Proton 对该存储库发出拉取请求。
- 您必须创建 CodeStar 连接才能连接到 AWS Proton 您的配置存储库。

- 要从 AWS Proton 已编译的 IaC 文件中进行配置，您必须响应 AWS Proton 拉取请求。 AWS Proton 在环境和服务创建和更新操作之后发出拉取请求。有关更多信息，请参阅 AWS Proton 环境 和 AWS Proton 服务。
- 要使用 AWS Proton 已编译的 IaC 文件配置管道，必须创建 CI/CD 管道存储库。
- 基于拉取请求的配置自动化必须包括通知 AWS Proton 任何已配置 AWS Proton 资源状态变化的步骤。你可以使用 AWS Proton NotifyResourceDeploymentStatusChange API。
- 您无法将从 CloudFormation IaC 文件创建的服务、管道和组件部署到通过 Terraform iaC 文件创建的环境中。
- 您无法将从 Terraform IaC 文件创建的服务、管道和组件部署到通过 IaC 文件创建的环境中。 CloudFormation

在准备 Terraform IaC 文件时 AWS Proton，需要将命名空间附加到输入变量，如以下示例所示。有关更多信息，请参阅参数。

示例 1： AWS Proton 环境 Terraform iaC 文件

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

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

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

编译的基础设施即代码

创建环境或服务时，使用控制台或`spec file`输入将基础架构 AWS Proton 编译为代码文件。它为 Terraform 可使用的输入创建 `proton.`*`resource-type`*`.variables.tf` 和 `proton.auto.tfvars.json` 文件，如以下示例中所示。这些文件位于指定存储库中与环境或服务实例名称匹配的文件夹中。

该示例显示了如何在变量定义和变量值中 AWS Proton 包含标签，以及如何将这些 AWS Proton 标签传播到已配置的资源。有关更多信息，请参阅the section called “标签传播到预置的资源”。

示例 2：为名为“dev”的环境编译的 IaC 文件。

dev/environment.tf：

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

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

```
  }
}

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

dev/proton.environment.variables.tf :

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

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

dev/proton.auto.tfvars.json :

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

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

存储库路径

AWS Proton 使用来自环境或服务创建操作的控制台或规范输入来找到已编译的 IaC 文件的存储库和路径。输入值将传递到命名空间输入参数。

AWS Proton 支持两种存储库路径布局。在以下示例中，路径是由两个环境中的命名空间资源参数命名的。每个环境具有两个服务的服务实例，其中的一个服务的服务实例具有直接定义的组件。

资源类型	名称参数	=	资源名称
环境	environment.name		"env-prod"
环境	environment.name		"env-staged"
服务	service.name		"service-one"
服务实例	service_instance.name		"instance-one-prod"
服务实例	service_instance.name	=	"instance-one-staged"
服务	service.name		"service-two"
服务实例	service_instance.name		"instance-two-prod"
组件	service_instance.components.default.name		"component-prod"

资源类型	名称参数	=	资源名称
服务实例	service_instance.name		"instance-two-staged"
组件	service_instance.components.default.name		"component-staged"

Layout 1

如果 AWS Proton 找到带有environments文件夹的指定存储库，它将创建一个包含已编译的 IaC 文件并以命名的文件夹。environment.name

如果在指定的存储库中 AWS Proton 找到的environments文件夹包含与服务实例兼容环境名称相匹配的文件夹名称，则它会创建一个包含已编译的实例 IaC 文件并以命名的文件夹。service_instance.name

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

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

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

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

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

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

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

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

Layout 2

如果 AWS Proton 找到没有`environments`文件夹的指定存储库，它将创建一个文件`environment.name`夹，用于查找已编译的环境 IaC 文件。

如果 AWS Proton 找到的指定存储库的文件夹名称与服务实例兼容的环境名称相匹配，它将创建一个`service_instance.name`文件夹，用于查找已编译的实例 IaC 文件。

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

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

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

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

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

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

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

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

架构文件

作为管理员，当您使用 Open API 数据模型（架构）部分为模板包定义参数架构 YAML 文件时， AWS Proton 可以根据您在架构中定义的要求验证参数值输入。

有关格式和可用关键字的更多信息，请参阅 OpenAPI 的 Schema object 部分。

环境模板捆绑包的架构要求

您的架构必须遵循 OpenAPI 的 Data Models (schemas) 部分并采用 YAML 格式。它还必须是您的环境模板捆绑包的一部分。

对于您的环境架构，您必须包含设置了格式的标头，以确认您使用了 Open API 的 Data Models (schemas) 部分。在以下环境架构示例中，这些标头出现在前三行中。

必须包含 environment_input_type 并使用您提供的名称进行定义。在以下示例中，这是在第 5 行中定义的。通过定义此参数，可以将其与 AWS Proton 环境资源相关联。

要遵循 Open API 架构模型，您必须包含 types。在以下示例中，它位于第 6 行。

在 types 后面，您必须定义一个 environment_input_type 类型。您将环境的输入参数定义为 environment_input_type 的属性。您必须包含至少一个属性，其名称至少与架构的关联环境基础设施即代码 (IaC) 文件中列出的一个参数匹配。

创建环境并提供自定义参数值时，会 AWS Proton 使用架构文件进行匹配、验证并将其注入关联的 CloudFormation IaC 文件中的花括号参数中。对于每个属性（参数），提供 name 和 type。（可选）还提供 description、default 和 pattern。

为以下示例标准 环境模板架构定义的参数包括 vpc_cidr、subnet_one_cidr 和 subnet_two_cidr，它们使用 default 关键字和默认值。在您使用该环境模板捆绑包架构创建环境时，您可以接受默认值或提供您自己的值。如果参数没有 默认值并且作为 required 属性（参数）列出，您在创建环境时必须为其提供值。

第二个示例标准 环境模板架构列出了 required 参数 my_other_sample_input。

您可以为两种类型的环境模板创建一个架构。有关更多信息，请参阅注册并发布模板。

- 标准 环境模板

  在以下示例中，环境输入类型是使用描述和输入属性定义的。此架构示例可以与示例 3 中所示的 AWS Proton CloudFormation IaC 文件一起使用。

  标准 环境模板的示例架构：

```
schema:                             # required
  format:                           # required
    openapi: "3.0.0"                # required
  # required              defined by administrator
  environment_input_type: "PublicEnvironmentInput"
  types:                            # required
    # defined by administrator
    PublicEnvironmentInput:
      type: object
      description: "Input properties for my environment"
      properties:
        vpc_cidr:                    # parameter
          type: string
```

```
      description: "This CIDR range for your VPC"
      default: 10.0.0.0/16
      pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
    subnet_one_cidr:            # parameter
      type: string
      description: "The CIDR range for subnet one"
      default: 10.0.0.0/24
      pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
    subnet_two_cidr:            # parameter
      type: string
      description: "The CIDR range for subnet one"
      default: 10.0.1.0/24
      pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
```

包含 `required` 参数的标准 环境模板的示例架构：

```
schema:                           # required
  format:                         # required
    openapi: "3.0.0"              # required
  # required              defined by administrator
  environment_input_type: "MyEnvironmentInputType"
  types:                          # required
    # defined by administrator
    MyEnvironmentInputType:
      type: object
      description: "Input properties for my environment"
      properties:
        my_sample_input:          # parameter
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_other_sample_input:    # parameter
          type: string
          description: "Another sample input"
        another_optional_input:   # parameter
          type: string
          description: "Another optional input"
          default: "!"
      required:
        - my_other_sample_input
```

- 客户托管 环境模板

在以下示例中，架构仅包含输出列表，这些输出复制您用于预置客户托管 基础设施的 IaC 的输出。您需要将输出值类型仅定义为字符串（而不能 定义为列表、数组或其他类型）。例如，下一个代码片段显示了外部 AWS CloudFormation 模板的输出部分。它来自于示例 1 中所示的模板。它可用于为根据示例 4 创建的 AWS Proton Fargate 服务创建外部客户托管基础架构。

> **Important**
>
> 作为管理员，您必须确保您的预配置和托管基础设施以及所有输出参数都与相关的客户托管环境模板兼容。 AWS Proton 无法代表你解释更改，因为这些更改对他们不可见 AWS Proton。不一致会导致失败。

客户托管环境模板的 CloudFormation IaC 文件输出示例：

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

以下示例显示了相应的 AWS Proton 客户托管环境模板包的架构。每个输出值定义为一个字符串。

客户托管 环境模板的示例架构：

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

```
      type: object
      description: "Outputs of the environment"
      properties:
        ClusterName:               # parameter
          type: string
          description: "The name of the ECS cluster"
        ECSTaskExecutionRole:      # parameter
          type: string
          description: "The ARN of the ECS role"
        VpcId:                     # parameter
          type: string
          description: "The ID of the VPC that this stack is deployed in"
[...]
```

服务模板捆绑包的架构要求

您的架构必须遵循 OpenAPI 的 Data Models (schemas) 部分并采用 YAML 格式，如以下示例中所示。您必须在服务模板捆绑包中提供架构文件。

在以下服务架构示例中，您必须包含设置了格式的标头。在以下示例中，它位于前三行。这是为了确认您使用了 Open API 的 Data Models (schemas) 部分。

必须包含 `service_input_type` 并使用您提供的名称进行定义。在以下示例中，它位于第 5 行。这会将参数与 AWS Proton 服务资源相关联。

当您使用控制台或 CLI 创建 AWS Proton 服务时，默认情况下会包含服务管道。在您为服务包含服务管道时，您必须包含 `pipeline_input_type` 并具有您提供的名称。在以下示例中，它位于第 7 行。如果您@@ 不包括 AWS Proton 服务管道，请不要包含此参数。有关更多信息，请参阅注册并发布模板。

要遵循 Open API 架构模型，您必须在以下示例中包含 `types`，它位于第 9 行。

在 `types` 后面，您必须定义一个 `service_input_type` 类型。您将服务的输入参数定义为 `service_input_type` 的属性。您必须包含至少一个属性，其名称至少与架构的关联服务基础设施即代码 (IaC) 文件中列出的一个参数匹配。

要定义服务管道，您必须在 `service_input_type` 定义下面定义一个 `pipeline_input_type`。如上所述，您必须包含至少一个属性，其名称至少与架构的关联管道 IaC 文件中列出的一个参数匹配。如果您@@ 不包括 AWS Proton 服务管道，请不要包含此定义。

作为管理员或开发人员，当您创建服务并提供自定义参数值时，会 AWS Proton 使用架构文件来匹配、验证这些值，并将其注入关联的 CloudFormation IaC 文件的花括号参数中。对于每个属性（参数），提供 name 和 type。（可选）还提供 description、default 和 pattern。

为示例模式定义的参数包括 port、desired_count、task_size 和 image，它们使用 default 关键字和默认值。在您使用该服务模板捆绑包架构创建服务时，您可以接受默认值或提供您自己的值。unique_name 参数也包含在该示例中，并且没有 默认值。它作为 required 属性（参数）列出。作为管理员或开发人员，您在创建服务时必须提供 required 参数的值。

如果要创建具有服务管道的服务模板，请将 pipeline_input_type 包含在架构中。

包含服务管道的服务的示例 AWS Proton 服务架构文件。

此架构示例可以与示例 4 和示例 5 中所示的 AWS Proton IaC 文件一起使用。包含一个服务管道。

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

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

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

如果要创建没有服务管道的服务模板，请不要 将 `pipeline_input_type` 包含在架构中，如以下示例中所示。

不包含服务管道的服务的示例 AWS Proton 服务架构文件

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

```
  service_input_type: "MyServiceInstanceInputType"

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

总结模板文件 AWS Proton

在准备环境和服务基础设施即代码 (IaC) 文件及其相应的架构文件后，您必须将它们放置到不同的目录中。您还必须创建一个清单 YAML 文件。清单文件列出了目录中的 IaC 文件、渲染引擎以及用于开发该模板中的 IaC 的模板语言。

> **Note**
>
> 也可以独立于模板捆绑包使用清单文件，以作为直接定义的组件 的直接输入。在这种情况下，它总是为两者 CloudFormation 和 Terraform 指定一个 IaC 模板文件。有关组件的更多信息，请参阅组件。

清单文件需要符合以下示例中所示的格式和内容。

CloudFormation 清单文件格式：

使用 CloudFormation，您可以列出单个文件。

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

```
      template_language: cloudformation
```

Terraform 清单文件格式：

通过使用 Terraform，您可以明确列出单个文件，或使用 * 通配符列出目录中的每个文件。

> **Note**
>
> 通配符仅包含名称以 .tf 结尾的文件。将忽略其他文件。

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

CodeBuild基于配置的清单文件格式：

使用 CodeBuild基于基础的配置，您可以指定配置和取消置备 shell 命令。

> **Note**
>
> 除了清单以外，您的捆绑包还应包含您的命令依赖的任何文件。

以下示例清单使用 CodeBuild基于配置的资源调配，使用 () 来配置（部署）和取消配置 AWS Cloud Development Kit (AWS CDK)（销毁AWS CDK）资源。模板捆绑包还应包含 CDK 代码。

在预置期间，AWS Proton 创建一个输入文件，其中包含您在名为 proton-input.json 的模板架构中定义的输入参数的值。

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
        provision:
```

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

为环境或服务模板包设置目录和清单文件后，您可以将这些目录压缩成焦油球，然后将其上传到可以检索它们的亚马逊简单存储服务 (Amazon S3) 存储桶 AWS Proton ，或者上传到模板同步 Git 存储库。

当您创建注册的环境或服务模板的次要版本时 AWS Proton，您需要提供指向位于 S3 存储桶中的环境或服务模板包 tar ball 的路径。 AWS Proton 将其与新的模板次要版本一起保存。您可以选择新的模板次要版本来创建或更新环境或服务 AWS Proton。

打包环境模板捆绑包

您创建的环境模板包有两种类型。 AWS Proton

- 要为标准 环境模板创建环境模板捆绑包，请在目录中放置架构、基础设施即代码 (IaC) 文件和清单文件，如以下环境模板捆绑包目录结构中所示。
- 要为客户托管 环境模板创建环境模板捆绑包，请仅提供架构文件和目录。不要包含基础架构目录和文件。 AWS Proton 如果包含基础架构目录和文件，则会引发错误。

有关更多信息，请参阅注册并发布模板。

CloudFormation 环境模板包目录结构：

```
/schema
  schema.yaml
```

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

Terraform 环境模板捆绑包目录结构：

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

打包服务模板捆绑包

要创建服务模板捆绑包，您必须将架构、基础设施即代码 (IaC) 文件和清单文件放置在目录中，如服务模板捆绑包目录结构示例中所示。

如果在您的模板捆绑包中不 包含服务管道，则不要 包含管道目录和文件，并在创建与该模板捆绑包关联的服务模板时设置 `"pipelineProvisioning": "CUSTOMER_MANAGED"`。

> **Note**
>
> 在创建服务模板后，您无法修改 `pipelineProvisioning`。

有关更多信息，请参阅注册并发布模板。

CloudFormation 服务模板包目录结构：

```
/schema
  schema.yaml
/instance_infrastructure
  manifest.yaml
  cloudformation.yaml
/pipeline_infrastructure
  manifest.yaml
  cloudformation.yaml
```

Terraform 服务模板捆绑包目录结构：

```
/schema
```

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

模板捆绑包注意事项

- 基础设施即代码 (IaC) 文件

  AWS Proton 审核模板以确定文件格式是否正确。但是，AWS Proton 不检查模板开发、依赖关系和逻辑错误。例如，假设您在服务或环境模板中指定在 AWS CloudFormation IaC 文件中创建 Amazon S3 存储桶。服务是根据这些模板创建的。现在，假设您在某个时候希望删除该服务。如果指定的 S3 存储桶不为空且 CloudFormation IaC 文件未将其标记为`Retain`在中`DeletionPolicy`，AWS Proton 则服务删除操作将失败。
- 捆绑包文件大小限制和格式
  - 可以在AWS Proton 配额中找到捆绑包文件大小、计数和名称大小限制。
  - 文件的模板捆绑包目录使用 gzip 压缩为 tar 包，并位于 Amazon Simple Storage Service (Amazon S3) 存储桶中。
  - 捆绑包中的每个文件必须是有效的 YAML 格式文件。
- S3 存储桶模板捆绑包加密

  如果您想在 S3 存储桶中对模板包中的敏感数据进行静态加密，请使用 SSE-S3 或 SSE-KMS 密钥 AWS Proton 允许检索这些数据。

AWS Proton 模板

要将模板捆绑包添加到 AWS Proton 模板库中，请创建一个模板次要版本并在 AWS Proton 中注册该版本。在创建模板时，请提供模板捆绑包的 Amazon S3 存储桶名称和路径。在发布模板后，平台团队成员和开发人员可以选择这些模板。在选择模板后，AWS Proton 使用该模板创建和预置基础设施和应用程序。

作为管理员，您可以使用 AWS Proton 创建并注册一个环境模板。然后，可以使用该环境模板部署多个环境。例如，可以使用该模板部署“dev”、“staging”和“prod”环境。“dev”环境可能包括具有私有子网的 VPC 以及所有资源的限制性访问策略。可以将环境输出作为服务的输入。

您可以创建并注册环境模板以创建两种不同类型的环境。您和开发人员可以使用 AWS Proton 将服务部署到这两种类型的环境中。

- 注册并发布一个标准 环境模板，AWS Proton 使用该模板创建标准 环境以预置和管理环境基础设施。
- 注册并发布客户托管 环境模板，AWS Proton 使用该模板创建客户托管环境以连接到现有的预置基础设施。AWS Proton 不会 管理您的现有预置基础设施。

您可以使用 AWS Proton 创建并注册服务模板，以将服务部署到环境中。必须先创建一个 AWS Proton 环境，然后才能将服务部署到该环境中。

以下列表介绍了如何使用 AWS Proton 创建和管理模板。

- （可选）准备一个 IAM 角色以控制开发人员对 AWS Proton API 调用和 AWS Proton IAM 服务角色的访问。有关更多信息，请参阅the section called “IAM 角色”。
- 编写一个模板捆绑包。有关更多信息，请参阅模板捆绑包。
- 在编写和压缩模板捆绑包并将其保存到 Amazon S3 存储桶后，使用 AWS Proton 创建并注册一个模板。您可以在控制台中或使用 AWS CLI 执行该操作。
- 在 AWS Proton 中注册模板后，测试并使用该模板创建和管理 AWS Proton 预置的资源。
- 在模板的整个生命周期中创建和管理模板的主要版本和次要版本。

您可以手动或使用模板同步配置管理模板版本：

- 使用 AWS Proton 控制台和 AWS CLI 创建新的次要或主要版本。

- 创建模板同步配置，让 AWS Proton 在检测到您定义的存储库中的模板捆绑包发生变化时自动创建新的次要或主要版本。

有关其他信息，请参阅 The AWS Proton Service API Reference。

主题

版本控制的模板

作为管理员或平台团队成员，您定义、创建和管理版本控制的模板库以用于预置基础设施资源。共有两种类型的模板版本 - 次要版本和主要版本。

- 次要版本 - 具有向后兼容架构的模板更改。这些更改不要求开发人员在更新为新模板版本时提供新信息。

  在您尝试进行次要版本更改时，AWS Proton 尽力确定新版本的架构是否与以前的模板次要版本向后兼容。如果新架构不向后兼容，AWS Proton 将无法注册新的次要版本。

  > **Note**
  >
  > 兼容性是仅根据架构确定的。AWS Proton 不检查模板捆绑包基础设施即代码 (IaC) 文件是否与以前的次要版本向后兼容。例如，AWS Proton 不检查新的 IaC 文件是否会导致在以前的模板次要版本预置的基础设施上运行的应用程序发生重大更改。

- 主要版本 - 可能不向后兼容的模板更改。这些更改通常要求开发人员提供新的输入，并且通常涉及模板架构更改。

  有时，您可能会根据团队的运营模式选择将向后兼容的更改指定为主要版本。

AWS Proton 确定模板版本请求是针对次要版本还是主要版本的方式取决于跟踪模板更改的方式：

- 在您明确请求创建新的模板版本时，您可以指定主要版本号以请求主要版本，或者不指定主要版本号以请求次要版本。
- 在您使用模板同步（因此，您不会发出明确的模板版本请求）时，AWS Proton 尝试为现有 YAML 文件中发生的模板更改创建新的次要版本。在您为新的模板更改创建新目录（例如，从 v1 移动到 v2）时，AWS Proton 创建一个主要版本。

> **Note**
>
> 如果 AWS Proton 确定更改不向后兼容，基于模板同步的新次要版本注册仍然会失败。

在您发布新的模板版本时，如果它是最高的主要版本和次要版本，它将成为推荐版本。新的 AWS Proton 资源是使用新的推荐版本创建的，并且 AWS Proton 提示管理员使用新版本并更新使用过时版本的现有 AWS Proton 资源。

注册并发布模板

您可以使用 AWS Proton 注册并发布环境模板和服务模板，如以下几节中所述。

您可以使用控制台或 AWS CLI 创建新的模板版本。

或者，您可以使用控制台或 AWS CLI 创建模板，并为其配置模板同步。该配置让 AWS Proton 从位于您定义的注册 Git 存储库中的模板捆绑包同步。每次将提交推送到更改某个模板捆绑包的存储库时，都会创建新的模板次要或主要版本（如果该版本尚不存在）。要了解模板同步配置先决条件和要求的更多信息，请参阅模板同步配置。

注册和发布环境模板

您可以注册并发布以下类型的环境模板。

- 注册并发布一个标准 环境模板，AWS Proton 使用该模板部署和管理环境基础设施。
- 注册并发布一个客户托管 环境模板，AWS Proton 使用该模板连接到您管理的现有预置基础设施。AWS Proton 不会 管理您的现有预置基础设施。

> **Important**
>
> 作为管理员，确保您的预置和托管基础设施以及所有输出参数与关联的客户托管 环境模板兼容。AWS Proton 无法代表您处理更改，因为这些更改对 AWS Proton 不可见。不一致会导致失败。

您可以使用控制台或 AWS CLI 注册并发布环境模板。

AWS Management Console

使用控制台注册并发布新的环境模板。

1. 在 AWS Proton 控制台中，选择环境模板。
2. 选择创建环境模板。
3. 在创建环境模板页面上的模板选项部分中，选择两个可用的模板选项之一。
   - 创建用于预置新环境的模板。
   - 创建模板以使用您管理的预置基础设施。
4. 如果您选择创建用于预置新环境的模板，则在模板捆绑包源部分中选择三个可用的模板捆绑包源选项之一。要了解模板同步要求和先决条件的更多信息，请参阅模板同步配置。
   - 使用我们的示例模板捆绑包。
   - 使用自己的模板捆绑包。
   - 从 Git 同步模板。
5. 提供模板捆绑包的路径。
   a. 如果您选择了使用我们的示例模板捆绑包：

      在示例模板捆绑包部分中，选择一个示例模板捆绑包。
   b. 如果您选择了从 Git 同步模板，在源代码部分中：
      i. 为您的模板同步配置选择存储库。
      ii. 输入要从中同步的存储库分支的名称。
      iii. （可选）输入目录名称以限制模板捆绑包搜索。
   c. 否则，在 S3 捆绑包位置部分中，提供您的模板捆绑包的路径。
6. 在模板详细信息部分中。

a. 输入模板名称。

b. （可选）输入模板显示名称。

c. （可选）输入环境模板的模板描述。

7. （可选）选中加密设置部分中的自定义加密设置 (高级) 复选框以提供您自己的加密密钥。
8. （可选）在标签部分中，选择添加新标签，并输入键和值以创建一个客户托管标签。
9. 选择创建环境模板。

   您现在位于一个新页面中，其中显示新环境模板的状态和详细信息。这些详细信息包括 AWS 托管标签和客户托管标签列表。在您创建 AWS Proton 资源时，AWS Proton 自动为您生成 AWS 托管标签。有关更多信息，请参阅AWS Proton 资源和标记。
10. 新环境模板的初始状态为草稿。您和具有 `proton:CreateEnvironment` 权限的其他人可以查看和访问该模板。执行下一步，以使该模板可供其他人使用。
11. 在模板版本部分中，选择刚创建的模板次要版本 (1.0) 左侧的单选按钮。或者，您可以在信息提醒中选择发布并跳过下一步。
12. 在模板版本部分中，选择发布。
13. 模板状态变为已发布。由于它是最新的模板版本，因此，它是推荐版本。
14. 在导航窗格中，选择环境模板以查看环境模板和详细信息列表。

使用控制台注册新的环境模板主要版本和次要版本。

有关更多信息，请参阅版本控制的模板。

1. 在 AWS Proton 控制台中，选择环境模板。
2. 在环境模板列表中，选择要创建主要或次要版本的环境模板的名称。
3. 在环境模板详细信息视图中，在模板版本部分中选择创建新版本。
4. 在创建新环境模板版本页面上的模板捆绑包源部分中，选择两个可用的模板捆绑包源选项之一。

   - 使用我们的示例模板捆绑包。
   - 使用自己的模板捆绑包。
5. 提供选定模板捆绑包的路径。

   - 如果您选择了使用我们的示例模板捆绑包，请在示例模板捆绑包部分中选择一个示例模板捆绑包。

- 如果您选择了使用自己的模板捆绑包包，请在 S3 捆绑包位置部分中选择您的模板捆绑包的路径。

6. 在模板详细信息部分中。

   a. （可选）输入模板显示名称。

   b. （可选）输入服务模板的模板描述。

7. 在模板详细信息部分中，选择以下选项之一。

   - 要创建次要版本，请将选中可创建新的主要版本复选框保留为空。
   - 要创建主要版本，请选中选中可创建新的主要版本复选框。

8. 继续执行控制台步骤以创建新的次要版本或主要版本，然后选择创建新版本。

AWS CLI

使用 CLI 注册并发布新的环境模板，如以下步骤中所示。

1. 指定区域、名称、显示名称（可选）和描述（可选）以创建一个标准 或客户托管 环境模板。

   a. 创建一个标准 环境模板。

      运行以下命令：

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

      响应：

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

```
}
```

b. 添加值为 CUSTOMER_MANAGED 的 provisioning 参数以创建一个客户托管 环境模板。

运行以下命令：

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

响应：

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

2. 创建环境模板主要版本 1 的次要版本 0

对于标准 和客户托管 环境模板，该步骤和其余步骤是相同的。

包括模板名称、主要版本以及包含环境模板捆绑包的 S3 存储桶的存储桶名称和密钥。

运行以下命令：

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

响应：

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

3. 使用 get 命令检查注册状态。

   运行以下命令：

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

   响应：

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

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

4. 提供模板名称以及主要版本和次要版本，以发布环境模板主要版本 1 的次要版本 0。该版本是 Recommended 版本。

   运行以下命令：

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

   响应：

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

```
}
```

在使用 AWS CLI 创建新模板后，您可以查看 AWS 和客户托管标签列表。AWS Proton 自动为您生成 AWS 托管标签。您也可以使用 AWS CLI 修改和创建客户托管标签。有关更多信息，请参阅AWS Proton 资源和标记。

运行以下命令：

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

注册和发布服务模板

在创建服务模板版本时，您可以指定兼容的环境模板列表。这样，在开发人员选择服务模板时，他们可以选择将其服务部署到哪个环境中。

在通过服务模板创建服务或发布服务模板之前，请确认环境是通过列出的兼容环境模板部署的。

如果将服务部署到通过删除的兼容环境模板构建的环境中，则无法 将该服务更新为新的主要版本。

要添加或删除服务模板版本的兼容环境模板，您可以为其创建新的主要版本。

您可以使用控制台或 AWS CLI 注册并发布服务模板。

AWS Management Console

使用控制台注册并发布新的服务模板。

1. 在 AWS Proton 控制台中，选择服务模板。
2. 选择创建服务模板。
3. 在创建服务模板页面上的模板捆绑包源部分中，选择可用的模板选项之一。
   - 使用自己的模板捆绑包。
   - 从 Git 同步模板。
4. 提供模板捆绑包的路径。
   a. 如果您选择了从 Git 同步模板，在源代码存储库部分中：

i. 为您的模板同步配置选择存储库。

ii. 输入要从中同步的存储库分支的名称。

iii. （可选）输入目录名称以限制模板捆绑包搜索。

b. 否则，在 S3 捆绑包位置部分中，提供您的模板捆绑包的路径。

5. 在模板详细信息部分中。

a. 输入模板名称。

b. （可选）输入模板显示名称。

c. （可选）输入服务模板的模板描述。

6. 在兼容的环境模板部分中，从兼容的环境模板列表中进行选择。

7. （可选）在加密设置部分中，选择自定义加密设置 (高级) 以提供您自己的加密密钥。

8. （可选）在管道部分中：

如果在您的服务模板中不包含服务管道定义，请取消选中页面底部的管道 - 可选复选框。在创建服务模板后，您无法 更改该设置。有关更多信息，请参阅模板捆绑包。

9. （可选）在支持的组件源部分中，为组件源选择直接定义以允许将直接定义的组件附加到服务实例。

10. （可选）在标签部分中，选择添加新标签，并输入键和值以创建一个客户托管标签。

11. 选择创建服务模板。

您现在位于一个新页面中，其中显示新服务模板的状态和详细信息。这些详细信息包括 AWS 托管标签和客户托管标签列表。在您创建 AWS Proton 资源时，AWS Proton 自动为您生成 AWS 托管标签。有关更多信息，请参阅AWS Proton 资源和标记。

12. 新服务模板的初始状态为草稿。您和具有 `proton:CreateService` 权限的其他人可以查看和访问该模板。执行下一步，以使该模板可供其他人使用。

13. 在模板版本部分中，选择刚创建的模板次要版本 (1.0) 左侧的单选按钮。或者，您可以在信息提醒中选择发布并跳过下一步。

14. 在模板版本部分中，选择发布。

15. 模板状态变为已发布。由于它是最新的模板版本，因此，它是推荐版本。

16. 在导航窗格中，选择服务模板以查看服务模板和详细信息列表。

使用控制台注册新的服务模板主要版本和次要版本。

有关更多信息，请参阅版本控制的模板。

1. 在 AWS Proton 控制台中，选择服务模板。
2. 在服务模板列表中，选择要创建主要或次要版本的服务模板的名称。
3. 在服务模板详细信息视图中，在模板版本部分中选择创建新版本。
4. 在创建新服务模板版本页面上的捆绑包源部分中，选择使用自己的模板捆绑包。
5. 在 S3 捆绑包位置部分中，选择您的模板捆绑包的路径。
6. 在模板详细信息部分中。
   a. （可选）输入模板显示名称。
   b. （可选）输入服务模板的模板描述。
7. 在模板详细信息部分中，选择以下选项之一。
   - 要创建次要版本，请将选中可创建新的主要版本复选框保留为空。
   - 要创建主要版本，请选中选中可创建新的主要版本复选框。
8. 继续执行控制台步骤以创建新的次要版本或主要版本，然后选择创建新版本。

AWS CLI

要创建用于部署服务并且没有服务管道的服务模板，请将 `--pipeline-provisioning "CUSTOMER_MANAGED"` 参数和值添加到 `create-service-template` 命令中。按照模板捆绑包创建和服务模板捆绑包的架构要求中所述，配置您的模板捆绑包。

> **Note**
> 在创建服务模板后，您无法修改 `pipelineProvisioning`。

1. 使用 CLI 注册并发布一个新的服务模板（具有或没有服务管道），如以下步骤中所示。
   a. 使用 CLI 创建一个具有服务管道的服务模板。

      指定名称、显示名称（可选）和描述（可选）。

      运行以下命令：

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

响应：

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

b. 创建一个没有服务管道的服务模板。

添加 `--pipeline-provisioning`。

运行以下命令：

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

响应：

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

```
        "name": "fargate-service",
        "pipelineProvisioning": "CUSTOMER_MANAGED"
    }
}
```

2. 创建服务模板主要版本 1 的次要版本 0。

   包括模板名称、兼容的环境模板、主要版本以及包含服务模板捆绑包的 S3 存储桶的存储桶名称和密钥。

   运行以下命令：

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

   响应：

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

3. 使用 get 命令检查注册状态。

运行以下命令：

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

响应：


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

```
    }
}
```

4. 使用 update 命令将状态更改为 "PUBLISHED" 以发布服务模板。

   运行以下命令：

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

   响应：

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

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

5. 使用 get 命令检索服务模板详细数据，以检查 AWS Proton 是否发布了版本 1.0。

   运行以下命令：

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

   响应：

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

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

查看模板数据

您可以使用 AWS Proton 控制台和 AWS CLI 查看模板和详细信息列表，以及查看各个模板和详细数据。

客户托管 环境模板数据包括值为 CUSTOMER_MANAGED 的 `provisioned` 参数。

如果服务模板不 包含服务管道，则服务模板数据包括值为 CUSTOMER_MANAGED 的 `pipelineProvisioning` 参数。

有关更多信息，请参阅注册并发布模板。

您可以使用控制台或 AWS CLI 列出和查看模板数据。

AWS Management Console

使用控制台列出和查看模板。

1. 要查看模板列表，请选择（环境或服务）模板。
2. 要查看详细数据，请选择一个模板的名称。

   查看模板的详细数据、模板的主要版本和次要版本列表、使用模板版本和模板标签部署的 AWS Proton 资源列表。

   推荐的主要版本和次要版本标记为推荐。

AWS CLI

使用 AWS CLI 列出和查看模板。

运行以下命令：

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

响应：

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

运行以下命令：

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

响应：

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

查看服务模板的次要版本。

运行以下命令：

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

响应：

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

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

查看没有服务管道的服务模板，如下一个示例命令和响应中所示。

运行以下命令：

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

响应：

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

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

更新模板

您可以按照以下列表中所述更新模板。

- 在您使用控制台或 AWS CLI 时，编辑模板的 `description` 或 `display name`。您无法 编辑模板的 `name`。
- 在您使用控制台或 AWS CLI 时，更新模板次要版本的状态。您只能将状态从 `DRAFT` 更改为 `PUBLISHED`。
- 在使用 AWS CLI 时，编辑模板次要或主要版本的显示名称和描述。

AWS Management Console

使用控制台编辑模板描述和显示名称，如以下步骤中所述。

在模板列表中。

1. 在 AWS Proton 控制台中，选择（环境或服务）模板。
2. 在模板列表中，选择要更新描述或显示名称的模板左侧的单选按钮。
3. 选择操作，然后选择编辑。
4. 在编辑（环境或服务）模板页面上的模板详细信息部分中，在表单中输入您的编辑内容，然后选择保存更改。

使用控制台发布模板以更改模板次要版本状态，如下所述。您只能将状态从 `DRAFT` 更改为 `PUBLISHED`。

在（环境或服务）模板详细信息页面中。

1. 在 AWS Proton 控制台中，选择（环境或服务）模板。

2. 在模板列表中，选择您希望将次要版本状态从草稿更新为已发布的模板的名称。
3. 在（环境或服务）模板详细信息页面上的模板版本部分中，选择要发布的次要版本左侧的单选按钮。
4. 在模板版本部分中选择发布。状态将从草稿变为已发布。

AWS CLI

以下示例命令和响应说明了如何编辑环境模板描述。

运行以下命令。

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

响应：

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

您也可以使用 AWS CLI 更新服务模板。有关更新服务模板次要版本状态的示例，请参阅注册和发布服务模板中的步骤 5。

删除模板

可以使用控制台和 AWS CLI 删除模板。

如果没有部署到某个环境模板次要版本的环境，您可以删除该版本。

如果没有部署到某个服务模板次要版本的服务实例或管道，您可以删除该版本。可以将您的管道部署到与您的服务实例不同的模板版本中。例如，如果您的服务实例从 1.0 更新为版本 1.1，并且您的管道仍部署到版本 1.0 中，则无法删除服务模板 1.0。

AWS Management Console

您可以使用控制台删除整个模板或模板的各个次要版本和主要版本。

使用控制台删除模板，如下所示。

> **Note**
>
> 在使用控制台删除模板时：
>
> - 在删除整个模板时，您还会删除该模板的主要版本和次要版本。

在（环境或服务）模板列表中。

1. 在 AWS Proton 控制台中，选择（环境或服务）模板。
2. 在模板列表中，选择要删除的模板左侧的单选按钮。

   只有在没有将任何 AWS Proton 资源部署到模板的版本时，您才能删除整个模板。
3. 选择操作，然后选择删除以删除整个模板。
4. 一个模态框提示您确认删除操作。
5. 按照说明进行操作并选择是，删除。

在（环境或服务）模板详细信息页面中。

1. 在 AWS Proton 控制台中，选择（环境或服务）模板。

2. 在模板列表中，选择要完全删除的模板名称或要删除各个主要或次要版本的模板名称。
3. 删除整个模板。

   只有在没有将任何 AWS Proton 资源部署到模板的版本时，您才能删除整个模板。

   a. 选择页面右上角的删除。
   b. 一个模态框提示您确认删除操作。
   c. 按照说明进行操作并选择是，删除。
4. 删除模板的主要或次要版本。

   只有在没有将任何 AWS Proton 资源部署到某个模板次要版本时，您才能删除该版本。

   a. 在模板版本部分中，选择要删除的版本左侧的单选按钮。
   b. 在模板版本部分中选择删除。
   c. 一个模态框提示您确认删除操作。
   d. 按照说明进行操作并选择是，删除。

AWS CLI

AWS CLI 模板删除操作不 包括删除模板的其他版本。在使用 AWS CLI 删除模板时，应满足以下条件。

- 如果模板不存在次要或主要版本，则删除整个模板。
- 在您删除剩下的最后一个次要版本时，将删除主要版本。
- 如果没有将任何 AWS Proton 资源部署到某个模板次要版本中，则删除该版本。
- 如果模板不存在其他次要版本，并且没有将任何 AWS Proton 资源部署到推荐的模板次要版本中，则删除该版本。

以下示例命令和响应说明了如何使用 AWS CLI 删除模板。

运行以下命令：

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

响应：

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:54.610000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

运行以下命令：

```
$ aws proton delete-environment-template \
    --name "simple-env"
```

响应：

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
        "description": "VPC with Public Access",
        "displayName": "VPC",
        "lastModifiedAt": "2020-11-12T00:23:22.339000+00:00",
        "name": "simple-env",
        "recommendedVersion": "1.0"
    }
}
```

运行以下命令：

```
$ aws proton delete-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
```

```
    --minor-version "0"
```

响应：

```
{
    "serviceTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [{"majorVersion": "1", "templateName":
 "simple-env"}],
        "createdAt": "2020-11-28T22:07:05.798000+00:00",
        "lastModifiedAt": "2020-11-28T22:19:05.368000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

模板同步配置

了解如何配置模板，以让 AWS Proton 从位于您定义的注册 Git 存储库中的模板捆绑包同步。在将提交推送到您的存储库时，AWS Proton 检查您的存储库模板捆绑包的更改。如果它检测到模板捆绑包更改，则会创建其模板的新次要或主要版本（如果该版本尚不存在）。AWS Proton 目前支持 GitHub、GitHub Enterprise 和 BitBucket。

将提交推送到同步的模板捆绑包

在您将提交推送到由您的模板之一跟踪的分支时，AWS Proton 克隆您的存储库，并确定需要同步哪些模板。它扫描目录中的文件以查找符合 `{template-name}/{major-version}/` 约定的目录。

在 AWS Proton 确定哪些模板和主要版本与您的存储库和分支关联后，它开始尝试并行同步所有这些模板。

在每次同步到特定模板期间，AWS Proton 先检查自上次成功同步以来模板目录内容是否发生变化。如果内容没有发生变化，AWS Proton 将跳过注册重复的捆绑包。这可确保在模板捆绑包内容发生变化时创建新的模板次要版本。如果模板捆绑包内容发生变化，则在 AWS Proton 中注册该捆绑包。

在注册模板捆绑包后，AWS Proton 监控注册状态，直到注册完成。

特定模板次要版本和主要版本在一个给定时间只能发生一次同步。在进行同步时可能推送的任何提交都会进行批量处理。批量提交将在上一个同步尝试完成后进行同步。

同步服务模板

AWS Proton 可以从 Git 存储库中同步环境模板和服务模板。要同步您的服务模板，您可以将一个名为 `.template-registration.yaml` 的附加文件添加到模板捆绑包的每个主要版本目录中。该文件包含 AWS Proton 在提交后创建服务模板版本时所需的额外详细信息：兼容的环境和支持的组件源。

该文件的完整路径是 *`service-template-name`*`/`*`major-version`*`/.template-registration.yaml`。有关更多信息，请参阅the section called “同步服务模板”。

模板同步配置注意事项

查看以下使用模板同步配置的注意事项。

- 存储库不能超过 250 MB。
- 要配置模板同步，请先将存储库链接到 AWS Proton。有关更多信息，请参阅the section called “创建存储库链接”。
- 通过同步的模板创建新的模板版本时，该版本处于 DRAFT 状态。
- 如果满足以下条件之一，则会创建新的模板次要版本：
  - 模板捆绑包内容与上次同步的模板次要版本内容不同。
  - 已删除上次同步的模板次要版本。
- 无法暂停同步。
- 新的次要版本或主要版本都是自动同步的。
- 无法通过模板同步配置创建新的顶级模板。
- 您无法使用模板同步配置从多个存储库同步到一个模板。
- 您无法使用标签替代分支。
- 在创建服务模板时，您可以指定兼容的环境模板。
- 您可以创建一个环境模板，并将其作为服务模板的兼容环境添加到同一提交中。
- 到单个模板主要版本的同步每次运行一个。在同步期间，如果检测到任何新的提交，则会在活动同步结束时批处理并应用新的提交。到不同模板主要版本的同步是并行发生的。
- 如果您更改模板从中同步的分支，则正在从旧分支进行的任何同步先完成。然后，开始从新分支进行同步。

- 如果您更改模板从中同步的存储库，则正在从旧存储库进行的任何同步可能会失败或无法完成。这取决于它们处于同步的哪个阶段。

有关更多信息，请参阅 The AWS Proton Service API Reference。

主题

创建模板同步配置

了解如何使用 AWS Proton 创建模板同步配置。

创建模板同步配置的先决条件：

- 您已将存储库链接到 AWS Proton。
- 模板捆绑包位于您的存储库中。

存储库链接包含以下内容：

- 一个 CodeConnections 连接，它为 AWS Proton 授予权限以访问您的存储库以及订阅其通知。
- 一个服务相关角色。在您链接存储库时，将为您创建服务相关角色。

在创建第一个模板同步配置之前，将一个模板捆绑包推送到您的存储库，如以下目录布局中所示。

```
/templates/                                                 # subdirectory (optional)
/templates/my-env-template/                                 # template name
/templates/my-env-template/v1/                              # template version
/templates/my-env-template/v1/infrastructure/               # template bundle
/templates/my-env-template/v1/schema/
```

在创建第一个模板同步配置后，当您推送在新版本（例如 `/my-env-template/v2/`）中添加更新的模板捆绑包的提交时，将会自动创建新的模板版本。

```
/templates/                                                 # subdirectory (optional)
```

```
/templates/my-env-template/                                  # template name
/templates/my-env-template/v1/                               # template version
/templates/my-env-template/v1/infrastructure/                # template bundle
/templates/my-env-template/v1/schema/
/templates/my-env-template/v2/
/templates/my-env-template/v2/infrastructure/
/templates/my-env-template/v2/schema/
```

您可以在一次提交中包含一个或多个同步配置模板的新模板捆绑包版本。AWS Proton 为提交中包含的每个新模板捆绑包版本创建一个新模板版本。

在创建模板同步配置后，您仍然可以在控制台中或通过 AWS CLI 手动创建新的模板版本（从 S3 存储桶上传模板捆绑包）。模板同步仅适用于一个方向：从您的存储库到 AWS Proton。不会 同步手动创建的模板版本。

在设置模板同步配置后，AWS Proton 侦听存储库的更改。每次推送更改时，它都会查找与您的模板名称相同的任何目录。然后，它在该目录中查找任何看起来像主要版本的目录。AWS Proton 将模板捆绑包注册为相应的模板主要版本。新版本始终处于 DRAFT 状态。您可以使用控制台或 AWS CLI 发布新版本。

例如，假设您配置了一个名为 `my-env-template` 的模板以从分支 `main` 上的 `my-repo/templates` 同步，并具有以下布局。

```
/code
/code/service.go
README.md
/templates/
/templates/my-env-template/
/templates/my-env-template/v1/
/templates/my-env-template/v1/infrastructure/
/templates/my-env-template/v1/schema/
/templates/my-env-template/v2/
/templates/my-env-template/v2/infrastructure/
/templates/my-env-template/v2/schema/
```

AWS Proton 将 `/templates/my-env-template/v1/` 内容同步到 `my-env-template:1`，并将 `/templates/my-env-template/v2/` 内容同步到 `my-env-template:2`。如果这些主要版本尚不存在，则会创建这些版本。

AWS Proton 查找第一个与模板名称匹配的目录。您可以在创建或编辑模板同步配置时指定 `subdirectoryPath` 以限制目录 AWS Proton 搜索。例如，您可以为 `subdirectoryPath` 指定 `/production-templates/`。

您可以使用控制台或 CLI 创建模板同步配置。

AWS Management Console

使用控制台创建一个模板和模板同步配置。

1. 在 AWS Proton 控制台中，选择环境模板。
2. 选择创建环境模板。
3. 在创建环境模板页面上的模板选项部分中，选择创建用于预置新环境的模板。
4. 在模板捆绑包源部分中，选择从 Git 同步模板。
5. 在源代码存储库部分中：
   a. 对于存储库，选择包含您的模板捆绑包的链接存储库。
   b. 对于分支，选择要从中同步的存储库分支。
   c. （可选）对于模板捆绑包目录，输入一个目录名称以缩小模板捆绑包的搜索范围。
6. 在模板详细信息部分中。
   a. 输入模板名称。
   b. （可选）输入模板显示名称。
   c. （可选）输入环境模板的模板描述。
7. （可选）选中加密设置部分中的自定义加密设置 (高级) 复选框以提供您自己的加密密钥。
8. （可选）在标签部分中，选择添加新标签，并输入键和值以创建一个客户托管标签。
9. 选择创建环境模板。

   您现在位于一个新页面中，其中显示新环境模板的状态和详细信息。这些详细信息包括 AWS 托管标签和客户托管标签列表。在您创建 AWS Proton 资源时，AWS Proton 自动为您生成 AWS 托管标签。有关更多信息，请参阅AWS Proton 资源和标记。
10. 在模板详细信息页面中，选择同步选项卡以查看模板同步配置详细数据。
11. 选择模板版本选项卡以查看模板版本和状态详细信息。
12. 新环境模板的初始状态为草稿。您和具有 `proton:CreateEnvironment` 权限的其他人可以查看和访问该模板。执行下一步，以使该模板可供其他人使用。

13. 在模板版本部分中，选择刚创建的模板次要版本 (1.0) 左侧的单选按钮。或者，您可以在信息提醒中选择发布并跳过下一步。
14. 在模板版本部分中，选择发布。
15. 模板状态变为已发布。这是模板的最新版本（推荐版本）。
16. 在导航窗格中，选择环境模板以查看环境模板和详细信息列表。

创建服务模板和模板同步配置的过程是类似的。

AWS CLI

使用 AWS CLI 创建一个模板和模板同步配置。

1. 创建一个模板。在该示例中，创建了一个环境模板。

   运行以下命令。

   ```
   $ aws proton create-environment-template \
       --name "env-template"
   ```

   响应如下所示。

   ```
   {
       "environmentTemplate": {
           "arn": "arn:aws:proton:us-east-1:123456789012:environment-template/env-template",
           "createdAt": "2021-11-07T23:32:43.045000+00:00",
           "displayName": "env-template",
           "lastModifiedAt": "2021-11-07T23:32:43.045000+00:00",
           "name": "env-template",
           "status": "DRAFT",
           "templateName": "env-template"
       }
   }
   ```

2. 通过提供以下内容，使用 AWS CLI 创建模板同步配置：

   - 您要同步到的模板。在创建模板同步配置后，您仍然可以在控制台中或使用 AWS CLI 手动创建新版本。
   - 模板名称。
   - 模板类型。

- 您要从中同步的链接存储库。
- 链接的存储库提供商。
- 模板捆绑包所在的分支。
- （可选）包含模板捆绑包的目录的路径。默认情况下，AWS Proton 查找与您的模板名称匹配的第一个目录。

运行以下命令。

```
$ aws proton create-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --repository-name "myrepos/templates" \
    --repository-provider "GITHUB" \
    --branch "main" \
    --subdirectory "env-template/"
```

响应如下所示。

```
{
    "templateSyncConfigDetails": {
        "branch": "main",
        "repositoryName": "myrepos/templates",
        "repositoryProvider": "GITHUB",
        "subdirectory": "templates",
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

3. 要发布您的模板版本，请参阅注册并发布模板。

同步服务模板

前面的示例说明了如何同步环境模板。服务模板是类似的。要同步服务模板，您可以将一个名为 `.template-registration.yaml` 的附加文件添加到模板捆绑包的每个主要版本目录中。该文件包含 AWS Proton 在提交后创建服务模板版本时所需的额外详细信息。在您使用 AWS Proton 控制台或 API 明确创建服务模板版本时，您提供这些详细信息以作为输入，并且该文件替换这些输入以进行模板同步。

```
./templates/                                                    # subdirectory (optional)
 /templates/my-svc-template/                                    # service template name
 /templates/my-svc-template/v1/                                 # service template version
 /templates/my-svc-template/v1/.template-registration.yaml      # service template version
 properties
 /templates/my-svc-template/v1/instance_infrastructure/         # template bundle
 /templates/my-svc-template/v1/schema/
```

`.template-registration.yaml` 文件包含以下详细信息：

- 兼容的环境 [必需] - 基于这些环境模板和主要版本的环境与基于该服务模板版本的服务兼容。
- 支持的组件源 [可选] - 使用这些源的组件与基于该服务模板版本的服务兼容。如果未指定，则无法将组件附加到这些服务。有关组件的更多信息，请参阅组件。

该文件的 YAML 语法如下所示：

```
compatible_environments:
  - env-templ-name:major-version
  - ...
supported_component_sources:
  - DIRECTLY_DEFINED
```

指定一个或多个环境模板/主要版本组合。指定 `supported_component_sources` 是可选的，唯一支持的值为 `DIRECTLY_DEFINED`。

Example .template-registration.yaml

在该示例中，服务模板版本与 `my-env-template` 环境模板的主要版本 1 和 2 兼容。它还与 `another-env-template` 环境模板的主要版本 1 和 3 兼容。该文件未指定 `supported_component_sources`，因此，无法将组件附加到基于该服务模板版本的服务。

```
compatible_environments:
  - my-env-template:1
  - my-env-template:2
  - another-env-template:1
  - another-env-template:3
```

> **Note**
>
> 以前，AWS Proton 定义了一个不同的文件 (`.compatible-envs`) 以指定兼容的环境。AWS Proton 仍然支持该文件及其格式以保持向后兼容。我们建议不要再使用该文件，因为它无法进行扩展，并且无法支持较新的功能（例如组件）。

查看模板同步配置详细信息

使用控制台或 CLI 查看模板同步配置详细数据。

AWS Management Console

使用控制台查看模板同步配置详细信息。

1. 在导航窗格中，选择（环境或服务）模板。
2. 要查看详细数据，请选择您创建模板同步配置的模板的名称。
3. 在模板的详细信息页面中，选择同步选项卡以查看模板同步配置详细数据。

AWS CLI

使用 AWS CLI 查看同步的模板。

运行以下命令。

```
$ aws proton get-template-sync-config \
    --template-name "svc-template" \
    --template-type "SERVICE"
```

响应如下所示。

```
{
    "templateSyncConfigDetails": {
        "branch": "main",
        "repositoryProvider": "GITHUB",
        "repositoryName": "myrepos/myrepo",
        "subdirectory": "svc-template",
        "templateName": "svc-template",
        "templateType": "SERVICE"
    }
```

```
}
```

使用 AWS CLI 获取模板同步状态。

对于 `template-version`，输入模板主要版本。

运行以下命令。

```
$ aws proton get-template-sync-status \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --template-version "1"
```

编辑模板同步配置

您可以编辑除 `template-name` 和 `template-type` 以外的任何模板同步配置参数。

了解使用控制台或 CLI 编辑模板同步配置。

AWS Management Console

使用控制台编辑模板同步配置分支。

在模板列表中。

1. 在 AWS Proton 控制台中，选择（环境或服务）模板。
2. 在模板列表中，选择具有要编辑的模板同步配置的模板的名称。
3. 在模板详细信息页面中，选择模板同步选项卡。
4. 在模板同步详细信息部分中，选择编辑。
5. 在编辑页面上的源代码存储库部分中，为分支选择一个分支，然后选择保存配置。

AWS CLI

以下示例命令和响应说明了如何使用 CLI 编辑模板同步配置**branch**。

运行以下命令。

```
$ aws proton update-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
```

```
    --repository-provider "GITHUB" \
    --repository-name "myrepos/templates" \
    --branch "fargate" \
    --subdirectory "env-template"
```

响应如下所示。

```
{
    "templateSyncConfigDetails": {
        "branch": "fargate",
        "repositoryProvider": "GITHUB",
        "repositoryName": "myrepos/myrepo",
        "subdirectory": "templates",
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

您可以类似地使用 AWS CLI 更新同步的服务模板。

删除模板同步配置

使用控制台或 CLI 删除模板同步配置。

AWS Management Console

使用控制台删除模板同步配置。

1. 在模板详细信息页面中，选择同步选项卡。
2. 在同步详细信息部分中，选择断开。

AWS CLI

以下示例命令和响应说明了如何使用 AWS CLI 删除同步的模板配置。

运行以下命令。

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

响应如下所示。

```
{
    "templateSyncConfig": {
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

服务同步配置

通过服务同步，您可以使用 Git 配置和部署 AWS Proton 服务。您可以使用服务同步通过 Git 存储库中定义的配置管理 AWS Proton 服务的初始部署和更新。通过 Git，您可以使用版本跟踪和拉取请求等功能配置、管理和部署您的服务。服务同步将 AWS Proton 与 Git 结合使用，以帮助您预置通过 AWS Proton 模板定义和管理的标准化基础设施。它管理 Git 存储库中的服务定义并减少工具切换。与单独使用 Git 相比，AWS Proton 中的模板和部署标准化可以帮助您花更少的时间管理基础设施。AWS Proton 还为开发人员和平台团队提供了更高的透明度和可审核性。

AWS Proton OPS 文件

`proton-ops` 文件定义 AWS Proton 查找用于更新服务实例的规范文件的位置。它还定义了更新服务实例的顺序以及何时将更改从一个实例传播到另一个实例。

`proton-ops` 文件支持使用位于链接的存储库中的一个或多个规范文件同步服务实例。您可以在 `proton-ops` 文件中定义同步块以实现该目的，如以下示例中所示。

示例 ./configuration/proton-ops.yaml：

```
sync:
  services:
      frontend-svc:
          alpha:
              branch: dev
              spec: ./frontend-svc/test/frontend-spec.yaml
          beta:
              branch: dev
              spec: ./frontend-svc/test/frontend-spec.yaml
          gamma:
              branch: pre-prod
              spec: ./frontend-svc/pre-prod/frontend-spec.yaml
```

```
        prod-one:
            branch: prod
            spec: ./frontend-svc/prod/frontend-spec-second.yaml
        prod-two:
            branch: prod
            spec: ./frontend-svc/prod/frontend-spec-second.yaml
        prod-three:
            branch: prod
            spec: ./frontend-svc/prod/frontend-spec-second.yaml
```

在前面的示例中，frontend-svc 是服务名称，alpha、beta、gamma、prod-one、prod-two 和 prod-three 是实例。

spec 文件可以是 proton-ops 文件中定义的所有实例或一部分实例。不过，它必须至少在从中同步的分支和规范中定义了实例。如果在 proton-ops 文件中未使用特定分支和 spec 文件位置定义实例，则服务同步不会创建或更新这些实例。

以下示例显示了 spec 文件的外观。请记住，proton-ops 文件是从这些 spec 文件同步的。

示例**./frontend-svc/test/frontend-spec.yaml**：

```
proton: "ServiceSpec"
instances:
- name: "alpha"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "beta"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

示例**./frontend-svc/pre-prod/frontend-spec.yaml**：

```
proton: "ServiceSpec"
instances:
- name: "gamma"
```

```
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

示例**./frontend-svc/prod/frontend-spec-second.yaml**：

```
proton: "ServiceSpec"
instances:
- name: "prod-one"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "prod-two"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "prod-three"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

如果一个实例未同步，并且尝试同步该实例时持续出现问题，调用 GetServiceInstanceSyncStatus API 可能有助于解决该问题。

> **Note**
>
> 使用服务同步的客户仍然受到 AWS Proton 限制的约束。

阻止标记

通过使用 AWS Proton 服务同步同步您的服务，您可以更新服务规范并从 Git 存储库中创建和更新服务实例。不过，您可能有时需要通过AWS Management Console或 AWS CLI 手动更新服务或实例。

AWS Proton 可以帮助避免覆盖您通过AWS CLI或 AWS Management Console 所做的任何手动更改，例如更新服务实例或删除服务实例。为了实现该目的，在检测到手动更改时，AWS Proton 禁用服务同步以自动创建服务同步阻止标记。

要获取与某个服务关联的所有阻止标记，您必须按顺序为与该服务关联的每个 `serviceInstance` 执行以下操作：

- 仅使用 `serviceName` 调用 `getServiceSyncBlockerSummary` API。
- 使用 `serviceName` 和 `serviceInstanceName` 调用 `getServiceSyncBlockerSummary` API。

这会返回最近的阻止标记及其关联状态的列表。如果任何阻止标记标为 ACTIVE，您必须使用 `blockerId` 和 `resolvedReason` 为每个阻止标记调用 `UpdateServiceSyncBlocker` API 以解除这些阻止标记。

如果您手动更新或创建一个服务实例，则 AWS Proton 在该服务实例上创建一个服务同步阻止标记。AWS Proton 继续同步所有其他服务实例，但禁止同步该服务实例，直到解除了阻止标记。如果您从服务中删除一个服务实例，则 AWS Proton 在该服务上创建一个服务同步阻止标记。这会禁止 AWS Proton 同步任何服务实例，直到解除了阻止标记。

在创建所有活动阻止标记后，您必须使用 `blockerId` 和 `resolvedReason` 为每个活动阻止标记调用 `UpdateServiceSyncBlocker` API 以解除这些阻止标记。

您可以使用AWS Management Console导航到 AWS Proton 并选择服务同步选项卡，以确定是否禁用了服务同步。如果阻止了服务或服务实例，则会显示启用按钮。要启用服务同步，请选择启用。

主题

创建服务同步配置

您可以使用控制台或 AWS CLI 创建服务同步配置。

AWS Management Console

1. 在选择服务模板页面上，选择一个模板并选择配置。
2. 在配置服务页面上的服务详细信息部分中，输入新的服务名称。
3. （可选）输入服务的描述。
4. 在应用程序源代码存储库部分中，选择选择链接的 Git 存储库以选择一个已链接到 AWS Proton 的存储库。如果您还没有链接的存储库，请选择链接另一个 Git 存储库，然后按照创建存储库的链接中的说明进行操作。
5. 对于存储库，从列表中选择您的源代码存储库的名称。
6. 对于分支，从列表中选择您的源代码的存储库分支的名称。
7. （可选）在标签部分中，选择添加新标签，并输入键和值以创建一个客户托管标签。
8. 选择下一步。
9. 在配置服务实例页面上的服务定义源部分中，选择从 Git 同步您的服务。
10. 在服务定义文件部分中，如果您希望 AWS Proton 创建 `proton-ops` 文件，请选择我希望 AWS Proton 创建文件。在使用该选项时，AWS Proton 在您指定的位置中创建 `spec` 和 `proton-ops` 文件。选择我正在提供我自己的文件以创建您自己的 OPS 文件。
11. 在服务定义存储库部分中，选择选择链接的 Git 存储库以选择一个已链接到 AWS Proton 的存储库。
12. 对于存储库名称，从列表中选择您的源代码存储库的名称。
13. 对于 **`proton-ops`** 文件分支，从列表中选择您的分支名称，AWS Proton 将在其中放置您的 OPS 和规范文件。
14. 在服务实例部分中，将根据 `proton-ops` 文件中的值自动填充每个字段。
15. 选择下一步并检查您的输入。
16. 选择创建。

AWS CLI

使用 AWS CLI 创建服务同步配置

- 运行以下命令。

```
$ aws proton create-service-sync-config \
    --resource "service-arn" \
```

```
    --repository-provider "GITHUB" \
    --repository "example/proton-sync-service" \
    --ops-file-branch "main" \
    --proton-ops-file "./configuration/custom-proton-ops.yaml" (optional)
```

响应如下所示。

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

查看服务同步的配置详细信息

您可以使用控制台或 AWS CLI 查看服务同步的配置详细数据。

AWS Management Console

使用控制台查看服务同步的配置详细信息

1. 在导航窗格中，选择服务。
2. 要查看详细数据，请选择您创建服务同步配置的服务的名称。
3. 在服务的详细信息页面中，选择服务同步选项卡以查看服务同步的配置详细数据。

AWS CLI

使用 AWS CLI 获取一个同步的服务。

运行以下命令。

```
$ aws proton get-service-sync-config \
    --service-name "service name"
```

响应如下所示。

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

使用 AWS CLI 获取服务同步状态。

运行以下命令。

```
$ aws proton get-service-sync-status \
    --service-name "service name"
```

编辑服务同步配置

您可以使用控制台或 AWS CLI 编辑服务同步配置。

AWS Management Console

使用控制台编辑服务同步配置。

1. 在导航窗格中，选择服务。
2. 要查看详细数据，请选择您创建服务同步配置的服务的名称。
3. 在服务详细信息页面上，选择服务同步选项卡。
4. 在服务同步部分中，选择编辑。
5. 在编辑页面上，更新要编辑的信息，然后选择保存。

AWS CLI

以下示例命令和响应说明了如何使用 AWS CLI 编辑服务同步配置。

运行以下命令。

```
$ aws proton update-service-sync-config \
```

```
    --service-name "service name" \
    --repository-provider "GITHUB" \
    --repository "example/proton-sync-service" \
    --ops-file-branch "main" \
    --ops-file "./configuration/custom-proton-ops.yaml"
```

响应如下所示。

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

删除服务同步配置

您可以使用控制台或 AWS CLI 删除服务同步配置。

AWS Management Console

使用控制台删除服务同步配置

1. 在服务详细信息页面上，选择服务同步选项卡。
2. 在服务同步详细信息部分中，选择断开以断开连接您的存储库。在断开连接您的存储库后，我们不再从该存储库同步服务。

AWS CLI

以下示例命令和响应说明了如何使用 AWS CLI 删除服务同步配置。

运行以下命令。

```
$ aws proton delete-service-sync-config \
    --service-name "service name"
```

响应如下所示。

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

> **Note**
>
> 服务同步不会删除服务实例。它仅删除配置。

AWS Proton 环境

对于 AWS Proton，环境表示 AWS Proton 服务部署到的一组共享资源和策略。它们可以包含预计在 AWS Proton 服务实例之间共享的任何资源。这些资源可能包括 VPC、集群和共享负载均衡器或 API 网关。必须先创建一个 AWS Proton 环境，然后才能将服务部署到该环境中。

本节介绍了如何使用创建、查看、更新和删除操作管理环境。有关其他信息，请参阅 The AWS Proton Service API Reference。

主题

IAM 角色

通过使用 AWS Proton，可以为您拥有和管理的 AWS 资源提供 IAM 角色和 AWS KMS 密钥。然后，这些角色和密钥应用于开发人员拥有和管理的资源，并由这些资源使用。您可以创建一个 IAM 角色，以控制您的开发人员团队对 AWS Proton API 的访问。

AWS Proton 服务角色

在您创建新的环境时，您需要提供相关的 IAM 服务角色。该角色包含所需的所有权限，以更新环境模板和服务模板中定义的所有预置的基础设施。有关角色示例，请参阅使用 AWS CloudFormation 进行预置的 AWS Proton 服务角色。如果您使用环境账户连接和环境账户，您可以在选定的环境账户中创建角色。有关更多信息，请参阅 在一个账户中创建环境并在另一个账户中预置 和 环境账户连接：

如何提供该服务角色以及由谁担任该角色取决于您的环境的预置方法。

- AWS 托管式预置 - 您可以在创建环境时直接向 AWS Proton 提供角色，也可以通过账户连接间接提供角色。AWS Proton 在相关的账户中担任角色，以预置环境和服务基础设施。

- 自托管式预置 - 在拉取请求 (PR) 触发预置操作时，您负责配置预置自动化以使用相应凭证担任相应的角色。有关担任角色的示例 GitHub 操作，请参阅 "Configure AWS Credentials" Action For GitHub Actions 文档中的 Assuming a Role。

有关预置方法的更多信息，请参阅the section called “预置方法”。

创建环境

了解如何创建 AWS Proton 环境。

您可以通过以下两种方法之一创建 AWS Proton 环境：

- 使用标准环境模板 创建、管理和预置标准环境。AWS Proton 为您的环境预置基础设施。
- 使用客户托管环境模板 将 AWS Proton 连接到客户托管基础设施。您在 AWS Proton 外部预置自己的共享资源，然后提供 AWS Proton 可使用的预置输出。

在创建环境时，您可以选择多种预置方法之一。

- AWS 托管式预置 - 在单个账户中创建、管理和预置环境。AWS Proton 预置您的环境。

  该方法仅支持 CloudFormation 基础设施即代码 (IaC) 模板。
- 到另一个账户的 AWS 托管式预置 - 在单个管理账户中，创建和管理一个环境以在另一个具有环境账户连接的账户中进行预置。AWS Proton 在另一个账户中预置您的环境。有关更多信息，请参阅 在一个账户中创建环境并在另一个账户中预置 和 环境账户连接：

  该方法仅支持 CloudFormation IaC 模板。
- 自托管式预置 - AWS Proton 使用您自己的预置基础设施将预置拉取请求提交到链接的存储库。

  该方法仅支持 Terraform IaC 模板。
- CodeBuild 预置 - AWS Proton 使用 AWS CodeBuild 运行您提供的 Shell 命令。您的命令可以读取 AWS Proton 提供的输入，并负责预置或取消预置基础设施和生成输出值。该方法的模板捆绑包包括清单文件中的命令，以及这些命令可能需要的任何程序、脚本或其他文件。

  作为一个使用 CodeBuild 预置的示例，您可以包含使用 AWS Cloud Development Kit (AWS CDK) 预置 AWS 资源的代码，以及安装 CDK 和运行 CDK 代码的清单。

  有关更多信息，请参阅the section called “CodeBuild 捆绑包”。

> **Note**
>
> 您可以将 CodeBuild 预置与环境和服务一起使用。目前，您无法通过这种方法预置组件。

通过使用 AWS 托管式预置（在同一账户中预置以及预置到另一个账户），AWS Proton 直接进行调用以预置您的资源。

通过使用自托管式预置，AWS Proton 发出拉取请求以提供编译的 IaC 文件，您的 IaC 引擎使用这些文件以预置资源。

有关更多信息，请参阅the section called “预置方法”、the section called “模板捆绑包”和the section called “环境架构要求”。

主题

- 在同一账户中创建和预置标准环境
- 在一个账户中创建环境并在另一个账户中预置
- 使用自托管式预置创建和预置环境

在同一账户中创建和预置标准环境

使用控制台或 AWS CLI 在单个账户中创建和预置环境。预置是由 AWS 管理的。

AWS Management Console

使用控制台在单个账户中创建和预置环境

1. 在 AWS Proton 控制台中，选择环境。
2. 选择 Create environment（创建环境）。
3. 在选择环境模板页面中，选择一个模板并选择配置。
4. 在配置环境页面上的预置部分中，选择 AWS 托管式预置。
5. 在部署账户部分中，选择该 AWS 账户 账户。
6. 在配置环境页面上的环境设置部分中，输入一个环境名称。
7. （可选）输入环境的描述。
8. 在环境角色部分中，选择您在设置 AWS Proton 服务角色过程中创建的 AWS Proton 服务角色。

9. （可选）在组件角色部分中，选择一个服务角色，该角色允许直接定义的组件在环境中运行并缩小它们可以预置的资源范围。有关更多信息，请参阅组件。
10. （可选）在标签部分中，选择添加新标签，并输入键和值以创建一个客户托管标签。
11. 选择下一步。
12. 在配置环境自定义设置页面中，您必须输入 `required` 参数的值。您可以输入 `optional` 参数的值，或使用给定的默认值。
13. 选择下一步并检查您的输入。
14. 选择创建。

    查看环境详细信息和状态，以及您的环境的 AWS 托管标签和客户托管标签。
15. 在导航窗格中，选择环境。

    新页面显示您的环境列表以及状态和其他环境详细信息。

AWS CLI

使用 AWS CLI 在单个账户中创建和预置环境。

要创建环境，您需要指定 AWS Proton 服务角色 ARN、规范文件路径、环境名称、环境模板 ARN、主要版本和次要版本以及描述（可选）。

下一个示例显示 YAML 格式的规范文件，该文件指定环境模板架构文件中定义的两个输入的值。您可以使用 `get-environment-template-minor-version` 命令查看环境模板架构。

```
proton: EnvironmentSpec
spec:
  my_sample_input: "the first"
  my_other_sample_input: "the second"
```

运行以下命令以创建一个环境。

```
$ aws proton create-environment \
    --name "MySimpleEnv" \
    --template-name simple-env \
    --template-major-version 1 \
    --proton-service-role-arn "arn:aws:iam::123456789012:role/AWSProtonServiceRole"
 \
```

```
    --spec "file://env-spec.yaml"
```

响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2020-11-11T23:03:05.405000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2020-11-11T23:03:05.405000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "templateName": "simple-env"
    }
}
```

在创建新的环境后，您可以查看 AWS 和客户托管标签列表，如以下示例命令中所示。AWS Proton 自动为您生成 AWS 托管标签。您也可以使用 AWS CLI 修改和创建客户托管标签。有关更多信息，请参阅AWS Proton 资源和标记。

命令:

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

在一个账户中创建环境并在另一个账户中预置

可以使用控制台或 AWS CLI 在管理账户中创建标准环境，以在另一个账户中预置环境基础设施。预置是由 AWS 管理的。

在使用控制台或 CLI 之前，请完成以下步骤。

1. 找到管理和环境账户的 AWS 账户 ID，并复制它们以供以后使用。
2. 在环境账户中，创建一个 AWS Proton 服务角色，该角色具有要创建的环境的最低权限。有关更多信息，请参阅使用 AWS CloudFormation 进行预置的 AWS Proton 服务角色。

AWS Management Console

使用控制台在一个账户中创建环境并在另一个账户中进行预置。

1. 在环境账户中，创建一个环境账户连接，并使用该连接发送请求以连接到管理账户。
   a. 在 AWS Proton 控制台中，在导航窗格中选择环境账户连接。
   b. 在环境账户连接页面中，选择请求连接。

      > **Note**
      > 验证环境账户连接页面标题中列出的账户 ID 是否与您预先找到的环境账户 ID 匹配。

   c. 在请求连接页面上的环境角色部分中，选择现有的服务角色以及您为环境创建的服务角色的名称。
   d. 在连接到管理账户部分中，输入您的 AWS Proton 环境的管理账户 ID 和环境名称。复制该名称以供以后使用。
   e. 选择页面右下角的请求连接。
   f. 您的请求在发送到管理账户的环境连接表中显示为“待处理”，并且一个模态框说明了如何接受来自管理账户的请求。
2. 在管理账户中，接受来自环境账户的连接请求。
   a. 登录到您的管理账户，并在 AWS Proton 控制台中选择环境账户连接。
   b. 在环境账户连接页面上的环境账户连接请求表中，选择环境账户 ID 与您预先找到的环境账户 ID 匹配的环境账户连接。

      > **Note**
      > 验证环境账户连接页面标题中列出的账户 ID 是否与您预先找到的管理账户 ID 匹配。

   c. 选择 Accept (接受)。状态从“待处理”变为“已连接”。
3. 在管理账户中，创建一个环境。
   a. 在导航窗格中，选择环境模板。
   b. 在环境模板页面中，选择创建环境模板。

c. 在选择环境模板页面中，选择一个环境模板。
d. 在配置环境页面上的预置部分中，选择 AWS 托管式预置。
e. 在部署账户部分中，选择其他 AWS 账户。
f. 在环境详细信息部分中，选择您的环境账户连接和环境名称。
g. 选择下一步。
h. 填写表单并选择下一步，直至到达审核和创建页面。
i. 检查并选择创建环境。

AWS CLI

可以使用 AWS CLI 在一个账户中创建环境并在另一个账户中进行预置。

在环境账户中，创建一个环境账户连接，并运行以下命令以请求连接。

```
$ aws proton create-environment-account-connection \
    --environment-name "simple-env-connected" \
    --role-arn "arn:aws:iam::222222222222:role/service-role/env-account-proton-service-role" \
    --management-account-id "111111111111"
```

响应：

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-service-role",
        "status": "PENDING"
    }
}
```

在管理账户中，运行以下命令以接受环境账户连接请求。

```
$ aws proton accept-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

响应：

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

运行以下命令以查看您的环境账户连接。

```
$ aws proton get-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

响应：

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
```

```
    }
}
```

在管理账户中，运行以下命令以创建一个环境。

```
$ aws proton create-environment \
    --name "simple-env-connected" \
    --template-name simple-env-template \
    --template-major-version "1" \
    --template-minor-version "1" \
    --spec "file://simple-env-template/specs/original.yaml" \
    --environment-account-connection-id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:111111111111:environment/simple-env-
connected",
        "createdAt": "2021-04-28T23:02:57.944000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "environmentAccountConnectionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "lastDeploymentAttemptedAt": "2021-04-28T23:02:57.944000+00:00",
        "name": "simple-env-connected",
        "templateName": "simple-env-template"
    }
}
```

使用自托管式预置创建和预置环境

在您使用自托管式预置时，AWS Proton 使用您自己的预置基础设施将预置拉取请求提交到链接的存储库。拉取请求启动您自己的工作流（调用 AWS 服务）以预置基础设施。

自托管式预置注意事项：

- 在创建环境之前，设置一个用于自托管式预置的存储库资源目录。有关更多信息，请参阅AWS Proton 基础架构即代码文件。
- 在创建环境后，AWS Proton 等待接收有关基础设施预置状态的异步通知。您的预置代码必须使用 AWS Proton `NotifyResourceStateChange` API 将这些异步通知发送到 AWS Proton。

您可以在控制台中或通过 AWS CLI 使用自托管式预置。以下示例说明了如何使用 Terraform 进行自托管式预置。

AWS Management Console

使用控制台创建一个使用自托管式预置的 Terraform 环境。

1. 在 AWS Proton 控制台中，选择环境。
2. 选择 Create environment（创建环境）。
3. 在选择环境模板页面中，选择一个 Terraform 模板并选择配置。
4. 在配置环境页面上的预置部分中，选择自托管式预置。
5. 在预置存储库详细信息部分中：

   a. 如果您没有将预置存储库链接到 AWS Proton，请选择新存储库，选择存储库提供商之一，然后为 CodeStar 连接选择您的连接之一。

   > **Note**
   >
   > 如果您还没有连接到相关的存储库提供商账户，请选择添加新的 CodeStar 连接。接下来，创建一个连接，然后选择 CodeStar 连接菜单旁边的刷新按钮。您现在应该可以在菜单中选择您的新连接。

   如果您已将存储库链接到 AWS Proton，请选择现有存储库。

   b. 对于存储库名称，选择一个存储库。下拉菜单为现有存储库显示链接的存储库，或者为新存储库显示提供商账户中的存储库列表。

   c. 对于分支名称，选择存储库分支之一。
6. 在环境设置部分中，输入一个环境名称。
7. （可选）输入环境的描述。
8. （可选）在标签部分中，选择添加新标签，并输入键和值以创建一个客户托管标签。
9. 选择下一步。
10. 在配置环境自定义设置页面中，您必须输入 `required` 参数的值。您可以输入 `optional` 参数的值，或使用给定的默认值。
11. 选择下一步并检查您的输入。
12. 选择创建以发送一个拉取请求。

- 如果您批准拉取请求，则会进行部署。
- 如果您拒绝拉取请求，将取消创建环境。
- 如果拉取请求超时，则不会 完成创建环境。

13. 查看环境详细信息和状态，以及您的环境的 AWS 托管标签和客户托管标签。
14. 在导航窗格中，选择环境。

    新页面显示您的环境列表以及状态和其他环境详细信息。

AWS CLI

在您创建使用自托管式预置的环境时，您可以添加 `provisioningRepository` 参数并省略 `ProtonServiceRoleArn` 和 `environmentAccountConnectionId` 参数。

使用 AWS CLI 创建具有自托管式预置的 Terraform 环境。

1. 创建一个环境，并向存储库发送拉取请求以进行审核和批准。

   下一个示例显示一个 YAML 格式的规范文件，该文件根据环境模板架构文件定义两个输入的值。您可以使用 `get-environment-template-minor-version` 命令查看环境模板架构。

   规范：

   ```
   proton: EnvironmentSpec
   spec:
     ssm_parameter_value: "test"
   ```

   运行以下命令以创建一个环境。

   ```
   $ aws proton create-environment \
       --name "pr-environment" \
       --template-name "pr-env-template" \
       --template-major-version "1" \
       --provisioning-repository="branch=main,name=myrepos/env-
   repo,provider=GITHUB" \
       --spec "file://env-spec.yaml"
   ```

   响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T17:06:58.679000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-11-18T17:06:58.679000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
        "templateName": "pr-env-template"
    }
```

2. 检查请求。

   - 如果您批准请求，则会进行预置。
   - 如果您拒绝请求，将取消创建环境。
   - 如果拉取请求超时，则不会 完成创建环境。

3. 异步向 AWS Proton 提供预置状态。以下示例向 AWS Proton 通知预置成功。

```
$ aws proton notify-resource-deployment-status-change \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/pr-
environment" \
    --status "SUCCEEDED"
```

查看环境数据

您可以使用 AWS Proton 控制台或 AWS CLI 查看环境详细数据。

AWS Management Console

您可以使用 AWS Proton 控制台查看包含详细信息的环境列表以及包含详细数据的各个环境。

1. 要查看您的环境列表，请在导航窗格中选择环境。

2. 要查看详细数据，请选择一个环境的名称。

   查看您的环境详细数据。

AWS CLI

使用 AWS CLI get 或 list 获取或列出环境详细信息。

运行以下命令：

```
$ aws proton get-environment \
    --name "MySimpleEnv"
```

响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2020-11-11T23:03:05.405000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "lastDeploymentAttemptedAt": "2020-11-11T23:03:05.405000+00:00",
        "lastDeploymentSucceededAt": "2020-11-11T23:03:05.405000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\nspec:\n  my_sample_input: \"the first\"\n
 my_other_sample_input: \"the second\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "simple-env"
    }
}
```

更新环境

如果 AWS Proton 环境与一个环境账户连接关联，则不要 更新或包含 `protonServiceRoleArn` 参数以更新或连接到环境账户连接。

只有在满足以下两个条件时，您才能更新为新的环境账户连接：

- 环境账户连接是在创建当前环境账户连接的同一环境账户中创建的。

- 环境账户连接与当前环境相关联。

如果环境与环境账户连接不 关联，则不要 更新或包含 environmentAccountConnectionId 参数。

您可以更新 environmentAccountConnectionId 或 protonServiceRoleArn 参数和值。您无法同时更新两者。

如果您的环境使用自托管式预置，请不要 更新 provisioning-repository 参数并省略 environmentAccountConnectionId 和 protonServiceRoleArn 参数。

可以使用 4 种模式更新环境，如以下列表中所述。在使用 AWS CLI 时，deployment-type 字段定义模式。在使用控制台时，这些模式映射到操作下拉列表中的编辑、更新、更新次要和更新主要操作。

NONE

在该模式下，不会 进行部署。仅更新请求的元数据参数。

CURRENT_VERSION

在该模式下，将使用您提供的新规范部署和更新环境。仅更新请求的参数。在使用该 deployment-type 时，不要 包含次要或主要版本参数。

MINOR_VERSION

在该模式下，默认使用当前使用的主要版本的已发布推荐（最新）次要版本部署和更新环境。您也可以指定当前使用的主要版本的不同次要版本。

MAJOR_VERSION

在该模式下，默认使用当前模板的已发布推荐（最新）主要版本和次要版本部署和更新环境。您也可以指定高于正在使用的主要版本的不同主要版本和次要版本（可选）。

主题

- 更新 AWS 托管式预置环境
- 更新自托管式预置环境
- 取消正在进行的环境部署

更新 AWS 托管式预置环境

仅使用 AWS CloudFormation 预置的环境支持标准预置。

使用控制台或 AWS CLI 更新您的环境。

AWS Management Console

使用控制台更新一个环境，如以下步骤中所示。

1. 选择以下 2 个步骤之一。
   a. 在环境列表中。
      i. 在 AWS Proton 控制台中，选择环境。
      ii. 在环境列表中，选择要更新的环境左侧的单选按钮。
   b. 在控制台环境详细信息页面中。
      i. 在 AWS Proton 控制台中，选择环境。
      ii. 在环境列表中，选择要更新的环境名称。
2. 选择接下来的 4 个步骤之一以更新您的环境。
   a. 进行不需要部署环境的编辑。
      i. 例如，更改描述。

         选择编辑。
      ii. 填写表单并选择下一步。
      iii. 检查您的编辑内容并选择更新。
   b. 仅更新元数据输入。
      i. 选择操作，然后选择更新。
      ii. 填写表单并选择编辑。
      iii. 填写表单并选择下一步，直至到达审核页面。
      iv. 检查您的更新内容并选择更新。
   c. 将其环境模板更新为新的次要版本。
      i. 选择操作，然后选择更新次要。

ii. 填写表单并选择下一步。

iii. 填写表单并选择下一步，直至到达审核页面。

iv. 检查您的更新内容并选择更新。

d. 将其环境模板更新为新的主要版本。

i. 选择操作，然后选择更新主要。

ii. 填写表单并选择下一步。

iii. 填写表单并选择下一步，直至到达审核页面。

iv. 检查您的更新内容并选择更新。

AWS CLI

可以使用 AWS Proton AWS CLI 将环境更新为新的次要版本。

运行以下命令以更新您的环境：

```
$ aws proton update-environment \
        --name "MySimpleEnv" \
        --deployment-type "MINOR_VERSION" \
        --template-major-version "1" \
        --template-minor-version "1" \
        --proton-service-role-arn arn:aws:iam::123456789012:role/service-
role/ProtonServiceRole \
        --spec "file:///spec.yaml"
```

响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:29:55.472000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
```

```
        "templateName": "simple-env"
    }
}
```

运行以下命令以获取并确认状态：

```
$ aws proton get-environment \
        --name "MySimpleEnv"
```

响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "MySimpleEnv",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

更新自托管式预置环境

仅使用 Terraform 预置的环境支持自托管式预置。

使用控制台或 AWS CLI 更新您的环境。

AWS Management Console

使用控制台更新一个环境，如以下步骤中所示。

1. 选择以下 2 个步骤之一。

   a. 在环境列表中。

      i. 在 AWS Proton 控制台中，选择环境。

      ii. 在环境列表中，选择要更新的环境模板左侧的单选按钮。

   b. 在控制台环境详细信息页面中。

      i. 在 AWS Proton 控制台中，选择环境。

      ii. 在环境列表中，选择要更新的环境名称。

2. 选择接下来的 4 个步骤之一以更新您的环境。

   a. 进行不需要部署环境的编辑。

      i. 例如，更改描述。

         选择编辑。

      ii. 填写表单并选择下一步。

      iii. 检查您的编辑内容并选择更新。

   b. 仅更新元数据输入。

      i. 选择操作，然后选择更新。

      ii. 填写表单并选择编辑。

      iii. 填写表单并选择下一步，直至到达审核页面。

      iv. 检查您的更新内容并选择更新。

   c. 将其环境模板更新为新的次要版本。

      i. 选择操作，然后选择更新次要。

      ii. 填写表单并选择下一步。

      iii. 填写表单并选择下一步，直至到达审核页面。

      iv. 检查您的更新内容并选择更新。

d. 将其环境模板更新为新的主要版本。

i. 选择操作，然后选择更新主要。

ii. 填写表单并选择下一步。

iii. 填写表单并选择下一步，直至到达审核页面。

iv. 检查您的更新内容并选择更新。

AWS CLI

使用 AWS CLI 将 Terraform 环境更新为具有自托管式预置的新次要版本。

1. 运行以下命令以更新您的环境：

```
$ aws proton update-environment \
    --name "pr-environment" \
    --deployment-type "MINOR_VERSION" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --provisioning-repository "branch=main,name=myrepos/env-
repo,provider=GITHUB" \
    --spec "file://env-spec-mod.yaml"
```

响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T21:09:15.745000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-11-18T21:25:41.998000+00:00",
        "lastDeploymentSucceededAt": "2021-11-18T21:09:15.745000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
```

```
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "pr-env-template"
    }
}
```

2. 运行以下命令以获取并确认状态：

```
$ aws proton get-environment \
    --name "pr-environment"
```

响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T21:09:15.745000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "lastDeploymentAttemptedAt": "2021-11-18T21:25:41.998000+00:00",
        "lastDeploymentSucceededAt": "2021-11-18T21:25:41.998000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
        "spec": "proton: EnvironmentSpec\nspec:\n   ssm_parameter_value: \"test
\"\n ssm_another_parameter_value: \"update\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "pr-env-template"
    }
}
```

3. 检查 AWS Proton 发送的拉取请求。

- 如果您批准请求，则会进行预置。
- 如果您拒绝请求，将取消创建环境。
- 如果拉取请求超时，则不会完成创建环境。

4. 向 AWS Proton 提供预置状态。

```
$ aws proton notify-resource-deployment-status-change \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/pr-environment" \
    --status "SUCCEEDED"
```

取消正在进行的环境部署

如果 deploymentStatus 为 IN_PROGRESS，您可以尝试取消环境更新部署。AWS Proton 将尝试取消部署。不能 保证成功取消。

在您取消更新部署时，AWS Proton 尝试取消部署，如以下步骤中列出的一样。

对于 AWS 托管式预置，AWS Proton 执行以下操作：

- 将部署状态设置为 CANCELLING。
- 停止正在进行的部署，并删除状态为 IN_PROGRESS 时部署创建的任何新资源。
- 将部署状态设置为 CANCELLED。
- 将资源状态恢复为开始部署之前的状态。

对于自托管式预置，AWS Proton 执行以下操作：

- 尝试关闭拉取请求，以防止将更改合并到存储库中。
- 如果已成功关闭拉取请求，则将部署状态设置为 CANCELLED。

有关如何取消环境部署的说明，请参阅 AWS Proton API Reference 中的 CancelEnvironmentDeployment。

您可以使用控制台或 CLI 取消正在进行的环境。

AWS Management Console

使用控制台取消环境更新部署，如以下步骤中所示。

1. 在 AWS Proton 控制台中，在导航窗格中选择环境。
2. 在环境列表中，选择包含要取消的部署更新的环境名称。

3. 如果您的更新部署状态为进行中，请在环境详细信息页面中选择操作，然后选择取消部署。
4. 一个模态框提示您确认是否要取消。选择取消部署。
5. 您的更新部署状态设置为正在取消，然后设置为已取消以完成取消。

AWS CLI

使用 AWS Proton AWS CLI 取消将 IN_PROGRESS 环境部署更新为新的次要版本 2。

在用于该示例的模板中包含一个等待条件，以便在更新部署成功之前开始取消。

运行以下命令以取消更新：

```
$ aws proton cancel-environment-deployment \
        --environment-name "MySimpleEnv"
```

响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "CANCELLING",
        "lastDeploymentAttemptedAt": "2021-04-02T18:15:10.243000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

运行以下命令以获取并确认状态：

```
$ aws proton get-environment \
        --name "MySimpleEnv"
```

响应：


```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "CANCELLED",
        "deploymentStatusMessage": "User initiated cancellation.",
        "lastDeploymentAttemptedAt": "2021-04-02T18:15:10.243000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```


删除环境

您可以使用 AWS Proton 控制台或 AWS CLI 删除 AWS Proton 环境。

> **Note**
>
> 您无法删除具有任何关联组件的环境。要删除这样的环境，您应该先删除环境中运行的所有组件。有关组件的更多信息，请参阅组件。

AWS Management Console

使用控制台删除一个环境，如以下两个选项中所述。

在环境列表中。

1. 在 AWS Proton 控制台中，选择环境。
2. 在环境列表中，选择要删除的环境左侧的单选按钮。
3. 选择操作，然后选择删除。

4. 一个模态框提示您确认删除操作。
5. 按照说明进行操作并选择是，删除。

在环境详细信息页面中。

1. 在 AWS Proton 控制台中，选择环境。
2. 在环境列表中，选择要删除的环境名称。
3. 在环境详细信息页面中，选择操作，然后选择删除。
4. 一个模态框提示您确认是否要删除。
5. 按照说明进行操作并选择是，删除。

AWS CLI

使用 AWS CLI 删除一个环境。

如果在一个环境中部署了服务或服务实例，请不要 删除该环境。

运行以下命令：

```
$ aws proton delete-environment \
    --name "MySimpleEnv"
```

响应：

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "DELETE_IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

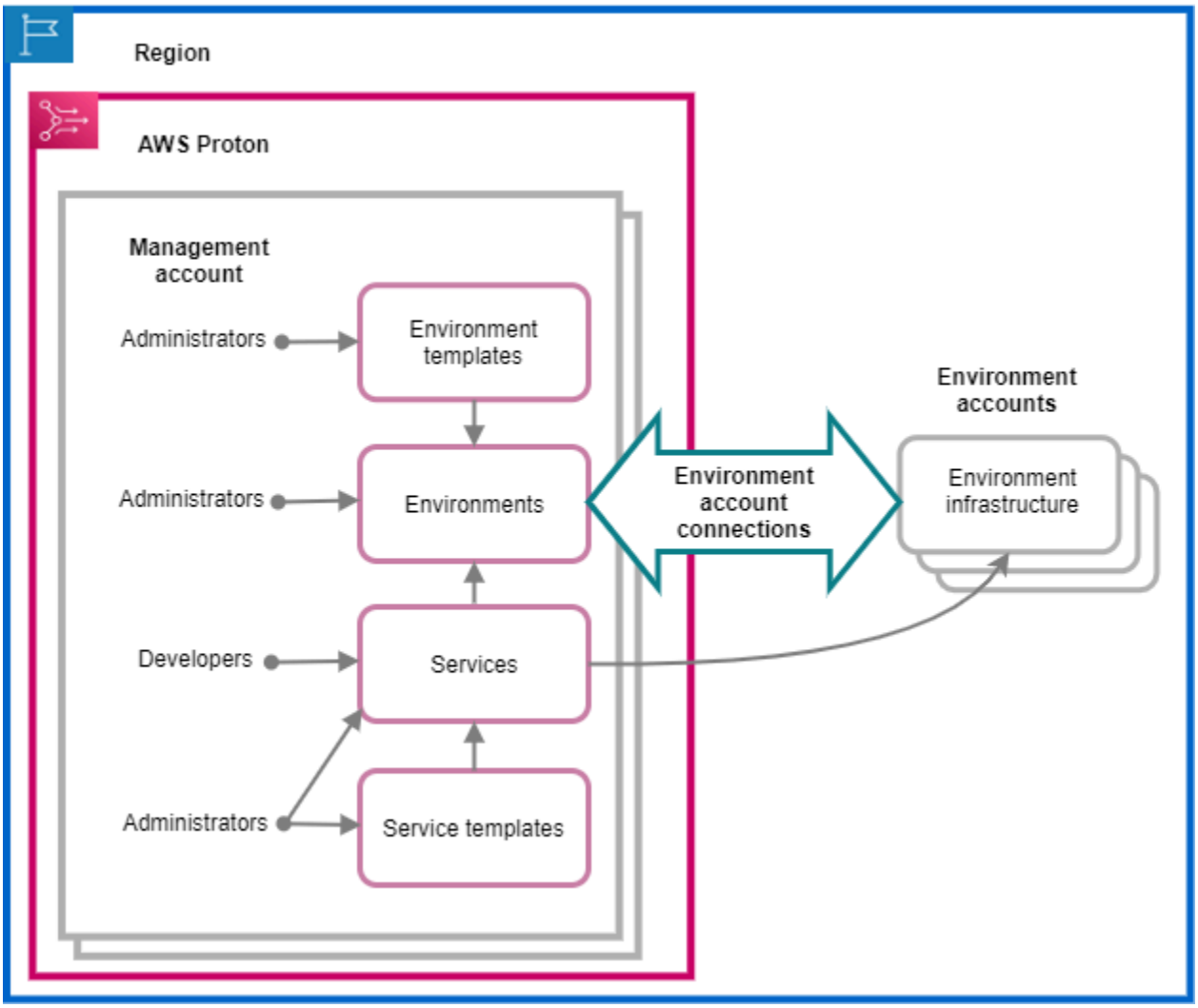
环境账户连接

概述

了解如何在一个账户中创建和管理 AWS Proton 环境，并在另一个账户中预置其基础设施资源。这有助于在大规模操作中提高可见性和效率。环境账户连接仅支持使用 AWS CloudFormation 基础设施即代码的标准预置。

> **Note**
>
> 本主题中的信息与配置了 AWS 托管式预置 的环境相关。对于配置了自托管式预置 的环境，AWS Proton 不会直接预置您的基础设施。相反，它将拉取请求 (PR) 发送到您的存储库以进行预置。您负责确保您的自动化代码担任正确的身份和角色。
> 有关预置方法的更多信息，请参阅the section called “预置方法”。

术语



通过使用 AWS Proton 环境账户连接，您可以从一个账户中创建 AWS Proton 环境，并在另一个账户中预置其基础设施。

管理账户

您作为管理员可以在该单一账户中创建一个 AWS Proton 环境，该环境在另一个环境账户 中预置基础设施资源。

环境账户

在另一个账户中创建 AWS Proton 环境时，将在环境账户中预置环境基础设施。

环境账户连接

管理账户 和环境账户 之间的安全双向连接。它维护授权和权限，在以下几节中进一步中介绍了该内容。

在特定区域的环境账户中创建一个环境账户连接时，仅同一区域中的管理账户可以看到和使用该环境账户连接。这意味着，在管理账户中创建的 AWS Proton 环境和在环境账户中预置的环境基础设施必须位于同一区域中。

环境账户连接注意事项

- 对于要在环境账户中预置的每个环境，您需要建立一个环境账户连接。
- 有关环境账户连接配额的信息，请参阅AWS Proton 配额。

标记

在环境账户中，使用控制台或 AWS CLI 查看和管理环境账户连接客户托管标签。不会 为环境账户连接生成 AWS 托管标签。有关更多信息，请参阅标记。

在一个账户中创建环境并在另一个账户中预置其基础设施

要从单个管理账户中创建和预置环境，请为您计划创建的环境设置一个环境账户。

在环境账户中启动并创建连接。

在环境账户中，创建一个 AWS Proton 服务角色，其范围缩小到仅预置环境基础设施资源所需的权限。有关更多信息，请参阅使用 AWS CloudFormation 进行预置的 AWS Proton 服务角色。

然后，创建一个环境账户连接请求并将其发送到您的管理账户。如果接受了该请求，AWS Proton 可以使用关联的 IAM 角色，该角色允许在关联的环境账户中预置环境资源。

在管理账户中，接受或拒绝环境账户连接。

在管理账户中，接受或拒绝环境账户连接请求。您无法 从管理账户中删除环境账户连接。

如果您接受该请求，AWS Proton 可以使用关联的 IAM 角色，该角色允许在关联的环境账户中预置资源。

环境基础设施资源是在关联的环境账户中预置的。您只能使用 AWS Proton API 从您的管理账户中访问和管理您的环境及其基础设施资源。有关更多信息，请参阅 在一个账户中创建环境并在另一个账户中预置 和 更新环境：

在拒绝请求后，您将无法 接受或使用拒绝的环境账户连接。

> **Note**
>
> 您无法 拒绝已连接到环境的环境账户连接。要拒绝环境账户连接，您必须先删除关联的环境。

在环境账户中，访问预置的基础设施资源。

在环境账户中，您可以查看和访问预置的基础设施资源。例如，如果需要，您可以使用 CloudFormation API 操作监控和清理堆栈。您无法使用 AWS Proton API 操作访问或管理用于预置基础设施资源的 AWS Proton 环境。

在环境账户中，您可以删除在环境账户中创建的环境账户连接。您无法 接受或拒绝这些连接。如果删除 AWS Proton 环境正在使用的环境账户连接，AWS Proton 将无法管理环境基础设施资源，直到环境账户和指定环境接受了新的环境连接。您负责清理没有环境连接的预置资源。

使用控制台或 CLI 管理环境账户连接

您可以使用控制台或 CLI 创建和管理环境账户连接。

AWS Management Console

使用控制台创建一个环境账户连接，并向管理账户发送请求，如以下步骤中所示。

1. 确定您计划在管理账户中创建的环境名称，或选择需要环境账户连接的现有环境名称。
2. 在环境账户中，在 AWS Proton 控制台上的导航窗格中选择环境账户连接。
3. 在环境账户连接页面中，选择请求连接。

   > **Note**
   >
   > 验证环境账户连接页面标题中列出的账户 ID。确保它与您希望在其中预置指定环境的环境账户的账户 ID 匹配。

4. 在请求连接页面中：

   a. 在连接到管理账户部分中，输入您在步骤 1 中输入的管理账户 ID 和环境名称。

   b. 在环境角色部分中，选择新服务角色，AWS Proton 自动为您创建一个新角色。或者，选择现有的服务角色和您以前创建的服务角色的名称。

> **Note**
>
> AWS Proton 自动为您创建的角色具有广泛的权限。我们建议您将角色范围缩小到预置环境基础设施资源所需的权限。有关更多信息，请参阅使用 AWS CloudFormation 进行预置的 AWS Proton 服务角色。

c. （可选）在标签部分中，选择添加新标签，为您的环境账户连接创建一个客户托管标签。

d. 选择请求连接。

5. 您的请求在发送到管理账户的环境连接表中显示为“待处理”，并且一个模态框让您知道如何从管理账户中接受该请求。

接受或拒绝环境账户连接请求。

1. 在管理账户中，在 AWS Proton 控制台上的导航窗格中选择环境账户连接。
2. 在环境账户连接页面上的环境账户连接请求表中，选择要接受或拒绝的环境连接请求。

> **Note**
>
> 验证环境账户连接页面标题中列出的账户 ID。确保它与要拒绝的环境账户连接的关联管理账户的账户 ID 匹配。在拒绝该环境账户连接后，您将无法 接受或使用拒绝的环境账户连接。

3. 选择拒绝或接受。
   - 如果您选择拒绝，状态将从待处理变为已拒绝。
   - 如果您选择接受，状态将从待处理变为已连接。

删除环境账户连接。

1. 在环境账户中，在 AWS Proton 控制台上的导航窗格中选择环境账户连接。

> **Note**
>
> 验证环境账户连接页面标题中列出的账户 ID。确保它与要拒绝的环境账户连接的关联管理账户的账户 ID 匹配。在删除该环境账户连接后，AWS Proton 无法 管理环境账户

中的环境基础设施资源。只有在管理账户接受环境账户和指定环境的新环境账户连接后，它才能管理该资源。

2. 在环境账户连接页面上的已发送连接至管理账户的请求部分中，选择删除。
3. 一个模态框提示您确认是否要删除。选择 Delete (删除)。

AWS CLI

确定您计划在管理账户中创建的环境名称，或选择需要环境账户连接的现有环境名称。

在环境账户中创建环境账户连接。

运行以下命令：

```
$ aws proton create-environment-account-connection \
    --environment-name "simple-env-connected" \
    --role-arn "arn:aws:iam::222222222222:role/service-role/env-account-proton-service-role" \
    --management-account-id "111111111111"
```

响应：

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-service-role",
        "status": "PENDING"
    }
}
```

在管理账户中接受或拒绝环境账户连接，如以下命令和响应中所示。

> **Note**
>
> 在拒绝该环境账户连接后，您将无法接受或使用拒绝的环境账户连接。

如果您指定拒绝，状态将从待处理变为已拒绝。

如果您指定接受，状态将从待处理变为已连接。

运行以下命令以接受环境账户连接：

```
$ aws proton accept-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

响应：

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

运行以下命令以拒绝环境账户连接：

```
$ aws proton reject-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

响应：

```
{
```

```
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:us-east-1:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "status": "REJECTED",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-reject",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role"
    }
}
```

查看环境账户的连接。您可以使用 get 或 list 获取或列出环境账户连接。

运行以下 get 命令：

```
$ aws proton get-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

响应：

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

删除环境账户中的环境账户连接。

> **Note**
>
> 如果您删除该环境账户连接，AWS Proton 将无法管理环境账户中的环境基础设施资源，直到环境账户和指定环境接受了新的环境连接。您负责清理没有环境连接的预置资源。

运行以下命令：

```
$ aws proton delete-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

响应：

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:us-east-1:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

客户托管环境

在客户托管环境中，您可以使用已部署为 AWS Proton 环境的现有基础设施（例如 VPC）。在使用客户托管环境时，您可以在 AWS Proton 外部预置自己的共享资源。不过，您仍然可以允许 AWS Proton 在部署 AWS Proton 服务时将相关的预置输出作为这些服务的输入。如果输出可能发生变化，则 AWS Proton 能够接受更新。不过，AWS Proton 无法直接更改环境，因为预置是在 AWS Proton 外部管理的。

在创建环境后，您负责向 AWS Proton 提供与 AWS Proton 部署环境时创建的输出相同的输出，例如 Amazon ECS 集群名称或 Amazon VPC ID。

通过使用该功能，您可以通过 AWS Proton 服务模板将 AWS Proton 服务资源部署到该环境以及更新这些资源。不过，不会通过模板更新在 AWS Proton 中修改环境本身。您负责执行环境更新以及在 AWS Proton 中更新这些输出。

您可以在一个账户中具有多个环境，这些环境是 AWS Proton 托管环境和客户托管环境的混合。您也可以链接第二个账户，并使用主账户中的 AWS Proton 模板在第二个链接账户中执行环境和服务部署和更新。

如何使用客户托管环境

管理员需要做的第一件事是，注册导入的客户托管环境模板。不要在模板捆绑包中提供清单或基础设施文件。仅提供架构。

下面的架构概述了使用 Open API 格式的输出列表，并复制了 AWS CloudFormation 模板的输出。

> **Important**
>
> 输出仅允许使用字符串输入。

以下示例是相应 Fargate 模板的 AWS CloudFormation 模板的输出部分片段。

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

导入的相应 AWS Proton 环境的架构类似于以下内容。不要在架构中提供默认值。

```
schema:
  format:
    openapi: "3.0.0"
  environment_input_type: "EnvironmentOutput"
  types:
```

```
    EnvironmentOutput:
      type: object
      description: "Outputs of the environment"
      properties:
        ClusterName:
          type: string
          description: "The name of the ECS cluster"
        ECSTaskExecutionRole:
          type: string
          description: "The ARN of the ECS role"
        VpcId:
          type: string
          description: "The ID of the VPC that this stack is deployed in"
[...]
```

在注册模板时，您指示该模板是导入的，并提供捆绑包的 Amazon S3 存储桶位置。在将模板置于草稿状态之前，AWS Proton 验证架构是否仅包含 `environment_input_type` 而不包含 AWS CloudFormation 模板参数。

您提供以下内容以创建一个导入的环境。

- 在进行部署时使用的 IAM 角色。
- 包含所需输出值的规范。

您可以使用类似于常规环境部署的过程，通过控制台或 AWS CLI 提供角色和规范。

创建 CodeBuild 预置角色

基础设施即代码 (IaaC) 工具（例如 AWS CloudFormation 和 Terraform）需要具有很多不同类型的 AWS 资源的权限。例如，如果 IaaC 模板声明一个 Amazon S3 存储桶，它需要具有创建、读取、更新和删除 Amazon S3 存储桶的权限。将角色限制为所需的最低权限被视为一种安全最佳实践。鉴于 AWS 资源范围很广，为 IaaC 模板创建最低权限策略是非常困难的，特别是这些模板管理的资源以后可能会发生变化。例如，在您最近编辑 AWS Proton 管理的模板时，您添加了 RDS 数据库资源。

配置正确的权限有助于顺利部署 IaC。AWS ProtonCodeBuild 预置在位于客户账户的 CodeBuild 项目中执行客户提供的任意 CLI 命令。通常，这些命令使用基础设施即代码 (IaaC) 工具（例如 AWS CDK）创建和删除基础设施。如果部署的 AWS 资源的模板使用 CodeBuild 预置，则 AWS 在 AWS 管理的 CodeBuild 项目中开始构建。将为 CodeBuild 传递一个角色，CodeBuild 担任该角色以执行命令。该角色称为 CodeBuild 预置角色，它是由客户提供的，并包含预置基础设施所需的权限。只能由 CodeBuild 担任该角色，甚至 AWS Proton 也无法担任该角色。

创建 角色

可以在 IAM 控制台或 AWS CLI 中创建 CodeBuild 预置角色。要在 AWS CLI 中创建该角色，请运行以下命令：

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-
policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AWSProtonCodeBuildProvisioningBasicAccess
```

这还会附加 AWSProtonCodeBuildProvisioningBasicAccess，其中包含 CodeBuild 服务运行构建所需的最低权限。

如果您更喜欢使用控制台，请在创建角色时确保满足以下条件：

1. 对于可信实体，选择 AWS 服务，然后选择 CodeBuild。
2. 在“添加权限”步骤中，选择 AWSProtonCodeBuildProvisioningBasicAccess 以及您希望附加的任何其他策略。

管理员访问权限

如果您将 AdministratorAccess 策略附加到 CodeBuild 预置角色，这会保证任何 IaaC 模板不会由于缺少权限而失败。这也意味着，任何可以创建环境模板或服务模板的人都可以执行管理员级别的操作，即使该用户不是管理员。AWS Proton 建议不要将 AdministatorAccess 与 CodeBuild 预置角色一起使用。如果您决定将 AdministratorAccess 与 CodeBuild 预置角色一起使用，请在沙盒环境中执行该操作。

您可以在 IAM 控制台中使用 AdministratorAccess 创建一个角色，或执行以下命令以创建该角色：

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-
policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AdministratorAccess
```

创建最小范围的角色

如果要创建具有最低权限的角色，可以使用多种方法：

- 使用管理员权限进行部署，然后缩小角色范围。我们建议使用 IAM Access Analyzer。
- 使用托管策略授予您计划使用的服务的访问权限。

AWS CDK

如果您将 AWS CDK 与 AWS Proton 一起使用，并且已在每个环境账户/区域上运行 `cdk bootstrap`，则 `cdk deploy` 的角色已存在。在这种情况下，请将以下策略附加到 CodeBuild 预置角色：

```
{
    "Action": "sts:AssumeRole",
    "Resource": [
        "arn:aws:iam::account-id:role/cdk-*-deploy-role-*",
        "arn:aws:iam::account-id:role/cdk-*-file-publishing-role-*"
    ],
    "Effect": "Allow"
}
```

自定义 VPC

如果您决定在自定义 VPC 中运行 CodeBuild，则需要在您的 CodeBuild 角色中具有以下权限：

```
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:region:account-id:network-interface/*",
        "arn:aws:ec2:region:account-id:subnet/*",
        "arn:aws:ec2:region:account-id:security-group/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:DeleteNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:region:account-id:*/*"
    ]
```

```
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeVpcs"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterfacePermission"
    ],
    "Resource": "arn:aws:ec2:region:account-id:network-interface/*",
    "Condition": {
        "StringEquals": {
            "ec2:AuthorizedService": "codebuild.amazonaws.com"
        }
    }
}
```

您也可以使用 AmazonEC2FullAccess 托管策略，但其中可能包含您不需要的权限。要使用 CLI 附加托管策略，请运行以下命令：

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-
policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AdministratorAccess
```

AWS Proton 服务

AWS Proton 服务是服务模板的实例化形式，通常包括多个服务实例和一个管道。AWS Proton 服务实例是服务模板在特定环境中的实例化形式。服务模板是 AWS Proton 服务的基础设施和可选服务管道的完整定义。

在部署服务实例后，您可以通过触发 CI/CD 管道的源代码推送更新这些实例，或者将服务更新为其服务模板的新版本以更新这些实例。AWS Proton 在发布新版本的服务模板时提示您，以便将服务更新为新版本。在更新您的服务时，AWS Proton 重新部署服务和服务实例。

本章介绍了如何使用创建、查看、更新和删除操作管理服务。有关其他信息，请参阅 The AWS Proton Service API Reference。

主题

创建服务

要使用 AWS Proton 部署应用程序，您作为开发人员需要创建一个服务并提供以下输入。

1. 由平台团队发布的 AWS Proton 服务模板的名称。
2. 服务的名称。
3. 您要部署的服务实例数量。
4. 您要使用的一组环境。
5. 到代码存储库的连接 - 如果您使用的服务模板包含服务管道（可选）。

服务中包含的内容

在创建 AWS Proton 服务时，您可以从两种不同类型的服务模板中进行选择：

- 包含服务管道的服务模板（默认）。
- 不 包含服务管道的服务模板。

在创建服务时，您必须至少创建一个服务实例。

服务实例和可选管道与服务相关联。您只能在服务创建 和删除 操作的上下文中创建或删除管道。要了解如何在服务中添加和删除实例，请参阅编辑服务。

> **Note**
>
> 您的环境配置为 AWS 托管式预置或自托管式预置。AWS Proton 使用与环境相同的预置方法在环境中预置服务。创建或更新服务实例的开发人员看不到差异，并且他们在这两种情况下的体验是相同的。
> 有关预置方法的更多信息，请参阅the section called “预置方法”。

服务模板

服务模板具有主要版本和次要版本。在使用控制台时，您可以选择服务模板的最新 Recommended 主要版本和次要版本。在您使用 AWS CLI 并且仅指定服务模板的主要版本时，您隐式指定了最新的 Recommended 次要版本。

下面介绍了主要和次要模板版本的区别及其用途。

- 在获得平台团队成员的批准后，新的模板版本就会变为Recommended版本。这意味着，新服务是使用该版本创建的，并提示您将现有服务更新为新版本。
- 通过 AWS Proton，平台团队可以自动将服务实例更新为新的服务模板次要版本。次要版本必须向后兼容。
- 由于主要版本要求您在更新过程中提供新输入，因此，您需要将服务更新为其服务模板的主要版本。主要版本不 向后兼容。

创建服务

以下过程说明了如何使用 AWS Proton 控制台或 AWS CLI 创建具有或没有服务管道的服务。

AWS Management Console

创建一个服务，如以下控制台步骤中所示。

1. 在 AWS Proton 控制台中，选择服务。
2. 选择 Create service。
3. 在选择服务模板页面中，选择一个模板，然后选择配置。

   如果您不 希望使用启用的管道，请为您的服务选择标记为不包含管道的模板。
4. 在配置服务页面上的服务设置部分中，输入一个服务名称。
5. （可选）输入服务的描述。
6. 在服务存储库设置部分中：

   a. 对于 CodeStar 连接，从列表中选择您的连接。

   b. 对于存储库 ID，从列表中选择您的源代码存储库的名称。

   c. 对于分支名称，从列表中选择您的源代码存储库分支的名称。
7. （可选）在标签部分中，选择添加新标签，并输入键和值以创建一个客户托管标签。
8. 选择下一步。
9. 在配置自定义设置页面上的服务实例部分中，在新实例部分中，您必须输入 `required` 参数的值。您可以输入 `optional` 参数的值，或使用给定的默认值。
10. 在管道输入部分中，您必须输入 `required` 参数的值。您可以输入 `optional` 参数的值，或使用给定的默认值。
11. 选择下一步并检查您的输入。
12. 选择创建。

    查看服务详细信息和状态，以及您的服务的 AWS 托管标签和客户托管标签。
13. 在导航窗格中，选择服务。

    新页面将显示您的服务列表以及状态和其他服务详细信息。

AWS CLI

在使用 AWS CLI 时，您可以在位于源代码目录的 YAML 格式 `spec` 文件 (`.aws-proton/service.yaml`) 中指定服务输入。

您可以使用 CLI `get-service-template-minor-version` 命令，查看您在规范文件中提供值的架构必需参数和可选参数。

如果要使用具有 `pipelineProvisioning: "CUSTOMER_MANAGED"` 的服务模板，请不要 在规范中包含 `pipeline:` 部分，并且不要 在 `create-service` 命令中包含 `-repository-connection-arn`、`-repository-id` 和 `-branch-name` 参数。

创建一个具有服务管道的服务，如以下 CLI 步骤中所示。

1. 为管道设置服务角色，如以下 CLI 示例命令中所示。

   命令:

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

2. 以下列表显示基于服务模板架构的示例规范，其中包括服务管道和实例输入。

   规范：

   ```
   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"
   ```

   创建一个具有管道的服务，如以下 CLI 示例命令和响应中所示。

   命令:

   ```
   $ aws proton create-service \
           --name "MySimpleService" \
           --branch-name "mainline" \
           --template-major-version "1" \
   ```

```
        --template-name "fargate-service" \
        --repository-connection-arn "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111" \
        --repository-id "myorg/myapp" \
        --spec "file://spec.yaml"
```

响应：

```
{
    "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"
    }
}
```

创建一个没有服务管道的服务，如以下 CLI 示例命令和响应中所示。

下面显示了一个不 包含服务管道输入的示例规范。

规范：

```
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"
```

要创建一个没有 预置的服务管道的服务，您需要提供 **spec.yaml** 的路径，并且不 包含存储库参数，如以下 CLI 示例命令和响应中所示。

命令:

```
$ aws proton create-service \
        --name "MySimpleServiceNoPipeline" \
        --template-major-version "1" \
        --template-name "fargate-service" \
        --spec "file://spec-no-pipeline.yaml"
```

响应：

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/
MySimpleServiceNoPipeline",
        "createdAt": "2020-11-18T19:50:27.460000+00:00",
        "lastModifiedAt": "2020-11-18T19:50:27.460000+00:00",
        "name": "MySimpleServiceNoPipeline",
        "status": "CREATE_IN_PROGRESS",
        "templateName": "fargate-service-no-pipeline"
    }
}
```

查看服务数据

您可以使用 AWS Proton 控制台或 AWS CLI 查看和列出服务详细数据。

AWS Management Console

使用 AWS Proton 控制台列出和查看服务详细信息，如以下步骤中所示。

1. 要查看服务列表，请在导航窗格中选择服务。
2. 要查看详细数据，请选择一个服务的名称。

   查看您的服务详细数据。

AWS CLI

查看具有服务管道的服务的详细信息，如以下 CLI 示例命令和响应中所示。

命令:

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

```
    --name "simple-svc"
```

响应：

```
{
    "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"
    }
}
```

查看没有服务管道的服务的详细信息，如以下 CLI 示例命令和响应中所示。

命令:

```
$ aws proton get-service \
    --name "simple-svc-no-pipeline"
```

响应：

```
{
    "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"
    }
}
```

编辑服务

您可以对 AWS Proton 服务进行以下编辑。

- 编辑服务描述。
- 添加和删除服务实例以编辑服务。

编辑服务描述

您可以使用控制台或 AWS CLI 编辑服务描述。

AWS Management Console

使用控制台编辑服务，如以下步骤中所述。

在服务列表中。

1. 在 AWS Proton 控制台中，选择服务。
2. 在服务列表中，选择要更新的服务左侧的单选按钮。

3. 选择编辑。
4. 在配置服务页面中，填写表单并选择下一步。
5. 在配置自定义设置页面中，选择下一步。
6. 检查您的编辑内容并选择保存更改。

在服务详细信息页面中。

1. 在 AWS Proton 控制台中，选择服务。
2. 在服务列表中，选择要编辑的服务的名称。
3. 在服务详细信息页面中，选择编辑。
4. 在配置服务页面中，填写表单并选择下一步。
5. 在配置自定义设置页面中，填写表单并选择下一步。
6. 检查您的编辑内容并选择保存更改。

AWS CLI

编辑描述，如以下 CLI 示例命令和响应中所示。

命令:

```
$ aws proton update-service \
    --name "MySimpleService" \
    --description "Edit by updating description"
```

响应：

```
{
    "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"
    }
}
```

编辑服务以添加或删除服务实例

对于 AWS Proton 服务，您可以提交编辑的规范以添加或删除服务实例。必须满足以下条件才能成功完成请求：

- 在您提交编辑请求时，尚未编辑或删除您的服务和管道。
- 您编辑的规范不包括修改服务管道的编辑或对不会 删除的现有服务实例的编辑。
- 您编辑的规范不会删除任何具有附加组件的现有服务实例。要删除此类服务实例，您应该先更新组件以将其与服务实例分离。有关组件的更多信息，请参阅组件。

删除失败的实例是状态为 DELETE_FAILED 的服务实例。在您请求服务编辑时，AWS Proton 在编辑过程中尝试将删除失败的实例删除。如果无法删除您的任何服务实例，则可能仍然存在与这些实例关联的资源，即使这些资源在控制台或 AWS CLI 中不可见。检查删除失败的实例基础设施资源，并清理这些资源，以便 AWS Proton 可以删除它们。

有关服务的服务实例配额，请参阅AWS Proton 配额。在创建服务后，您还必须为其保留至少 1 个服务实例。在更新过程中，AWS Proton 对现有服务实例和要添加或删除的实例进行计数。删除失败的实例包括在该计数中，您在编辑 spec 时必须考虑到这些实例。

使用控制台或 AWS CLI 添加或删除服务实例

AWS Management Console

使用控制台编辑您的服务以添加或删除服务实例。

在 AWS Proton 控制台中

1. 在导航窗格中，选择服务。
2. 选择您要编辑的服务。
3. 选择编辑。
4. （可选）在配置服务页面上，编辑服务名称或描述，然后选择下一步。
5. 在配置自定义设置页面上，选择删除以删除一个服务实例，然后选择添加新实例以添加一个服务实例并填写表单。

6. 选择下一步。
7. 检查您的更新内容并选择保存更改。
8. 一个模态框要求您确认删除服务实例。按照说明进行操作并选择是，删除。
9. 在服务详细信息页面上，查看您的服务的状态详细信息。

AWS CLI

使用编辑的 **spec** 添加和删除服务实例，如以下 AWS CLI 示例命令和响应中所示。

在您使用 CLI 时，您的 spec 必须排除 要删除的服务实例，并包括 要添加的服务实例以及尚未 标记为删除的现有服务实例。

以下列表显示编辑之前的示例 spec 以及规范部署的服务实例列表。在前面的示例中使用了以下规范以编辑服务描述。

规范：

```
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"
```

以下示例 CLI list-service-instances 命令和响应显示添加或删除服务实例之前的活动实例。

命令:

```
$ aws proton list-service-instances \
    --service-name "MySimpleService"
```

响应：

```
{
    "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"
        }
    ]
}
```

以下列表显示用于删除和添加实例的编辑的 spec 示例。删除了名为 my-instance 的现有实例，并添加了名为 yet-another-instance 的新实例。

规范：

```
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"
```

您可以使用 "${Proton::CURRENT_VAL}" 指示要在原始 spec 中保留哪些参数值（如果这些值在 spec 中存在）。可以使用 get-service 查看服务的原始 spec，如查看服务数据中所述。

以下列表说明了如何使用 "${Proton::CURRENT_VAL}" 确保 spec 不 包含要保留的现有服务实例的参数值更改。

规范：

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "yet-another-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

下一个列表显示用于编辑服务的 CLI 命令和响应。

命令:

```
$ aws proton update-service
    --name "MySimpleService" \
    --description "Edit by adding and deleting a service instance" \
    --spec "file://spec.yaml"
```

响应：

```
{
    "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"
    }
}
```

以下 `list-service-instances` 命令和响应确认删除了名为 `my-instance` 的现有实例，并添加了名为 `yet-another-instance` 的新实例。

命令:

```
$ aws proton list-service-instances \
    --service-name "MySimpleService"
```

响应：

```
{
    "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"
        }
    ]
}
```

在添加或删除服务实例时发生的情况

在您提交服务编辑以删除和添加服务实例后，AWS Proton 执行以下操作。

- 将服务设置为 UPDATE_IN_PROGRESS。
- 如果服务具有管道，则将其状态设置为 IN_PROGRESS 并阻止管道操作。
- 将要删除的任何服务实例设置为 DELETE_IN_PROGRESS。
- 阻止服务操作。
- 阻止对标记为删除的服务实例执行操作。
- 创建新的服务实例。
- 删除您列出的要删除的实例。
- 尝试将删除失败的实例删除。
- 在添加和删除完成后，重新预置服务管道（如果有），将您的服务设置为 ACTIVE 并启用服务和管道操作。

AWS Proton 尝试修复故障模式，如下所示。

- 如果一个或多个服务实例创建失败，AWS Proton 尝试取消预置所有新创建的服务实例，并将 spec 恢复到以前的状态。它不会 删除任何服务实例，也不会 以任何方式修改管道。
- 如果一个或多个服务实例删除失败，AWS Proton 重新预置管道而不包含删除的实例。将更新 spec 以包含添加的实例并排除标记为删除的实例。
- 如果管道预置失败，则不会尝试回滚，并且服务和管道反映失败的更新状态。

标记和服务编辑

如果您在服务编辑期间添加服务实例，AWS 托管标签将传播到新实例和预置的资源，并自动为它们创建该标签。如果您创建新标签，这些标签仅应用于新实例。现有服务客户托管标签也会传播到新实例。有关更多信息，请参阅AWS Proton 资源和标记。

删除服务

您可以使用 AWS Proton 控制台或 AWS CLI 删除 AWS Proton 服务及其实例和管道。

如果服务的任何服务实例具有附加组件，则无法删除该服务。要删除此类服务，您应该先更新所有附加的组件，以将其与服务实例分离。有关组件的更多信息，请参阅组件。

AWS Management Console

使用控制台删除服务，如以下步骤中所述。

在服务详细信息页面中。

1. 在 AWS Proton 控制台中，选择服务。
2. 在服务列表中，选择要删除的服务的名称。
3. 在服务详细信息页面上，选择操作，然后选择删除。
4. 一个模态框提示您确认删除操作。
5. 按照说明进行操作并选择是，删除。

AWS CLI

删除服务，如以下 CLI 示例命令和响应中所示。

命令:

```
$ aws proton delete-service \
    --name "simple-svc"
```

响应：

```
{
    "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"
    }
}
```

查看服务实例数据

了解如何查看 AWS Proton 服务实例详细数据。您可以使用控制台或 AWS CLI。

服务实例属于服务。您只能在服务编辑、创建和删除操作的上下文中创建或删除实例。要了解如何在服务中添加和删除实例，请参阅编辑服务。

AWS Management Console

使用 AWS Proton 控制台列出和查看服务实例详细信息，如以下步骤中所示。

1. 要查看服务实例列表，请在导航窗格中选择服务实例。
2. 要查看详细数据，请选择一个服务实例的名称。

   查看您的服务实例详细数据。

AWS CLI

列出和查看服务实例详细信息，如以下 CLI 示例命令和响应中所示。

命令:

```
$ aws proton list-service-instances
```

响应：

```
{
    "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"
        }
    ]
}
```

命令:

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

响应：

```
{
    "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"
    }
}
```

更新服务实例

了解如何更新 AWS Proton 服务实例和取消更新。

服务实例属于服务。您只能在服务编辑、创建和删除操作的上下文中创建或删除实例。要了解如何在服务中添加和删除实例，请参阅编辑服务。

可以使用 4 种模式更新服务实例，如以下列表中所述。在使用 AWS CLI 时，deployment-type 字段定义模式。在使用控制台时，这些模式映射到服务实例详细信息页面上的操作下拉列表中的编辑和更新到最新的次要版本以及更新到最新的主要版本操作。

NONE

在该模式下，不会 进行部署。仅更新请求的元数据参数。

CURRENT_VERSION

在该模式下，将使用您提供的新规范部署和更新服务实例。仅更新请求的参数。在使用该 deployment-type 时，不要 包含次要或主要版本参数。

MINOR_VERSION

在该模式下，默认使用当前使用的主要版本的已发布推荐（最新）次要版本部署和更新服务实例。您也可以指定当前使用的主要版本的不同次要版本。

MAJOR_VERSION

在该模式下，默认使用当前模板的已发布推荐（最新）主要版本和次要版本部署和更新服务实例。您也可以指定高于正在使用的主要版本的不同主要版本和次要版本（可选）。

如果 deploymentStatus 为 IN_PROGRESS，您可以尝试取消服务实例更新部署。AWS Proton 将尝试取消部署。不能 保证成功取消。

在您取消更新部署时，AWS Proton 尝试取消部署，如以下步骤中列出的一样。

- 将部署状态设置为 CANCELLING。
- 停止正在进行的部署，并删除状态为 IN_PROGRESS 时部署创建的任何新资源。
- 将部署状态设置为 CANCELLED。
- 将资源状态恢复为开始部署之前的状态。

有关取消服务实例部署的更多信息，请参阅 AWS Proton API Reference 中的 CancelServiceInstanceDeployment。

使用控制台或 AWS CLI 进行更新或取消更新部署。

AWS Management Console

按照以下步骤，使用控制台更新服务实例。

1. 在 AWS Proton 控制台中，在导航窗格中选择服务实例。
2. 在服务实例列表中，选择要更新的服务实例的名称。
3. 选择操作，然后选择更新选项之一：编辑（用于更新规范）、更新到最新的次要版本或更新到最新的主要版本。
4. 填写每个表单并选择下一步，直至到达审核页面。
5. 检查您的编辑内容并选择更新。

AWS CLI

将服务实例更新为新的次要版本，如 CLI 示例命令和响应中所示。

在您使用修改的 spec 更新服务实例时，您可以使用 "${Proton::CURRENT_VAL}" 指示要在原始 spec 中保留哪些参数值（如果这些值在 spec 中存在）。可以使用 get-service 查看服务实例的原始 spec，如查看服务数据中所述。

以下示例说明了如何在 spec 中使用 "${Proton::CURRENT_VAL}"。

规范：

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "${Proton::CURRENT_VAL}"
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

命令：更新

```
$ 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"
```

响应：

```
{
    "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"
    }
}
```

命令：获取并确认状态

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

响应：

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:38:00.823000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def\"\n
      my_sample_service_instance_required_input: \"456\"\n  - name: \"my-
other-instance\"\n    environment: \"kls-simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

AWS Management Console

使用控制台取消服务实例部署，如以下步骤中所示。

1. 在 AWS Proton 控制台中，在导航窗格中选择服务实例。
2. 在服务实例列表中，选择包含要取消的部署更新的服务实例的名称。
3. 如果您的更新部署状态为进行中，请在服务实例详细信息页面中选择操作，然后选择取消部署。
4. 一个模态框要求您确认取消。选择取消部署。
5. 您的更新部署状态设置为正在取消，然后设置为已取消以完成取消。

AWS CLI

取消将 IN_PROGRESS 服务实例部署更新为新的次要版本 2，如以下 CLI 示例命令和响应中所示。

在用于该示例的模板中包含一个等待条件，以便在更新部署成功之前开始取消。

命令：取消

```
$ aws proton cancel-service-instance-deployment \
    --service-instance-name "instance-one" \
    --service-name "simple-svc"
```

响应：

```
{
    "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"
    }
}
```

命令：获取并确认状态

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

响应：

```
{
    "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"
    }
}
```

更新服务管道

了解如何更新 AWS Proton 服务管道和取消更新。

服务管道属于服务。您只能在服务创建 和删除 操作的上下文中创建或删除管道。

可以使用 4 种模式更新服务管道，如以下列表中所述。在使用 AWS CLI 时，deployment-type 字段定义模式。在您使用控制台时，这些模式映射到编辑管道和更新到推荐版本。

NONE

在该模式下，不会 进行部署。仅更新请求的元数据参数。

CURRENT_VERSION

在该模式下，将使用您提供的新规范部署和更新服务管道。仅更新请求的参数。在使用该 deployment-type 时，不要 包含次要或主要版本参数。

MINOR_VERSION

在该模式下，默认使用当前使用的主要版本的已发布推荐（最新）次要版本部署和更新服务管道。您也可以指定当前使用的主要版本的不同次要版本。

MAJOR_VERSION

在该模式下，默认使用当前模板的已发布推荐（最新）主要版本和次要版本部署和更新服务管道。您也可以指定高于正在使用的主要版本的不同主要版本和次要版本（可选）。

如果 deploymentStatus 为 IN_PROGRESS，您可以尝试取消服务管道更新部署。AWS Proton 将尝试取消部署。不能保证成功取消。

在您取消更新部署时，AWS Proton 尝试取消部署，如以下步骤中列出的一样。

- 将部署状态设置为 CANCELLING。
- 停止正在进行的部署，并删除状态为 IN_PROGRESS 时部署创建的任何新资源。
- 将部署状态设置为 CANCELLED。
- 将资源状态恢复为开始部署之前的状态。

有关取消服务管道部署的更多信息，请参阅 AWS Proton API Reference 中的 CancelServicePipelineDeployment。

使用控制台或 AWS CLI 进行更新或取消更新部署。

AWS Management Console

使用控制台更新服务管道，如以下步骤中所述。

1. 在 AWS Proton 控制台中，选择服务。
2. 在服务列表中，选择要更新管道的服务的名称。
3. 在服务详细信息页面上具有两个选项卡：概述和管道。选择管道。
4. 如果要更新规范，请选择编辑管道并填写每个表单，然后选择下一步，直到填写最终表单，然后选择更新管道。

   如果要更新为新版本，并且在管道模板上具有指示有新版本的信息图标，请选择新模板版本的名称。

   a. 选择更新到推荐版本。
   b. 填写每个表单并选择下一步，直到填写最终表单并选择更新。

AWS CLI

将服务管道更新为新的次要版本，如以下 CLI 示例命令和响应中所示。

在您使用修改的 `spec` 更新服务管道时，您可以使用 `"${Proton::CURRENT_VAL}"` 指示要在原始 `spec` 中保留哪些参数值（如果这些值在 `spec` 中存在）。可以使用 `get-service` 查看服务管道的原始 `spec`，如查看服务数据中所述。

以下示例说明了如何在 `spec` 中使用 `"${Proton::CURRENT_VAL}"`。

规范：

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-instance"
    environment: "simple-env"
    spec:
```

```
      my_sample_service_instance_optional_input: "${Proton::CURRENT_VAL}"
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

命令：更新

```
$ aws proton update-service-pipeline \
    --service-name "simple-svc" \
    --spec "file://service-spec.yaml" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --deployment-type "MINOR_VERSION"
```

响应：

```
{
    "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"
    }
}
```

命令：获取并确认状态

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

```
    --name "simple-svc"
```

响应：

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "main",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "lastModifiedAt": "2021-04-02T21:30:54.364000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline",
            "createdAt": "2021-04-02T21:29:59.962000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "lastDeploymentAttemptedAt": "2021-04-02T21:39:28.991000+00:00",
            "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
            "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "repo-name/myorg-myapp",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

AWS Management Console

使用控制台取消服务管道部署，如以下步骤中所示。

1. 在 AWS Proton 控制台中，在导航窗格中选择服务。
2. 在服务列表中，选择管道包含要取消的部署更新的服务的名称。
3. 在服务详细信息页面中，选择管道选项卡。
4. 如果您的更新部署状态为进行中，请在服务管道详细信息页面中选择取消部署。
5. 一个模态框要求您确认取消。选择取消部署。
6. 您的更新部署状态设置为正在取消，然后设置为已取消以完成取消。

AWS CLI

取消将 IN_PROGRESS 服务管道部署更新为次要版本 2，如以下 CLI 示例命令和响应中所示。

在用于该示例的模板中包含一个等待条件，以便在更新部署成功之前开始取消。

命令：取消

```
$ aws proton cancel-service-pipeline-deployment \
    --service-name "simple-svc"
```

响应：

```
{
    "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"
    }
}
```

命令：获取并确认状态

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

响应：

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "main",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "lastModifiedAt": "2021-04-02T21:30:54.364000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline",
            "createdAt": "2021-04-02T21:29:59.962000+00:00",
            "deploymentStatus": "CANCELLED",
            "deploymentStatusMessage": "User initiated cancellation.",
            "lastDeploymentAttemptedAt": "2021-04-02T22:02:45.095000+00:00",
            "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
            "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "repo-name/myorg-myapp",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
```

```
    }
}
```

AWS Proton 组件

组件是一种类型的 AWS Proton 资源。它们提高了服务模板的灵活性。组件为平台团队提供了一种扩展核心基础设施模式的机制，并定义了保护措施，以使开发人员能够管理其应用程序基础设施的各个方面。

在 AWS Proton 中，管理员定义了在开发团队和应用程序中使用的标准基础设施。不过，开发团队可能需要为其特定使用案例添加额外的资源，例如 Amazon Simple Queue Service (Amazon SQS) 队列或 Amazon DynamoDB 表。这些应用程序特定的资源可能经常发生变化，特别是在早期应用程序开发阶段。由于需要在管理员编写的模板中处理这些经常发生的变化，因而可能难以管理和扩展 - 管理员需要维护更多的模板，而没有为管理员提供实际的附加价值。替代方案（让应用程序开发人员为其应用程序编写模板）也不理想，因为这导致管理员无法标准化主要架构组件（例如 AWS Fargate 任务）。这就是组件发挥作用的地方。

除了管理员在环境和服务模板中定义的资源以外，开发人员可以使用组件在其应用程序中添加补充资源。然后，开发人员将组件附加到服务实例。AWS Proton 预置组件定义的基础设施资源，就像为环境和服务实例预置资源一样。

组件可以读取服务实例输入，并向服务实例提供输出，从而提供完全集成的体验。例如，如果组件添加一个 Amazon Simple Storage Service (Amazon S3) 存储桶以供服务实例使用，则组件模板可以在命名该存储桶时考虑环境和服务实例名称。在 AWS Proton 渲染服务模板以预置服务实例时，服务实例可以引用并使用该存储桶。

AWS Proton 当前支持的组件是直接定义的组件。您将定义组件基础设施的基础设施即代码 (IaC) 文件直接传送到 AWS Proton API 或控制台。这与环境或服务不同，在环境或服务中，您在模板捆绑包中定义 IaC 并将该捆绑包注册为模板资源，然后使用模板资源创建环境或服务。

> **Note**
>
> 通过使用直接定义的组件，开发人员可以定义额外的基础设施并对其进行预置。AWS Proton 预置使用相同 AWS Identity and Access Management (IAM) 角色在同一环境中运行的所有直接定义的组件。

管理员可以通过两种方式控制开发人员使用组件执行的操作：

- 支持的组件源 - 管理员可以允许根据 AWS Proton 服务模板版本属性将组件附加到服务实例。默认情况下，开发人员无法将组件附加到服务实例。

有关该属性的更多信息，请参阅 AWS Proton API Reference 中的 CreateServiceTemplateVersion API 操作的 supportedComponentSources 参数。

> **Note**
>
> 在您使用模板同步时，在您提交对存储库中的服务模板捆绑包的更改时，AWS Proton 隐式创建服务模板版本。在这种情况下，您无需在服务模板版本创建期间指定支持的组件源，而是在与每个服务模板主要版本关联的文件中指定该属性。有关更多信息，请参阅the section called “同步服务模板”。

- 组件角色 - 管理员可以为环境分配组件角色。在环境中预置由直接定义的组件定义的基础设施时，AWS Proton 担任该角色。因此，组件角色缩小了开发人员可以使用直接定义的组件在环境中添加的基础设施范围。在没有组件角色的情况下，开发人员无法在环境中创建直接定义的组件。

  有关分配组件角色的更多信息，请参阅 AWS Proton API Reference 中的 CreateEnvironment API 操作的 componentRoleArn 参数。

  > **Note**
  >
  > 不会在自托管式预置环境中使用组件角色。

主题

- 组件与其他 AWS Proton 资源相比有什么区别？
- AWS Proton 控制台中的组件
- AWS Proton API 和 AWS CLI 中的组件
- 组件常见问题
- 组件状态
- 组件基础设施即代码文件
- 组件 AWS CloudFormation 示例

组件与其他 AWS Proton 资源相比有什么区别？

在很多方面，组件与其他 AWS Proton 资源相似。它们的基础设施是在 IaC 模板文件中定义的，并且是以 AWS CloudFormation YAML 或 Terraform HCL 格式编写的。AWS Proton 可以使用 AWS 托管式预置或自托管式预置以预置组件基础设施。

不过，组件在以下几个方面与其他 AWS Proton 资源不同：

- 已分离状态 - 组件旨在附加到服务实例并扩展其基础设施，但也可以处于已分离 状态，即，它们不附加到任何服务实例。有关组件状态的更多信息，请参阅the section called “组件状态”。
- 没有架构 - 组件不像模板捆绑包那样具有关联的架构。组件输入是由服务定义的。在组件附加到服务实例时，它可以使用输入。
- 没有客户托管组件 - AWS Proton 始终为您预置组件基础设施。没有自带资源 的组件版本。有关客户托管环境的更多信息，请参阅the section called “创建”。
- 没有模板资源 - 直接定义的组件没有类似于环境和服务模板的关联模板资源。您直接向组件提供 IaC 模板文件。同样，您可以直接提供一个清单，以定义用于预置组件基础设施的模板语言和渲染引擎。您可以采用与编写模板捆绑包类似的方式编写模板文件和清单。不过，对于直接定义的组件，不要求将 IaC 文件作为捆绑包存储在特定位置，并且您不会在 AWS Proton 中通过 IaC 文件创建模板资源。
- 没有基于 CodeBuild 的预置 - 您无法使用自己的自定义预置脚本（称为基于 CodeBuild 的预置）预置直接定义的组件。有关更多信息，请参阅the section called “CodeBuild 预置”。

AWS Proton 控制台中的组件

可以使用 AWS Proton 控制台创建、更新、查看和使用 AWS Proton 组件。

以下控制台页面与组件相关。我们包含指向顶级控制台页面的直接链接。

- 组件 - 查看您的 AWS 账户中的组件列表。您可以创建新的组件，以及更新或删除现有的组件。可以在列表中选择一个组件名称以查看其详细信息页面。

  在环境详细信息和服务实例详细信息页面上也包含类似的列表。这些列表仅显示与查看的资源关联的组件。在您从其中的一个列表中创建组件时，AWS Proton 在创建组件页面上预选择关联的环境。
- 组件详细信息 - 要查看组件详细信息页面，请在组件列表中选择一个组件名称。

  在详细信息页面上，查看组件详细信息和状态以及更新或删除组件。查看和管理输出列表（例如，预置的资源 ARN）、预置的 AWS CloudFormation 堆栈和分配的标签。

- 创建组件 - 创建组件。输入组件名称和描述，选择关联的资源，指定组件源 IaC 文件并分配标签。
- 更新组件 - 要更新组件，请在组件列表中选择该组件，然后在操作菜单上选择更新组件。或者，在组件详细信息页面上，选择更新。

  您可以更新大多数组件的详细信息。您无法更新组件名称。您可以选择在成功更新后是否重新部署组件。
- 配置环境 - 在创建或更新环境时，您可以指定组件角色。该角色控制能否在环境中运行直接定义的组件，并提供预置这些组件的权限。
- 创建新的服务模板版本 - 在创建服务模板版本时，您可以为该模板版本指定支持的组件源。这会控制是否能够将组件附加到基于该模板版本的服务的服务实例。

AWS Proton API 和 AWS CLI 中的组件

可以使用 AWS Proton API 或 AWS CLI 创建、更新、查看和使用 AWS Proton 组件。

以下 API 操作直接管理 AWS Proton 组件资源。

- CreateComponent - 创建 AWS Proton 组件。
- DeleteComponent - 删除 AWS Proton 组件。
- GetComponent - 获取组件的详细数据。
- ListComponentOutputs - 获取组件基础设施即代码 (IaC) 输出列表。
- ListComponentProvisionedResources - 列出组件的预置资源及其详细信息。
- ListComponents - 列出组件以及摘要数据。您可以按环境、服务或单个服务实例筛选结果列表。

其他 AWS Proton 资源的以下 API 操作具有一些与组件相关的功能。

- CreateEnvironment、UpdateEnvironment - 使用 `componentRoleArn` 指定 AWS Proton 在该环境中预置直接定义的组件时使用的 IAM 服务角色的 Amazon 资源名称 (ARN)。它决定了直接定义的组件可以预置的基础设施范围。
- CreateServiceTemplateVersion - 使用 `supportedComponentSources` 指定支持的组件源。具有支持的源的组件可以附加到基于该服务模板版本的服务实例。

组件常见问题

组件的生命周期是怎样的？

组件可以处于已附加 或已分离 状态。它们旨在附加到服务实例，并在大多数时候增强其基础设施。分离的组件处于过渡状态，您可以使用受控且安全的方式删除组件或将其附加到另一个服务实例。有关更多信息，请参阅the section called “组件状态”。

为什么我无法删除附加的组件？

解决方案：要删除附加的组件，请更新该组件以将其与服务实例分离，验证服务实例稳定性，然后删除该组件。

为什么需要这样做？ 附加的组件提供了额外的基础设施，您的应用程序在执行其运行时功能需要使用该基础设施。服务实例可以使用组件输出以检测和使用该基础设施的资源。如果删除组件并因而删除其基础设施资源，可能会影响附加的服务实例。

作为一项额外的安全措施，AWS Proton 要求您更新组件并将其与服务实例分离，然后才能将其删除。然后，您可以验证您的服务实例，以确保它继续部署并正常工作。如果检测到问题，您可以快速将组件重新附加到服务实例，然后着手解决该问题。如果您确信服务实例不存在对该组件的任何依赖性，则可以安全地删除该组件。

为什么我不能直接更改组件附加的服务实例？

解决方案：要更改附加的服务实例，请更新组件以将其与服务实例分离，验证组件和服务实例稳定性，然后将组件附加到新的服务实例。

为什么需要这样做？ 组件旨在附加到服务实例。您的组件可以使用服务实例输入进行基础设施资源命名和配置。更改附加的服务实例可能会影响组件；还可能会影响服务实例，如前面的常见问题为什么我无法删除附加的组件？中所述）。例如，这可能会导致重命名组件的 IaC 模板中定义的资源，甚至可能替换这些资源。

作为一项额外的安全措施，AWS Proton 要求您更新组件并将其与服务实例分离，然后才能将其附加到另一个服务实例。接下来，您可以验证组件和服务实例的稳定性，然后再将组件附加到新服务实例。

组件状态

AWS Proton 组件可以处于两种截然不同的状态：

- 已附加 - 组件附加到服务实例。它定义了支持服务实例的运行时功能的基础设施。组件使用开发人员定义的基础设施扩展环境和服务模板中定义的基础设施。

  典型组件在其生命周期的大多数时间处于已附加状态。

- 已分离 - 组件与 AWS Proton 环境关联，并且未附加到环境中的任何服务实例。

  这是一种过渡状态，用于将组件生命周期延长到单个服务实例以外。

下表简要比较了不同的组件状态。

	已附加	已分离
状态的主要用途	扩展服务实例的基础设施。	在附加服务实例之间保留组件的基础设施。
关联对象	服务实例和环境	环境
关键具体属性	• 服务名称 • 服务实例名称 • 规范	• 环境名称
可以删除	✗ 否	✓ 是
可以更新为另一个服务实例	✗ 否	✓ 是
可以读取输入	✓ 是	✗ 否

组件的主要用途是附加到服务实例，并使用额外的资源扩展其基础设施。附加的组件可以根据规范从服务实例中读取输入。您无法直接删除组件，或将其附加到不同的服务实例。您也无法删除其服务实例或相关的服务和环境。要执行任何上述操作，请先更新组件以将其与服务实例分离。

要在单个服务实例的生命周期以外保留组件的基础设施，您可以更新组件，并删除服务和服务实例名称以将其与服务实例分离。这种已分离状态是一种过渡状态。组件没有输入。其基础设施保持已预置状态，您可以对其进行更新。您可以删除附加组件时与其关联的资源（服务实例、服务）。您可以删除组件，或更新组件以再次附加到服务实例。

组件基础设施即代码文件

组件基础设施即代码 (IaC) 文件与其他 AWS Proton 资源的 IaC 文件类似。可以在此处了解一些组件特定的详细信息。有关为 AWS Proton 编写 IaC 文件的完整信息，请参阅模板编写和捆绑包。

将参数与组件一起使用

AWS Proton 参数命名空间包含一些参数，服务 IaC 文件可以引用这些参数以获取关联组件的名称和输出。命名空间还包含一些参数，组件 IaC 文件可以引用这些参数以从与组件关联的环境、服务和服务实例中获取输入、输出和资源值。

组件没有自己的输入，它从附加到的服务实例中获取输入。组件也可以读取环境输出。

有关在组件和关联的服务 IaC 文件中使用参数的更多信息，请参阅the section called “组件 CloudFormation IaC 参数”。有关 AWS Proton 参数的一般信息以及参数命名空间的完整参考，请参阅the section called “参数”。

编写强大的 IaC 文件

作为管理员，在创建服务模板版本时，您可以决定是否要允许通过模板版本创建的服务实例具有附加的组件。请参阅 AWS Proton API Reference 中的 CreateServiceTemplateVersion API 操作的 supportedComponentSources 参数。不过，对于任何将来的服务实例，创建实例、决定是否将组件附加到服务实例以及（对于直接定义的组件）编写组件 IaC 的人通常是不同的人 - 使用您的服务模板的开发人员。因此，您无法保证组件将附加到服务实例。您也无法保证特定组件输出名称是否存在，或者这些输出的值是否有效和安全。

AWS Proton 和 Jinja 语法可以帮助您解决这些问题并编写强大的服务模板，这些模板可以通过以下方式渲染而不会失败：

- AWS Proton 参数筛选条件 - 在引用组件输出属性时，您可以使用参数筛选条件 - 验证和筛选参数值以及设置参数值格式的修饰符。有关更多信息以及示例，请参阅 the section called “CloudFormation 参数过滤器”。
- 单个属性默认值 - 在引用组件的单个资源或输出属性时，您可以使用 `default` 筛选条件（具有或没有默认值）保证服务模板渲染不会失败。如果您引用的组件或特定输出参数不存在，则渲染默认值或空字符串（如果未指定默认值），并且渲染成功。有关更多信息，请参阅the section called “提供默认值”。

  示例：

  - `{{ service_instance.components.default.name | default("") }}`
  - `{{ service_instance.components.default.outputs.my-output | default("17") }}`

> **Note**
>
> 不要将命名空间的 `.default` 部分（指定直接定义的组件）与 `default` 筛选条件混淆，该筛选条件在引用的属性不存在时提供默认值。

- 整个对象引用 - 在您引用整个组件或组件的输出集合时，AWS Proton 返回一个空对象 ({})，因此保证渲染服务模板不会失败。您不必使用任何筛选条件。请务必在可以使用空对象的上下文中进行引用，或使用 `{{ if .. }}` 条件测试空对象。

  示例：

  - `{{ service_instance.components.default }}`
  - `{{ service_instance.components.default.outputs }}`

组件 AWS CloudFormation 示例

以下是 AWS Proton 直接定义的组件以及如何在 AWS Proton 服务中使用该组件的完整示例。该组件预置一个 Amazon Simple Storage Service (Amazon S3) 存储桶和相关的访问策略。服务实例可以引用和使用该存储桶。存储桶名称基于环境、服务、服务实例和组件的名称，这意味着存储桶与扩展特定服务实例的特定组件模板实例结合使用。开发人员可以根据该组件模板创建多个组件，以针对不同的服务实例和功能需求预置 Amazon S3 存储桶。

该示例涵盖了编写各种所需的 AWS CloudFormation 基础设施即代码 (IaC) 文件以及创建所需的 AWS Identity and Access Management (IAM) 角色。该示例按拥有人角色对步骤进行分组。

管理员步骤

允许开发人员将组件与服务一起使用

1. 创建一个 AWS Identity and Access Management (IAM) 角色，以缩小在您的环境中运行的直接定义组件可以预置的资源范围。以后，AWS Proton 担任该角色以在环境中预置直接定义的组件。

   对于该示例，请使用以下策略：

   Example 直接定义的组件角色

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

```
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeStacks",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStackEvents",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:DeleteBucket",
        "s3:GetBucket",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:GetPolicy",
        "iam:ListPolicyVersions",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": "cloudformation.amazonaws.com"
        }
      }
    }
  ]
}
```

2. 在创建或更新环境时，提供您在上一步中创建的角色。在 AWS Proton 控制台中，在配置环境页面上指定一个组件角色。如果您使用 AWS Proton API 或 AWS CLI，请指定 CreateEnvironment 或 UpdateEnvironment API 操作的 `componentRoleArn`。
3. 创建一个服务模板，以引用附加到服务实例的直接定义组件。

   该示例说明了如何编写一个强大的服务模板，在组件未附加到服务实例时，该模板不会发生中断。

   Example 使用组件的服务 CloudFormation IaC 文件

```
# service/instance_infrastructure/cloudformation.yaml

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

  # ...

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

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

4. 创建一个新的服务模板次要版本，以将直接定义的组件声明为支持的组件。

- Amazon S3 中的模板捆绑包 - 在 AWS Proton 控制台中，在您创建服务模板版本时，为支持的组件源选择直接定义。如果您使用 AWS Proton API 或 AWS CLI，请在 CreateServiceTemplateVersion 或 UpdateServiceTemplateVersion API 操作的 `supportedComponentSources` 参数中指定 `DIRECTLY_DEFINED`。
- 模板同步 - 将更改提交到服务模板捆绑包存储库，其中，您将 `DIRECTLY_DEFINED` 指定为主要版本目录下面的 `.template-registration.yaml` 文件中的一个 `supported_component_sources:` 项目。有关此文件的更多信息，请参阅the section called “同步服务模板”。

5. 发布新的服务模板次要版本。有关更多信息，请参阅the section called “发布”。
6. 请务必在使用该服务模板的开发人员的 IAM 角色中允许 `proton:CreateComponent`。

开发人员步骤

将直接定义的组件与服务实例一起使用

1. 创建一个服务，以使用管理员通过组件支持创建的服务模板版本。或者，更新现有服务实例之一以使用最新的模板版本。
2. 编写一个组件 IaC 模板文件，该文件预置 Amazon S3 存储桶和相关访问策略，并将这些资源公开为输出。

   Example 组件 CloudFormation IaC 文件

```
# cloudformation.yaml

# A component that defines an S3 bucket and a policy for accessing the bucket.
Resources:
  S3Bucket:
    Type: 'AWS::S3::Bucket'
    Properties:
      BucketName: '{{environment.name}}-{{service.name}}-{{service_instance.name}}-
{{component.name}}'
  S3BucketAccessPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
```

```
            Action:
              - 's3:Get*'
              - 's3:List*'
              - 's3:PutObject'
            Resource: !GetAtt S3Bucket.Arn
Outputs:
  BucketName:
    Description: "Bucket to access"
    Value: !GetAtt S3Bucket.Arn
  BucketAccessPolicyArn:
    Value: !Ref S3BucketAccessPolicy
```

3. 如果您使用 AWS Proton API 或 AWS CLI，请为组件编写一个清单文件。

   Example 直接定义的组件清单

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

4. 创建直接定义的组件。AWS Proton 担任管理员定义的组件角色以预置组件。

   在 AWS Proton 控制台中的组件页面上，选择创建组件。对于组件设置，输入组件名称和可选的组件描述。对于组件附件，选择将组件附加到服务实例。选择您的环境、服务和服务实例。对于组件源，选择 AWS CloudFormation，然后选择组件 IaC 文件。

   > **Note**
   >
   > 您无需提供清单 - 控制台将为您创建清单。

   如果您使用 AWS Proton API 或 AWS CLI，请使用 CreateComponent API 操作。设置组件 `name` 和可选的 `description`。设置 `environmentName`、`serviceName` 和 `serviceInstanceName`。将 `templateSource` 和 `manifest` 设置为您创建的文件的路径。

> **Note**
>
> 在您指定服务和服务实例名称时，指定环境名称是可选的。这两者的组合在您的 AWS 账户中是唯一的，并且 AWS Proton 可以通过服务实例确定环境。

5. 更新您的服务实例以重新进行部署。AWS Proton 在渲染的服务实例模板中使用组件的输出，以使您的应用程序能够使用组件预置的 Amazon S3 存储桶。

在 AWS Proton 中使用 Git 存储库

AWS Proton 将 Git 存储库用于多种用途。以下列表对与 AWS Proton 资源关联的存储库类型进行了分类。对于反复连接到存储库以将内容推送到存储库或从中拉取内容的 AWS Proton 功能，您必须在 AWS Proton 中为您的 AWS 账户注册存储库链接。存储库链接是 AWS Proton 连接到存储库时可以使用的一组属性。AWS Proton 目前支持 GitHub、GitHub Enterprise 和 BitBucket。

开发人员存储库

代码存储库 - 开发人员用于存储应用程序代码的存储库。用于代码部署。AWS Proton 不直接与该存储库进行交互。在开发人员预置包含管道的服务时，他们提供存储库名称和分支以从中读取其应用程序代码。AWS Proton 将该信息传递给它预置的管道。

有关更多信息，请参阅the section called “创建”。

管理员存储库

模板存储库 - 管理员在其中存储 AWS Proton 模板捆绑包的存储库。用于模板同步。当管理员在 AWS Proton 中创建模板时，他们可以指向一个模板存储库，并且 AWS Proton 将新模板与该存储库保持同步。在管理员更新存储库中的模板捆绑包时，AWS Proton 自动创建新的模板版本。将一个模板存储库链接到 AWS Proton，然后才能使用该存储库进行同步。

有关更多信息，请参阅the section called “模板同步配置”。

> **Note**
>
> 如果您继续将模板上传到 Amazon Simple Storage Service (Amazon S3) 并调用 AWS Proton 模板管理 API 以创建新的模板或模板版本，则不需要使用模板存储库。

自托管式预置存储库

基础设施存储库 - 托管渲染的基础设施模板的存储库。用于资源基础设施 的自托管式预置。在管理员创建一个环境以进行自托管式预置时，他们提供一个存储库。AWS Proton 向该存储库提交拉取请求 (PR)，以便为该环境和部署到该环境的任何服务实例创建基础设施。将一个基础设施存储库链接到 AWS Proton，然后才能使用该存储库进行自托管式基础设施预置。

管道存储库 - 用于创建管道的存储库。用于管道 的自托管式预置。通过使用额外的存储库预置管道，AWS Proton 可以独立于任何单独的环境或服务存储管道配置。您只需为所有自托管式预置服

务提供一个管道存储库。将一个管道存储库链接到 AWS Proton，然后才能使用该存储库进行自托管式管道预置。

有关更多信息，请参阅the section called “AWS 托管式预置”。

主题

创建存储库的链接

您可以使用控制台或 CLI 创建存储库的链接。在您创建存储库链接时，AWS Proton 为您创建一个服务相关角色。

AWS Management Console

创建存储库的链接，如以下控制台步骤中所示。

1. 在 AWS Proton 控制台中，选择存储库。
2. 选择 Create repository（创建存储库）。
3. 在关联新存储库页面上的存储库详细信息部分中：
   a. 选择您的存储库提供商。
   b. 选择您的现有连接之一。如果没有，请选择添加新的 CodeStar 连接以创建一个连接，然后返回到 AWS Proton 控制台，刷新连接列表，然后选择您的新连接。
   c. 从您连接的源代码存储库中进行选择。
4. [可选] 在标签部分中，选择一次或多次添加新标签，然后输入键和值对。
5. 选择 Create repository（创建存储库）。
6. 查看您的链接存储库的详细数据。

AWS CLI

创建并注册您的存储库的链接。

运行以下命令：

```
$ aws proton create-repository \
    --name myrepos/environments \
    --connection-arn "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111" \
    --provider "GITHUB" \
    --encryption-key "arn:aws:kms:region-id:123456789012:key/bPxRfiCYEXAMPLEKEY" \
    --tags key=mytag1,value=value1 key=mytag2,value=value2
```

最后两个参数（--encryption-key 和 --tags）是可选的。

响应：

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments",
        "connectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/2ad03b28-a7c4-EXAMPLE11111",
        "encryptionKey": "arn:aws:kms:region-id:123456789012:key/
bPxRfiCYEXAMPLEKEY",
        "name": "myrepos/environments",
        "provider": "GITHUB"
    }
}
```

在创建存储库链接后，您可以查看 AWS 和客户托管标签列表，如以下示例命令中所示。AWS Proton 自动为您生成 AWS 托管标签。您也可以使用 AWS CLI 修改和创建客户托管标签。有关更多信息，请参阅AWS Proton 资源和标记。

命令:

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments"
```

查看链接的存储库数据

您可以使用控制台或 AWS CLI 列出和查看链接的存储库详细信息。对于用于将 Git 存储库与 AWS Proton 同步的存储库链接，您可以使用 AWS CLI 检索存储库同步定义和状态。

AWS Management Console

使用 AWS Proton 控制台列出和查看链接的存储库详细信息。

1. 要列出链接的存储库，请在导航窗格中选择存储库。
2. 要查看详细数据，请选择该存储库的名称。

AWS CLI

列出您的链接存储库。

运行以下命令：

```
$ aws proton list-repositories
```

响应：

```
{
    "repositories": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
            "name": "myrepos/templates",
            "provider": "GITHUB"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments",
            "name": "myrepos/environments",
            "provider": "GITHUB"
        }
    ]
}
```

查看链接的存储库的详细信息。

运行以下命令：

```
$ aws proton get-repository \
    --name myrepos/templates \
    --provider "GITHUB"
```

响应：

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
        "name": "myrepos/templates",
        "provider": "GITHUB"
    }
}
```

列出您的同步存储库。

以下示例列出您为模板同步配置的存储库。

运行以下命令：

```
$ aws proton list-repository-sync-definitions \
    --branch "main" \
    --repository-name myrepos/templates \
    --repository-provider "GITHUB" \
    --sync-type "TEMPLATE_SYNC"
```

查看存储库同步状态。

以下示例检索模板同步存储库的同步状态。

运行以下命令：

```
$ aws proton get-repository-sync-status \
    --branch "main" \
    --repository-name myrepos/templates \
    --repository-provider "GITHUB" \
    --sync-type "TEMPLATE_SYNC"
```

响应：

```
{
    "latestSync": {
        "events": [
            {
                "event": "Clone started",
```

```
                "time": "2021-11-21T00:26:35.883000+00:00",
                "type": "CLONE_STARTED"
            },
            {
                "event": "Updated configuration",
                "time": "2021-11-21T00:26:41.894000+00:00",
                "type": "CONFIG_UPDATED"
            },
            {
                "event": "Starting syncs for commit 62c03ff86eEXAMPLE1111111",
                "externalId": "62c03ff86eEXAMPLE1111111",
                "time": "2021-11-21T00:26:44.861000+00:00",
                "type": "STARTING_SYNC"
            }
        ],
        "startedAt": "2021-11-21T00:26:29.728000+00:00",
        "status": "SUCCEEDED"
    }
}
```

删除存储库链接

您可以使用控制台或 AWS CLI 删除存储库链接。

> **Note**
>
> 在删除存储库链接时，仅删除 AWS Proton 在您的 AWS 账户中注册的链接。它不会从您的存储库中删除任何信息。

AWS Management Console

使用控制台删除存储库链接。

在存储库详细信息页面中。

1. 在 AWS Proton 控制台中，选择存储库。
2. 在存储库列表中，选择要删除的存储库左侧的单选按钮。
3. 选择 Delete (删除)。
4. 一个模态框提示您确认删除操作。

5. 按照说明进行操作并选择是，删除。

AWS CLI

删除存储库链接。

运行以下命令：

```
$ aws proton delete-repository \
    --name myrepos/templates \
    --provider"GITHUB"
```

响应：

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
        "name": "myrepos/templates",
        "provider": "GITHUB"
    }
}
```

监控 AWS Proton

监控是维护 AWS 解决方案的可靠性、可用性和性能的重要组成部分。 AWS Proton 以下部分介绍可以与配合使用的监控工具 AWS Proton。

使用实现自动 AWS Proton 化 EventBridge

您可以在 Amazon 上监控 AWS Proton 事件 EventBridge。 EventBridge 提供来自您自己的应用程序、 software-as-a-service (SaaS) 应用程序和的实时数据流 AWS 服务。您可以配置事件以响应 AWS 资源状态的变化。 EventBridge 然后将这些数据路由到目标服务，例如 AWS Lambda 和 Amazon 简单通知服务。这些事件与 Amazon Events 中显示 CloudWatch 的事件相同。 CloudWatch Events 提供近乎实时的系统事件流，这些事件描述了 AWS 资源的变化。有关更多信息，请参阅什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中。

用于 EventBridge 接收 AWS Proton 配置工作流程中状态变化的通知。

事件类型

事件由包含事件模式和目标的规则组成。您可以选择事件模式和目标对象以配置规则：

事件模式

每个规则表示为一个事件模式，其中包含要监控的事件源和类型以及事件目标。要监控事件，您可以创建一个规则，并将您监控的服务作为事件源。例如，您可以创建一个具有事件模式的规则，该规则将 AWS Proton 作为事件源，以在部署状态发生变化时触发规则。

目标

该规则将选定的服务作为事件目标。您可以设置目标服务以发送通知，捕获状态信息，采取纠正措施，启动事件或采取其他措施。

事件对象包含 ID、帐户、详细信息类型 AWS 区域、来源、版本、资源、时间（可选）等标准字段。详细信息字段是一个嵌套对象，其中包含事件的自定义字段。

AWS Proton 事件是在尽力而为的基础上发出的。尽力交付意味着服务会尝试将所有事件发送到 EventBridge，但在极少数情况下，事件可能无法传送。

对于每种可以发出事件的 AWS Proton 资源，下表列出了详细信息类型值、详细信息字段以及（如果可用）`status`和`previousStatus`详细信息字段值列表的引用。在删除资源时，`status` 详细信息字段值为 `DELETED`。

资源	详细信息类型值	详细信息字段
EnvironmentTemplate	AWS Proton 环境模板状态更改	name status previousStatus
EnvironmentTemplateVersion	AWS Proton 环境模板版本状态更改	name majorVersion minorVersion status previousStatus 状态值
ServiceTemplate	AWS Proton 服务模板状态变更	name status previousStatus
ServiceTemplateVersion	AWS Proton 服务模板版本状态变更	name majorVersion minorVersion status

资源	详细信息类型值	详细信息字段
		previousStatus 状态值
Environment	AWS Proton 环境状态变更	name status previousStatus
Service	AWS Proton 服务状态变更	name status previousStatus 状态值
ServiceInstance	AWS Proton 服务实例状态更改	name serviceName status previousStatus
ServicePipeline	AWS Proton 服务管道状态变更	serviceName status previousStatus

资源	详细信息类型值	详细信息字段
EnvironmentAccountConnection	AWS Proton 环境账户连接状态变更	id status previousStatus 状态值
Component	AWS Proton 组件状态变更	name status previousStatus

AWS Proton 事件示例

以下示例显示了 AWS Proton 可以向发送事件的方式 EventBridge。

服务模板

```
{
    "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"
    }
}
```

服务模板版本

```
{
```

```
    "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"
    }
}
```

环境

```
{
    "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: 向亚马逊简单通知服务发送 AWS Proton 服务状态变更提醒

在本教程中，您将使用 AWS Proton 预先配置的事件规则来捕获 AWS Proton 服务的状态变化。EventBridge将状态更改发送到 Amazon SNS 主题。您订阅了该主题，Amazon SNS 就会向您发送服务状态变更电子邮件。 AWS Proton

先决条件

您有一个`Active`状态为的现有 AWS Proton 服务。作为本教程的一部分，您可以将服务实例添加到该服务中，然后删除这些实例。

如果您需要创建 AWS Proton 服务，请参阅开始使用。有关更多信息，请参阅 AWS Proton 配额 和 the section called “编辑”。

步骤 1：创建并订阅 Amazon SNS 主题

创建一个 Amazon SNS 主题以作为在步骤 2 中创建的事件规则 的事件目标。

创建 Amazon SNS 主题

1. 登录并打开 Amazon SNS 控制台。
2. 在导航窗格中，选择主题 > 创建主题。
3. 在创建主题页面中：
   a. 为类型选择标准。
   b. 对于名称，输入 **tutorial-service-status-change** 并选择创建主题。
4. 在tutorial-service-status-change详情页面中，选择创建订阅。
5. 在创建订阅页面中：
   a. 对于协议，选择电子邮件。
   b. 对于端点，输入您当前有权访问的电子邮件地址，然后选择 创建订阅。
6. 检查您的电子邮件账户，并等待接收订阅确认电子邮件。在收到该电子邮件后，将其打开并选择确认订阅。

步骤 2：注册事件规则

注册可捕获 AWS Proton 服务状态更改的事件规则。有关更多信息，请参阅先决条件。

创建一个事件规则。

1. 打开 Amazon EventBridge 控制台。
2. 在导航窗格中，依次选择 Events 和 Rules。
3. 在规则页面上的规则部分中，选择创建规则。
4. 在创建规则页面中：
   a. 在名称和描述部分中，为名称输入 **tutorial-rule**。
   b. 在定义模式部分中，选择事件模式。

i. 对于事件匹配模式，选择按服务预定义模式。

ii. 对于 Service provider（服务提供商），选择 AWS。

iii. 对于 Service name（服务名称），选择 AWS Proton。

iv. 对于事件类型，选择 AWS Proton 服务状态更改。

将在文本编辑器中显示事件模式。

v. 打开AWS Proton 控制台。

vi. 在导航窗格中，选择服务。

vii. 在服务页面中，选择您的 AWS Proton 服务名称。

viii. 在服务详细信息页面中，复制服务 Amazon 资源名称 (ARN)。

ix. 返回EventBridge 控制台和您的教程规则，然后在文本编辑器中选择 “编辑”。

x. 在文本编辑器中，为 `"resources":` 输入您在步骤 viii 中复制的服务 ARN。

```
{
    "source": ["aws.proton"],
    "detail-type": ["AWS Proton Service Status Change"],
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-service"]
}
```

xi. 保存事件模式。

c. 在选择目标部分中：

i. 对于 Target (目标)，选择 SNS topic (SNS 主题)。

ii. 对于 “主题”，选择tutorial-service-status-change。

d. 选择创建。

步骤 3：测试您的事件规则

向 AWS Proton 服务中添加实例，验证您的事件规则是否正常运行。

1. 切换到 AWS Proton 控制台。
2. 在导航窗格中，选择服务。
3. 在服务页面中，选择您的服务的名称。
4. 在服务详细信息页面中，选择编辑。

5. 在配置服务页面中，选择下一步。
6. 在配置自定义设置页面上的服务实例部分中，选择添加新实例。
7. 填写新实例的表单：
   a. 输入新实例的名称。
   b. 选择您为现有实例选择的相同兼容环境。
   c. 输入所需输入的值。
   d. 选择下一步。
8. 检查您的输入并选择更新。
9. 服务状态变为后Active，请查看您的电子邮件以确认您收到了提供状态更新的 AWS 通知。

```
{
    "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"
```

```
    }
}
```

步骤 4：清除

删除您的 Amazon SNS 主题和订阅，然后删除您的 EventBridge 规则。

删除您的 Amazon SNS 主题和订阅。

1. 导航到 Amazon SNS 控制台。
2. 在导航面板中，选择订阅。
3. 在订阅页面中，选择您对名为 `tutorial-service-status-change` 的主题进行的订阅，然后选择删除。
4. 在导航面板中，选择主题。
5. 在主题页面中，选择名为 `tutorial-service-status-change` 的主题，然后选择删除。
6. 一个模态框提示您确认删除。按照说明进行操作并选择删除。

删除您的 EventBridge 规则。

1. 导航至 Amazon EventBridge 控制台。
2. 在导航窗格中，依次选择 Events 和 Rules。
3. 在规则页面中，选择名为 `tutorial-rule` 的规则，然后选择删除。
4. 一个模态框提示您确认删除。选择 Delete（删除)。

删除添加的服务实例。

1. 导航到 AWS Proton 控制台。
2. 在导航窗格中，选择服务。
3. 在服务页面中，选择您的服务的名称。
4. 在服务详细信息页面中，选择编辑，然后选择下一步。
5. 在配置自定义设置页面上的服务实例部分中，为您在本教程中创建的服务实例选择删除，然后选择下一步。
6. 检查您的输入并选择更新。

7. 一个模态框提示您确认删除。按照说明进行操作并选择是，删除。

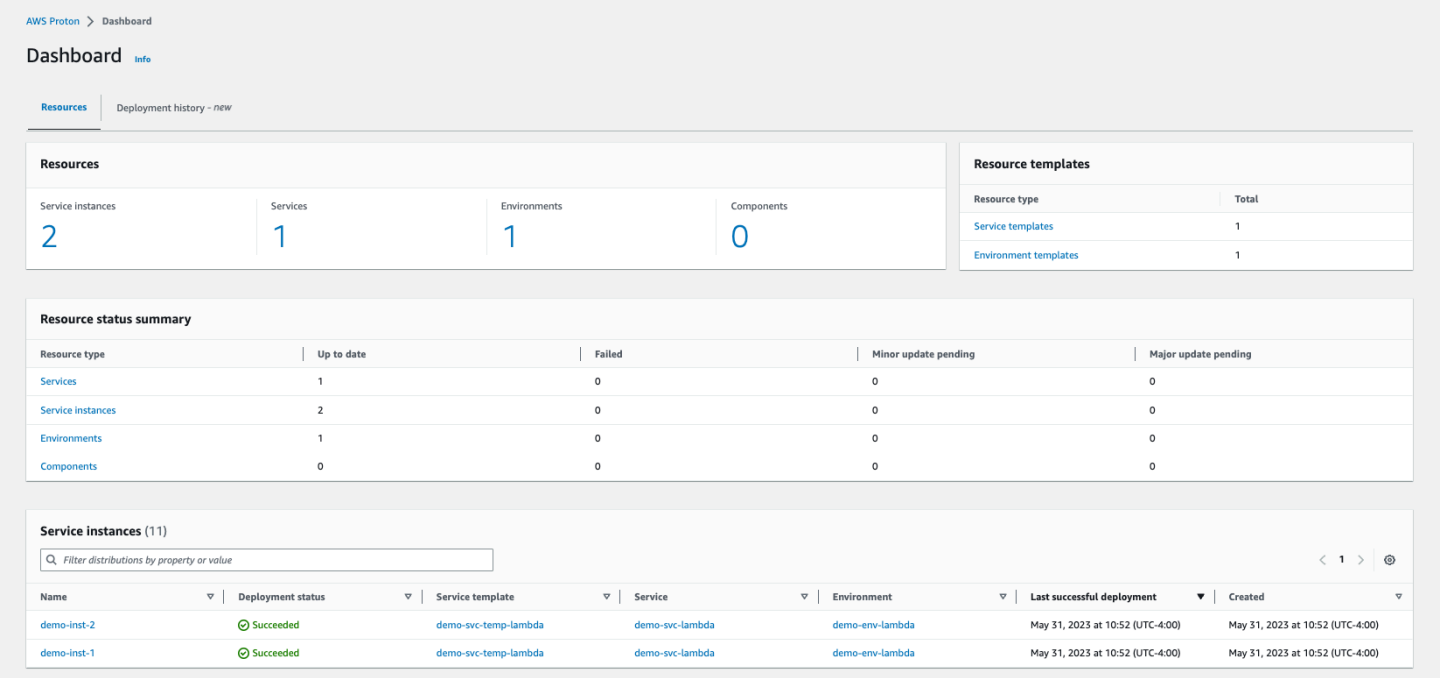
使用 AWS Proton 仪表板使基础架构保持最新状态

AWS Proton 控制面板提供您 AWS 账户中 AWS Proton 资源的摘要，特别关注过时情况，即已部署资源的更新情况。在部署的资源使用推荐的关联模板版本时，该资源就是最新的。 out-of-date 已部署的资源可能需要更新主要或次要模板版本。

在控制 AWS Proton 台中查看仪表板

要查看控制 AWS Proton 台，请打开AWS Proton 控制台，然后在导航窗格中选择 “控制面板”。

资源

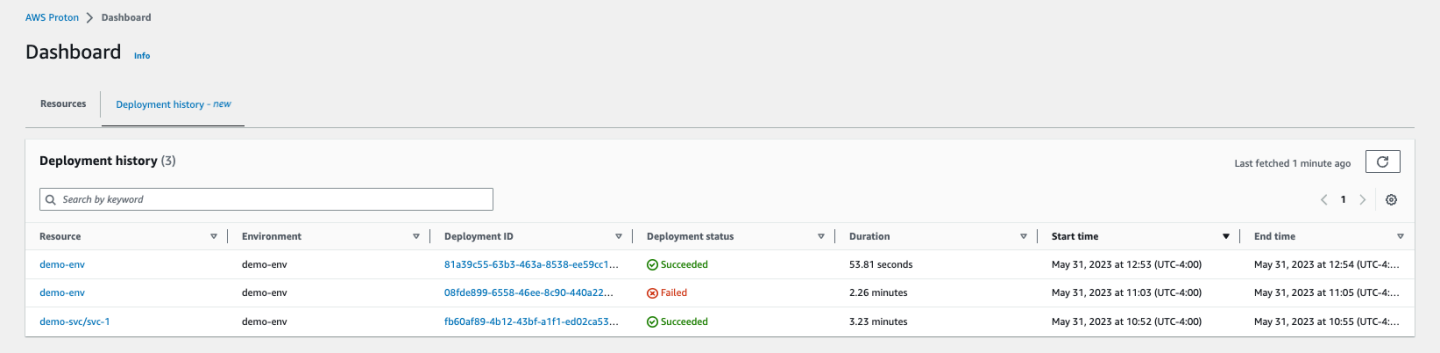


控制面板的第一个选项卡显示您的账户中的所有资源的计数。“资源”选项卡显示您的服务实例、服务、环境和组件数量以及您的资源模板。它还会按每个部署的资源类型的资源状态细分该类型的资源计数。服务实例表显示每个服务实例的详细信息，包括其部署状态、与之关联的 AWS Proton 资源、可用的更新以及一些时间戳。

您可以按任何表属性筛选服务实例列表。例如，您可以筛选以查看在特定时间范围内部署的服务实例，或相对于推荐的主要或次要版本已过时的服务实例。

选择一个服务实例名称以导航到服务实例详细信息页面，您可以在其中进行相应的版本更新。选择任何其他 AWS Proton 资源名称以导航到其详细信息页面，或者选择资源类型以导航到相应的资源列表。

部署历史记录



在“部署历史”选项卡中，您可以查看有关您的部署的详细信息。在部署历史表中，您可以跟踪部署状态以及环境和部署 ID。您可以选择资源名称或部署 ID 以查看甚至更多详细信息，例如部署状态消息和资源输出。该表还允许您筛选任何表属性。

AWS Proton 中的安全性

AWS 十分重视云安全性。为了满足对安全性最敏感的组织的需求，我们打造了具有超高安全性的数据中心和网络架构。作为 AWS 客户，您也将从这些数据中心和网络架构受益。

安全性是 AWS 和您的共同责任。shared responsibility model（责任共担模式）（责任共担模式）（责任共担模式）将其描述为云的安全性和云中的安全性：

- 云的安全性 - AWS 负责保护在 AWS Cloud 中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 AWS 合规性计划的一部分。要了解适用于 AWS Proton 的合规性计划，请参阅合规性计划范围内的 AWS 服务。
- 云中的安全性 - 您的责任由您使用的 AWS 服务 决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 AWS Proton 时应用责任共担模型 以下主题说明如何配置 AWS Proton 以实现您的安全性和合规性目标。您还将了解如何使用其他 AWS 服务以帮助您监控和保护 AWS Proton 资源。

主题

适用于 AWS Proton 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是一项 AWS 服务，可以帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以通过身份验证（登录）和授权（具有权限）来使用 AWS Proton 资源。IAM 是一项无需额外费用即可使用的 AWS 服务。

主题

受众

使用 AWS Identity and Access Management (IAM) 的方式因您可以在 AWS Proton 中执行的操作而异。

服务用户 - 如果使用 AWS Proton 服务来完成任务，则您的管理员会为您提供所需的凭证和权限。当您使用更多 AWS Proton 特征来完成工作时，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 AWS Proton 中的特征，请参阅 对 AWS Proton 身份和访问进行故障排除。

服务管理员 - 如果您在公司负责管理 AWS Proton 资源，则您可能具有 AWS Proton 的完全访问权限。您有责任确定您的服务用户应访问哪些 AWS Proton 特征和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 AWS Proton 搭配使用的更多信息，请参阅AWS Proton 如何与 IAM 协同工作。

IAM 管理员 - 如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 AWS Proton 的访问权限的详细信息。要查看您可在 IAM 中使用的 AWS Proton 基于身份的策略示例，请参阅 适用于 AWS Proton 的基于身份的策略示例。

使用身份进行身份验证

身份验证是使用身份凭证登录 AWS 的方法。您必须作为 AWS 账户根用户、IAM 用户或通过担任 IAM 角色进行身份验证（登录到 AWS）。

您可以使用通过身份源提供的凭证以联合身份登录到 AWS。AWS IAM Identity Center(IAM Identity Center) 用户、您公司的单点登录身份验证以及您的 Google 或 Facebook 凭证都是联合身份的示例。当您以联合身份登录时，管理员以前使用 IAM 角色设置了身份联合验证。当您使用联合身份验证访问 AWS 时，您就是在间接代入角色。

根据用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录到 AWS 的更多信息，请参阅《AWS 登录 用户指南》中的如何登录到您的 AWS 账户。

如果您以编程方式访问 AWS，则 AWS 将提供软件开发工具包 (SDK) 和命令行界面 (CLI)，以便使用您的凭证以加密方式签署您的请求。如果您不使用 AWS 工具，则必须自行对请求签名。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的签署 AWS API 请求。

无论使用何种身份验证方法，您都可能需要提供其它安全信息。例如，AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的在 AWS 中使用多重身份验证（MFA）。

AWS 账户 根用户

创建 AWS 账户 时，最初使用的是一个对账户中所有 AWS 服务 和资源拥有完全访问权限的登录身份。此身份称为 AWS 账户根用户，使用您创建账户时所用的电子邮件地址和密码登录，即可获得该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实操，要求人类用户（包括需要管理员访问权限的用户）结合使用联合身份验证和身份提供程序，以使用临时凭证来访问 AWS 服务。

联合身份是来自企业用户目录、网络身份提供程序、AWS Directory Service、Identity Center 目录的用户，或任何使用通过身份源提供的凭证来访问 AWS 服务 的用户。当联合身份访问 AWS 账户时，他们担任角色，而角色提供临时凭证。

要集中管理访问权限，我们建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到自己的身份源中的一组用户和组以跨所有 AWS 账户 和应用程序使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？

IAM 用户和群组

IAM 用户是 AWS 账户 内对某个人员或应用程序具有特定权限的一个身份。在可能的情况下，建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用群组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用群组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人担任。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

IAM 角色是 AWS 账户 中具有特定权限的身份。它类似于 IAM 用户，但与特定人员不关联。您可以通过切换角色，在 AWS Management Console 中暂时担任 IAM 角色。您可以调用 AWS CLI 或 AWS API 操作或使用自定义网址以代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限 – IAM 用户或角色可代入 IAM 角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户存取权限的主要方式。但是，对于某些 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为座席）。要了解用于跨账户存取的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问 – 某些 AWS 服务 使用其他 AWS 服务 中的特征。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。

- 转发访问会话：当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用主体调用 AWS 服务 的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的政策详情，请参阅转发访问会话。
- 服务角色 - 服务角色是服务代表您在您的账户中执行操作而担任的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
- 服务相关角色 - 服务相关角色是与 AWS 服务关联的一种服务角色。服务可以担任代表您执行操作的角色。服务相关角色显示在您的 AWS 账户中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

- 在 Amazon EC2 上运行的应用程序 - 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将 AWS 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您将创建策略并将其附加到 AWS 身份或资源，以控制 AWS 中的访问。策略是 AWS 中的对象；在与身份或资源相关联时，策略定义它们的权限。在主体（用户、根用户或角色会话）发出请求时，AWS 将评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略在 AWS 中存储为 JSON 文档。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。然后，管理员可以向角色添加 IAM policy，并且用户可以代入角色。

IAM policy 定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设有一个允许 `iam:GetRole` 操作的策略。具有该策略的用户可以从 AWS Management Console、AWS CLI 或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM policy。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、群组或角色中。托管策略是可以附加到AWS 账户中的多个用户、组和角色的独立策略。托管式策略包括 AWS 托管式策略和客户管理型策略。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用来自 IAM 的 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表 (ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3、AWS WAF 和 Amazon VPC 是支持 ACL 的服务示例。要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》 中的访问控制列表 (ACL) 概览。

其他策略类型

AWS 支持额外的、不太常用的策略类型。这些策略类型可以设置更常用的策略类型授予的最大权限。

- 权限边界 - 权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可以为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 字段中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。

- 服务控制策略 (SCP) – SCP 是 JSON 策略，指定了组织或组织单位 (OU) 在 AWS Organizations 中的最大权限。AWS Organizations 服务可以分组和集中管理您的企业拥有的多个 AWS 账户 账户。如果在组织内启用了所有特征，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中实体（包括每个 AWS 账户根用户）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解 AWS 如何确定在涉及多种策略类型时是否允许请求，请参阅《IAM 用户指南》中的策略评估逻辑。

AWS Proton 如何与 IAM 协同工作

在使用 IAM 管理对 AWS Proton 的访问之前，您应该了解哪些 IAM 功能可用于 AWS Proton。

您可以与 AWS Proton 搭配使用的 IAM 特征

IAM 特征	AWS Proton支持
基于身份的策略	可以
基于资源的策略	否
策略操作	可以
策略资源	可以
策略条件键	可以
ACL	否
ABAC（策略中的标签）	可以
临时凭证	可以

IAM 特征	AWS Proton支持
主体权限	可以
服务角色	可以
服务相关角色	可以

要大致了解 AWS Proton 和其他 AWS 服务如何与大多数 IAM 功能一起使用，请参阅《IAM 用户指南》中的使用 IAM 的 AWS 服务。

适用于 AWS Proton 的基于身份的策略

支持基于身份的策略	可以

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM policy。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

适用于 AWS Proton 的基于身份的策略示例

要查看 AWS Proton 基于身份的策略的示例，请参阅适用于 AWS Proton 的基于身份的策略示例。

AWS Proton 内基于资源的策略

支持基于资源的策略	否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当主体和资源处于不同的 AWS 账户 中时，则信任账户中的 IAM 管理员还必须授予主体实体（用户或角色）对资源的访问权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

适用于 AWS Proton 的策略操作

支持策略操作	可以

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与相关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 AWS Proton 操作列表，请参阅《服务授权参考》中的 AWS Proton 定义的操作。

AWS Proton 中的策略操作在操作前使用以下前缀：

```
proton
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "proton:action1",
      "proton:action2"
         ]
```

也可以使用通配符 (*) 指定多个操作。例如，要指定以单词 `List` 开头的所有操作，包括以下操作：

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

要查看 AWS Proton 基于身份的策略的示例，请参阅适用于 AWS Proton 的基于身份的策略示例。

AWS Proton 的策略资源

支持策略资源	可以

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件 下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称 (ARN) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

要查看 AWS Proton 资源类型及其 ARN 的列表，请参阅《服务授权参考》中的 AWS Proton 定义的资源。要了解可以在哪些操作中指定每个资源的 ARN，请参阅 AWS Proton 定义的操作。

要查看 AWS Proton 基于身份的策略的示例，请参阅适用于 AWS Proton 的基于身份的策略示例。

AWS Proton 的策略条件键

支持特定于服务的策略条件键	可以

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，您可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个密钥，则 AWS 使用逻辑 AND 运算评估它们。如果您要为单个条件密钥指定多个值，则 AWS 使用逻辑 OR 运算来评估条件。在授予语句的权限之前必须满足所有的条件。

您也可以在指定条件时使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM policy 元素：变量和标签。

AWS 支持全局条件键和特定于服务的条件键。要查看所有 AWS 全局条件键，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。

要查看 AWS Proton 条件键列表，请参阅《服务授权参考》中的 AWS Proton 的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 AWS Proton 定义的操作。

要查看用于限制对资源的访问的基于条件键的示例策略，请参阅AWS Proton 的基于条件键的策略示例。

AWS Proton 中的访问控制列表 (ACL)

支持 ACL	否

访问控制列表 (ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

访问控制列表 (ACL) 是您可以附加到资源的被授权者列表。他们向账户授予访问所附加到的资源的权限。

AWS Proton 中的基于属性的访问控制 (ABAC)

支持 ABAC（策略中的标签）	可以

基于属性的访问权限控制 (ABAC) 是一种授权策略，该策略基于属性来定义权限。在 AWS 中，这些属性称为标签。您可以将标签附加到 IAM 实体（用户或角色）以及 AWS 资源。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件密钥在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制 (ABAC)。

有关标记 AWS Proton 资源的更多信息，请参阅AWS Proton 资源和标记。

在 AWS Proton 中使用临时凭证

支持临时凭证	可以

某些 AWS 服务 在使用临时凭证登录时无法正常工作。有关更多信息，包括 AWS 服务 与临时凭证配合使用，请参阅《IAM 用户指南》中的使用 IAM 的 AWS 服务。

如果您不使用用户名和密码而用其他方法登录到 AWS Management Console，则使用临时凭证。例如，当您使用贵公司的单点登录（SSO）链接访问 AWS 时，该过程将自动创建临时凭证。当您以用户身份登录控制台，然后切换角色时，还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的切换到角色（控制台）。

您可以使用 AWS CLI 或者 AWS API 创建临时凭证。之后，您可以使用这些临时凭证访问 AWS。AWS 建议您动态生成临时凭证，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

AWS Proton 的跨服务主体权限

支持转发访问会话 (FAS)	可以

当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用主体调用 AWS 服务 的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详细信息，请参阅转发访问会话。

AWS Proton 的服务角色

支持服务角色	可以

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

有关更多信息，请参阅AWS Proton IAM 服务角色策略示例。

> **Warning**
>
> 更改服务角色权限可能会影响 AWS Proton 功能。只有在 AWS Proton 提供指导时，才能编辑服务角色。

AWS Proton 的服务相关角色

支持服务相关角色	可以

服务相关角色是一种与 AWS 服务 相关的服务角色。服务可以担任代表您执行操作的角色。服务相关角色显示在您的 AWS 账户中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅使用 IAM 的 AWS 服务。在表中查找服务相关角色列中包含 Yes 的服务。选择 Yes 链接以查看该服务的服务相关角色文档。

适用于 AWS Proton 的策略示例

在以下几节中查找 AWS Proton IAM 策略示例。

主题

- 适用于 AWS Proton 的基于身份的策略示例
- AWS Proton IAM 服务角色策略示例
- AWS Proton 的基于条件键的策略示例

适用于 AWS Proton 的基于身份的策略示例

默认情况下，用户和角色没有创建或修改 AWS Proton 资源的权限。他们也无法使用 AWS Management Console、AWS Command Line Interface（AWS CLI）或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。然后，管理员可以向角色添加 IAM policy，并且用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的 创建 IAM 策略。

有关 AWS Proton 定义的操作和资源类型的详细信息（包括每种资源类型的 ARN 格式），请参阅《服务授权参考》中的 AWS Proton 的操作、资源和条件键。

主题

- 策略最佳实操
- 适用于 AWS Proton 的基于身份的策略示例的链接

策略最佳实操

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 AWS Proton 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- AWS 托管式策略及转向最低权限许可入门 – 要开始向用户和工作负载授予权限，请使用 AWS 托管式策略来为许多常见使用场景授予权限。您可以在 AWS 账户 中找到这些策略。建议通过定义特定于您的使用场景的 AWS 客户管理型策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略或工作职能的 AWS 托管式策略。
- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果通过特定 AWS 服务（例如 AWS CloudFormation）使用服务操作，您还可以使用条件来授予对服务操作的访问权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM policy，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM policy 语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，有助于制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Access Analyzer 策略验证。
- 需要多重身份验证（MFA）– 如果您所处的场景要求您的 AWS 账户 中有 IAM 用户或根用户，请启用 MFA 来提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实践的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

适用于 AWS Proton 的基于身份的策略示例的链接

适用于 AWS Proton 的基于身份的策略示例的链接

- 适用于 AWS Proton 的 AWS 托管策略
- AWS Proton IAM 服务角色策略示例
- AWS Proton 的基于条件键的策略示例

AWS Proton IAM 服务角色策略示例

管理员拥有并管理 AWS Proton 根据环境和服务模板定义创建的资源。他们将 IAM 服务角色附加到他们的账户，以允许 AWS Proton 代表他们创建资源。当 AWS Proton 在 AWS Proton 环境中将开发人员的应用程序部署为 AWS Proton 服务时，管理员为以后由开发人员拥有和管理的资源提供 IAM 角色和 AWS Key Management Service 密钥。有关 AWS KMS 和数据加密的更多信息，请参阅 AWS Proton 中的数据保护。

服务角色是一种 Amazon Web Services (IAM) 角色，它允许 AWS Proton 代表您调用资源。如果指定服务角色，AWS Proton 将使用该角色的凭证。可以使用服务角色明确指定 AWS Proton 可执行的操作。

您可以使用 IAM 服务创建服务角色及其权限策略。有关创建服务角色的更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

使用 AWS CloudFormation 进行预置的 AWS Proton 服务角色

作为平台团队成员，您可以作为管理员创建 AWS Proton 服务角色，并在创建环境时向 AWS Proton 提供该角色以作为环境的 CloudFormation 服务角色（CreateEnvironment API 操作的 `protonServiceRoleArn` 参数）。在环境或其中运行的任何服务实例使用 AWS Proton 托管式预置和 AWS 预置基础设施时，该角色允许 AWS CloudFormation 代表您对其他服务进行 API 调用。

我们建议您在 AWS Proton 服务角色中使用以下 IAM 角色和信任策略。在您使用 AWS Proton 控制台创建环境并选择创建新角色时，AWS Proton 将该策略添加到为您创建的服务角色中。在缩小该策略的权限范围时，请记住，AWS Proton 由于 `Access Denied` 错误而失败。

> **Important**
>
> 请注意，以下示例中显示的策略为任何可以在您的账户中注册模板的人授予管理员权限。由于我们不知道您在 AWS Proton 模板中定义哪些资源，因此，这些策略具有广泛的权限。我们建议您将权限范围缩小到在您的环境中部署的特定资源。

适用于 AWS CloudFormation 的 AWS Proton 服务角色策略示例

将 *123456789012* 替换为您的 AWS 账户 ID。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:CreateChangeSet",
        "cloudformation:CreateStack",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:DescribeStackDriftDetectionStatus",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStacks",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources",
        "cloudformation:UpdateStack"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "NotAction": [
        "organizations:*",
        "account:*"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeOrganization",
        "account:ListRegions"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

AWS Proton 服务信任策略

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ServiceTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
      }
    }
  }
}
```

缩小范围的 AWS 托管式预置服务角色策略

以下是缩小范围的 AWS Proton 服务角色策略示例，如果您仅需要使用 AWS Proton 服务预置 S3 资源，您可以使用该策略。

```
{
  "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"
          ]
        }
      }
    }
```

```
  ]
}
```

CodeBuild 预置的 AWS Proton 服务角色

作为平台团队成员，您可以作为管理员创建 AWS Proton 服务角色，并在创建环境时向 AWS Proton 提供该角色以作为环境的 CodeBuild 服务角色（CreateEnvironment API 操作的 `codebuildRoleArn` 参数）。在环境或其中运行的任何服务实例使用 CodeBuild 预置以预置基础设施时，该角色允许 AWS Proton 代表您对其他服务进行 API 调用。

在您使用 AWS Proton 控制台创建环境并选择创建新角色时，AWS Proton 将具有管理员权限的策略添加到为您创建的服务角色中。在您创建自己的角色并缩小权限范围时，请记住，AWS Proton 由于 `Access Denied` 错误而失败。

> **Important**
>
> 请注意，AWS Proton 附加到为您创建的角色的策略为任何可以在您的账户中注册模板的人授予管理员权限。由于我们不知道您在 AWS Proton 模板中定义哪些资源，因此，这些策略具有广泛的权限。我们建议您将权限范围缩小到在您的环境中部署的特定资源。

CodeBuild 的 AWS Proton 服务角色策略示例

以下示例为 CodeBuild 提供权限以使用 AWS Cloud Development Kit (AWS CDK) 预置资源。

将 *123456789012* 替换为您的 AWS 账户 ID。

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Action": [
       "logs:CreateLogStream",
       "logs:CreateLogGroup",
       "logs:PutLogEvents"
     ],
     "Resource": [
       "arn:aws:logs:us-east-1:123456789012:log-group:/aws/codebuild/AWSProton-
Shell-*",
       "arn:aws:logs:us-east-1:123456789012:log-group:/aws/codebuild/AWSProton-
Shell-*:*"
```

```
      ],
      "Effect": "Allow"
    },
    {
      "Action": "proton:NotifyResourceDeploymentStatusChange",
      "Resource": "arn:aws:proton:us-east-1:123456789012:*",
      "Effect": "Allow"
    },
    {
      "Action": "sts:AssumeRole",
      "Resource": [
        "arn:aws:iam::123456789012:role/cdk-*-deploy-role-*",
        "arn:aws:iam::123456789012:role/cdk-*-file-publishing-role-*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

AWS Proton CodeBuild 信任策略

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "CodeBuildTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "codebuild.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
      }
    }
  }
}
```

AWS Proton 管道服务角色

要预置服务管道，AWS Proton 需要具有对其他服务进行 API 调用的权限。所需的服务角色与您在创建环境时提供的服务角色类似。不过，用于创建管道的角色是在您的 AWS 账户中的所有服务之间共享的，并且您可以在控制台中将这些角色作为账户设置提供，或通过 UpdateAccountSettings API 操作提供这些角色。

在您使用 AWS Proton 控制台更新账户设置并选择为 AWS CloudFormation 或 CodeBuild 服务角色创建新角色时，AWS Proton 在为您创建的服务角色中添加的策略与前面的AWS 托管式预置角色和CodeBuild 预置角色小节中所述的策略相同。在缩小该策略的权限范围时，请记住，AWS Proton 由于 `Access Denied` 错误而失败。

> **Important**
>
> 请注意，前面几节中的示例策略为任何可以在您的账户中注册模板的人授予管理员权限。由于我们不知道您在 AWS Proton 模板中定义哪些资源，因此，这些策略具有广泛的权限。我们建议您将权限范围缩小到在您的管道中部署的特定资源。

AWS Proton 组件角色

作为平台团队成员，您可以作为管理员创建 AWS Proton 服务角色，并在创建环境时向 AWS Proton 提供该角色以作为环境的 CloudFormation 组件角色（ CreateEnvironment API 操作的 `componentRoleArn` 参数）。该角色缩小了直接定义的组件可以预置的基础设施范围。有关组件的更多信息，请参阅组件。

以下示例策略支持创建直接定义的组件，该组件预置一个 Amazon Simple Storage Service (Amazon S3) 存储桶和相关的访问策略。

AWS Proton 组件角色策略示例

将 *123456789012* 替换为您的 AWS 账户 ID。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:CreateChangeSet",
```

```
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeStacks",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStackEvents",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:DeleteBucket",
        "s3:GetBucket",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:GetPolicy",
        "iam:ListPolicyVersions",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": "cloudformation.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS Proton 组件信任策略

```
{
  "Version": "2012-10-17",
  "Statement": {
```

```
    "Sid": "ServiceTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
      }
    }
  }
}
```

AWS Proton 的基于条件键的策略示例

以下示例 IAM 策略拒绝访问与 Condition 块中指定的模板匹配的 AWS Proton 操作。请注意，仅 AWS Proton 的操作、资源和条件键中列出的操作支持这些条件键。要管理其他操作的权限（例如 DeleteEnvironmentTemplate），您必须使用资源级访问控制。

拒绝对特定模板执行 AWS Proton 模板操作的示例策略：

```
{
    "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"]
                }
            }
        }
    ]
}
```

在下一个示例策略中，第一个资源级语句拒绝访问与 Resource 块中列出的服务模板匹配的 AWS Proton 模板操作（ListServiceTemplates 除外）。第二个语句拒绝访问与 Condition 块中列出的模板匹配的 AWS Proton 操作。

拒绝与特定模板匹配的 AWS Proton 操作的示例策略：

```
{
    "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"
                    ]
                }
            }
        }
    ]
```

```
}
```

最终的策略示例允许与 Condition 块中列出的特定服务模板匹配的开发人员 AWS Proton 操作。

允许与特定模板匹配的 AWS Proton 开发人员操作的示例策略：

```
{
    "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"
                }
            }
        }

    ]
}
```

适用于 AWS Proton 的 AWS 托管策略

要向用户、组和角色添加权限，与自己编写策略相比，使用 AWS 托管策略更简单。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些策略涵盖常见使用案例，可在您的 AWS 账户 中使用。有关 AWS 托管式策略的更多信息，请参阅《IAM 用户指南》中的AWS 托管式策略。

AWS 服务 负责维护和更新 AWS 托管策略。您无法更改 AWS 托管策略中的权限。服务偶尔会向 AWS 托管策略添加额外权限以支持新特征。此类更新会影响附加策略的所有身份（用户、组和角色）。当启动新特征或新操作可用时，服务最有可能会更新 AWS 托管策略。服务不会从 AWS 托管策略中删除权限，因此策略更新不会破坏您的现有权限。

此外，AWS 还支持跨多种服务的工作职能的托管式策略。例如，ReadOnlyAccess AWS 托管策略提供对所有 AWS 服务 和资源的只读访问权限。当服务启动新功能时，AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅《IAM 用户指南》中的适用于工作职能的 AWS 托管策略。

AWS Proton 提供托管 IAM 策略和信任关系，您可以将它们附加到用户、组或角色，以允许对资源和 API 操作进行不同级别的控制。您可以直接应用这些策略，或者也可以使用它们作为自行创建策略的起点。

以下信任关系用于每个 AWS Proton 托管策略。

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

```
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
      }
    }
  }
}
```

AWS 托管策略：AWSProtonFullAccess

您可以将 AWSProtonFullAccess 附加到您的 IAM 实体。AWS Proton 还会将该策略附加到一个服务角色，以允许 AWS Proton 代表您执行操作。

该策略授予管理权限，以允许对 AWS Proton 操作进行完全访问，并对 AWS Proton 依赖的其他 AWS 服务操作进行有限访问。

该策略包括以下关键操作命名空间：

- `proton` - 允许管理员完全访问 AWS Proton API。
- `iam` - 允许管理员将角色传递给 AWS Proton。这是必需的，以便 AWS Proton 可以代表管理员对其他服务进行 API 调用。
- `kms` - 允许管理员为客户托管密钥添加授权。
- `codestar-connections` - 允许管理员列出和传递 CodeStar 连接，以便 AWS Proton 可以使用它们。

权限详细信息

该策略包含以下权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "proton:*",
        "kms:ListAliases",
        "kms:DescribeKey",
        "codestar-connections:ListConnections"
```

```
      ],
      "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": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/sync.proton.amazonaws.com/
AWSServiceRoleForProtonSync",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "sync.proton.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "codestar-connections:PassConnection"
      ],
      "Resource": "arn:aws:codestar-connections:*:*:connection/*",
```

```
      "Condition": {
        "StringEquals": {
          "codestar-connections:PassedToService": "proton.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS 托管策略：AWSProtonDeveloperAccess

您可以将 AWSProtonDeveloperAccess 附加到 IAM 主体。AWS Proton 还会将该策略附加到服务角色，以允许 AWS Proton 代表您执行操作。

该策略授予权限，以允许对 AWS Proton 操作以及 AWS Proton 依赖的其他 AWS 操作进行有限访问。这些权限的范围旨在支持创建和部署 AWS Proton 服务的开发人员的角色。

该策略不提供对 AWS Proton 模板和环境创建、删除和更新 API 的访问。如果开发人员需要的权限比该策略提供的权限更有限，我们建议创建一个缩小范围以授予最低权限的自定义策略。

该策略包括以下关键操作命名空间：

- `proton` - 允许贡献者访问一组有限的 AWS Proton API。
- `codestar-connections` - 允许贡献者列出和传递 CodeStar 连接，以便 AWS Proton 可以使用它们。

权限详细信息

该策略包含以下权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:ListRepositories",
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineExecution",
        "codepipeline:GetPipelineState",
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListPipelines",
```

```
        "codestar-connections:ListConnections",
        "codestar-connections:UseConnection",
        "proton:CancelServiceInstanceDeployment",
        "proton:CancelServicePipelineDeployment",
        "proton:CreateService",
        "proton:DeleteService",
        "proton:GetAccountRoles",
        "proton:GetAccountSettings",
        "proton:GetEnvironment",
        "proton:GetEnvironmentAccountConnection",
        "proton:GetEnvironmentTemplate",
        "proton:GetEnvironmentTemplateMajorVersion",
        "proton:GetEnvironmentTemplateMinorVersion",
        "proton:GetEnvironmentTemplateVersion",
        "proton:GetRepository",
        "proton:GetRepositorySyncStatus",
        "proton:GetResourcesSummary",
        "proton:GetService",
        "proton:GetServiceInstance",
        "proton:GetServiceTemplate",
        "proton:GetServiceTemplateMajorVersion",
        "proton:GetServiceTemplateMinorVersion",
        "proton:GetServiceTemplateVersion",
        "proton:GetTemplateSyncConfig",
        "proton:GetTemplateSyncStatus",
        "proton:ListEnvironmentAccountConnections",
        "proton:ListEnvironmentOutputs",
        "proton:ListEnvironmentProvisionedResources",
        "proton:ListEnvironments",
        "proton:ListEnvironmentTemplateMajorVersions",
        "proton:ListEnvironmentTemplateMinorVersions",
        "proton:ListEnvironmentTemplates",
        "proton:ListEnvironmentTemplateVersions",
        "proton:ListRepositories",
        "proton:ListRepositorySyncDefinitions",
        "proton:ListServiceInstanceOutputs",
        "proton:ListServiceInstanceProvisionedResources",
        "proton:ListServiceInstances",
        "proton:ListServicePipelineOutputs",
        "proton:ListServicePipelineProvisionedResources",
        "proton:ListServices",
        "proton:ListServiceTemplateMajorVersions",
        "proton:ListServiceTemplateMinorVersions",
        "proton:ListServiceTemplates",
```

```
        "proton:ListServiceTemplateVersions",
        "proton:ListTagsForResource",
        "proton:UpdateService",
        "proton:UpdateServiceInstance",
        "proton:UpdateServicePipeline",
        "s3:ListAllMyBuckets",
        "s3:ListBucket"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "codestar-connections:PassConnection",
      "Resource": "arn:aws:codestar-connections:*:*:connection/*",
      "Condition": {
        "StringEquals": {
          "codestar-connections:PassedToService": "proton.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS 托管策略：AWSProtonReadOnlyAccess

您可以将 AWSProtonReadOnlyAccess 附加到您的 IAM 实体。AWS Proton 还会将该策略附加到一个服务角色，以允许 AWS Proton 代表您执行操作。

该策略授予权限，以允许对 AWS Proton 操作进行只读访问，并对 AWS Proton 依赖的其他 AWS 服务操作进行有限只读访问。

该策略包括以下关键操作命名空间：

- `proton` - 允许贡献者对 AWS Proton API 进行只读访问。

权限详细信息

该策略包含以下权限。

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

```
      "Effect": "Allow",
      "Action": [
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListPipelines",
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "proton:GetAccountRoles",
        "proton:GetAccountSettings",
        "proton:GetEnvironment",
        "proton:GetEnvironmentAccountConnection",
        "proton:GetEnvironmentTemplate",
        "proton:GetEnvironmentTemplateMajorVersion",
        "proton:GetEnvironmentTemplateMinorVersion",
        "proton:GetEnvironmentTemplateVersion",
        "proton:GetRepository",
        "proton:GetRepositorySyncStatus",
        "proton:GetResourcesSummary",
        "proton:GetService",
        "proton:GetServiceInstance",
        "proton:GetServiceTemplate",
        "proton:GetServiceTemplateMajorVersion",
        "proton:GetServiceTemplateMinorVersion",
        "proton:GetServiceTemplateVersion",
        "proton:GetTemplateSyncConfig",
        "proton:GetTemplateSyncStatus",
        "proton:ListEnvironmentAccountConnections",
        "proton:ListEnvironmentOutputs",
        "proton:ListEnvironmentProvisionedResources",
        "proton:ListEnvironments",
        "proton:ListEnvironmentTemplateMajorVersions",
        "proton:ListEnvironmentTemplateMinorVersions",
        "proton:ListEnvironmentTemplates",
        "proton:ListEnvironmentTemplateVersions",
        "proton:ListRepositories",
        "proton:ListRepositorySyncDefinitions",
        "proton:ListServiceInstanceOutputs",
        "proton:ListServiceInstanceProvisionedResources",
        "proton:ListServiceInstances",
        "proton:ListServicePipelineOutputs",
        "proton:ListServicePipelineProvisionedResources",
        "proton:ListServices",
        "proton:ListServiceTemplateMajorVersions",
        "proton:ListServiceTemplateMinorVersions",
```

```
        "proton:ListServiceTemplates",
        "proton:ListServiceTemplateVersions",
        "proton:ListTagsForResource"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS 托管策略：AWSProtonSyncServiceRolePolicy

AWS Proton 将该策略附加到 AWSServiceRoleForProtonSync 服务相关角色，以允许 AWS Proton 执行模板同步。

该策略授予权限，以允许对 AWS Proton 操作以及 AWS Proton 依赖的其他 AWS 服务操作进行有限访问。

该策略包括以下关键操作命名空间：

- `proton` - 允许 AWS Proton 同步对 AWS Proton API 进行有限访问。
- `codestar-connections` - 允许 AWS Proton 同步对 CodeConnections API 进行有限访问。

有关 AWSProtonSyncServiceRolePolicy 的权限详细信息，请参阅 Service-linked role permissions for AWS Proton。

AWS 托管策略：AWSProtonCodeBuildProvisioningBasicAccess

CodeBuild 为 AWS Proton CodeBuild 预置运行构建所需的权限。您可以将 `AWSProtonCodeBuildProvisioningBasicAccess` 附加到您的 CodeBuild 预置角色。

该策略授予 AWS Proton CodeBuild 预置运行所需的最低权限。它授予允许 CodeBuild 生成构建日志的权限。它还授予 Proton 向 AWS Proton 用户提供基础设施即代码 (IaC) 输出的权限。它不提供 IaC 工具管理基础设施所需的权限。

该策略包括以下关键操作命名空间：

- `logs` - 允许 CodeBuild 生成构建日志。如果没有该权限，CodeBuild 将无法启动。
- `proton` - 允许 CodeBuild 预置命令调用 `aws proton notify-resource-deployment-status-change` 以更新给定 AWS Proton 资源的 IaaC 输出。

权限详细信息

该策略包含以下权限。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
    "logs:PutLogEvents"
   ],
   "Resource": [
    "arn:aws:logs:*:*:log-group:/aws/codebuild/AWSProton-*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": "proton:NotifyResourceDeploymentStatusChange",
   "Resource": "arn:aws:proton:*:*:*"
  }
 ]
}
```

AWS 托管策略：AWSProtonCodeBuildProvisioningServiceRolePolicy

AWS Proton 将该策略附加到 AWSServiceRoleForProtonCodeBuildProvisioning 服务相关角色，以允许 AWS Proton 执行基于 CodeBuild 的预置。

该策略授予权限，以允许对 AWS Proton 依赖的 AWS 服务操作进行有限访问。

该策略包括以下关键操作命名空间：

- `cloudformation` - 允许 AWS Proton 基于 CodeBuild 的预置对 AWS CloudFormation API 进行有限访问。
- `codebuild` - 允许 AWS Proton 基于 CodeBuild 的预置对 CodeBuild API 进行有限访问。
- `iam` - 允许管理员将角色传递给 AWS Proton。这是必需的，以便 AWS Proton 可以代表管理员对其他服务进行 API 调用。
- `servicequotas` - 允许 AWS Proton 检查 CodeBuild 并发构建限制，以确保正确进行构建排队。

权限详细信息

该策略包含以下权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CreateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeStacks",
        "cloudformation:DescribeStackEvents",
        "cloudformation:ListStackResources"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/AWSProton-CodeBuild-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:CreateProject",
        "codebuild:DeleteProject",
        "codebuild:UpdateProject",
        "codebuild:StartBuild",
        "codebuild:StopBuild",
        "codebuild:RetryBuild",
        "codebuild:BatchGetBuilds",
        "codebuild:BatchGetProjects"
      ],
      "Resource": "arn:aws:codebuild:*:*:project/AWSProton*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEqualsIfExists": {
          "iam:PassedToService": "codebuild.amazonaws.com"
```

```
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
          "servicequotas:GetServiceQuota"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS 托管策略：AwsProtonServiceGitSyncServiceRolePolicy

AWS Proton 将该策略附加到 AwsProtonServiceGitSyncServiceRolePolicy 服务相关角色，以允许 AWS Proton 执行服务同步。

该策略授予权限，以允许对 AWS Proton 操作以及 AWS Proton 依赖的其他 AWS 服务操作进行有限访问。

该策略包括以下关键操作命名空间：

- `proton` - 允许 AWS Proton 同步对 AWS Proton API 进行有限访问。

有关 AwsProtonServiceGitSyncServiceRolePolicy 的权限详细信息，请参阅 Service-linked role permissions for AWS Proton。

AWS Proton 更新了 AWS 托管策略

查看有关 AWS Proton 的 AWS 托管策略更新的详细信息（从该服务开始跟踪这些更改开始）。有关此页面更改的自动提示，请订阅 AWS Proton 文档历史记录页面上的 RSS 源。

更改	说明	日期
AWSProtonCodeBuild ProvisioningServiceRolePolicy - 更新现有的策略	AWS Proton 更新了该策略以添加权限，以确保账户具有所需的 CodeBuild 并发构建限制，以便使用 CodeBuild 预置。	2023 年 5 月 12 日

更改	说明	日期
AwsProtonServiceGitSyncServiceRolePolicy - 新策略	AWS Proton 添加了一个新策略以允许 AWS Proton 执行服务同步。该策略用于 AWSServiceRoleForProtonServiceSync 服务相关角色。	2023 年 3 月 31 日
AWSProtonDeveloperAccess - 更新现有的策略	AWS Proton 添加了一个新的 `GetResourcesSummary` 操作，以允许您查看模板、部署的模板资源和过时资源的摘要。	2022 年 11 月 18 日
AWSProtonReadOnlyAccess - 更新现有的策略	AWS Proton 添加了一个新的 `GetResourcesSummary` 操作，以允许您查看模板、部署的模板资源和过时资源的摘要。	2022 年 11 月 18 日
AWSProtonCodeBuildProvisioningBasicAccess - 新策略	AWS Proton 添加了一个新策略，以向 CodeBuild 提供为 AWS Proton CodeBuild 预置运行构建所需的权限。	2022 年 11 月 16 日
AWSProtonCodeBuildProvisioningServiceRolePolicy - 新策略	AWS Proton 添加了一个新策略，以允许 AWS Proton 执行与基于 CodeBuild 的预置相关的操作。该策略用于 AWSServiceRoleForProtonCodeBuildProvisioning 服务相关角色。	2022 年 9 月 2 日

更改	说明	日期
AWSProtonFullAccess - 更新现有的策略	AWS Proton 更新了该策略，以提供对新 AWS Proton API 操作的访问，并修复某些 AWS Proton 控制台操作的权限问题。	2022 年 3 月 30 日
AWSProtonDeveloperAccess - 更新现有的策略	AWS Proton 更新该策略，以提供对新 AWS Proton API 操作的访问，并修复某些 AWS Proton 控制台操作的权限问题。	2022 年 3 月 30 日
AWSProtonReadOnlyAccess - 更新现有的策略	AWS Proton 更新该策略，以提供对新 AWS Proton API 操作的访问，并修复某些 AWS Proton 控制台操作的权限问题。	2022 年 3 月 30 日
AWSProtonSyncServiceRolePolicy - 新策略	AWS Proton 添加了一个新策略，以允许 AWS Proton 执行与模板同步相关的操作。该策略用于 AWSServiceRoleForProtonSync 服务相关角色。	2021 年 11 月 23 日
AWSProtonFullAccess - 新策略	AWS Proton 添加了一个新策略，以提供对 AWS Proton API 操作和 AWS Proton 控制台的管理角色访问。	2021 年 6 月 9 日
AWSProtonDeveloperAccess - 新策略	AWS Proton 添加了一个新策略，以提供对 AWS Proton API 操作和 AWS Proton 控制台的开发人员角色访问。	2021 年 6 月 9 日

更改	说明	日期
AWSProtonReadOnlyAccess - 新策略	AWS Proton 添加了一个新策略，以提供对 AWS Proton API 操作和 AWS Proton 控制台的只读访问。	2021 年 6 月 9 日
AWS Proton 已开启跟踪更改	AWS Proton 为其 AWS 托管策略开启了跟踪更改。	2021 年 6 月 9 日

将服务相关角色用于 AWS Proton

AWS Proton 使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 AWS Proton 直接相关。服务相关角色由 AWS Proton 预定义，并包含该服务代表您调用其他 AWS 服务所需的一切权限。

主题

- 使用角色进行 AWS Proton 同步
- 使用角色进行基于 CodeBuild 的预置

使用角色进行 AWS Proton 同步

AWS Proton 使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 AWS Proton 直接相关。服务相关角色由 AWS Proton 预定义，并包含该服务代表您调用其他 AWS 服务所需的一切权限。

服务相关角色使 AWS Proton 的设置更轻松，因为您不必手动添加必要的权限。AWS Proton 定义其服务相关角色的权限，除非另行定义，否则仅 AWS Proton 可以担任其角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这将保护您的 AWS Proton 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的 AWS 服务，并查找 Service-linked roles（服务相关角色）列中显示为 Yes（是）的服务。请选择是与查看该服务的服务相关角色文档的链接。

AWS Proton 的服务相关角色权限

AWS Proton 使用两个名为 AWSServiceRoleForProtonSync 和 AWSServiceRoleForProtonServiceSync 的服务相关角色。

AWSServiceRoleForProtonSync 服务相关角色信任以下服务担任该角色：

- `sync.proton.amazonaws.com`

名为 AWSProtonSyncServiceRolePolicy 的角色权限策略允许 AWS Proton 对指定资源完成以下操作：

- 操作：针对 AWS Proton 模板和模板版本 的创建、管理和读取
- 操作：针对 CodeConnections 的使用连接

AWSProtonSyncServiceRolePolicy

该策略包含以下权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SyncToProton",
      "Effect": "Allow",
      "Action": [
        "proton:UpdateServiceTemplateVersion",
        "proton:UpdateServiceTemplate",
        "proton:UpdateEnvironmentTemplateVersion",
        "proton:UpdateEnvironmentTemplate",
        "proton:GetServiceTemplateVersion",
        "proton:GetServiceTemplate",
        "proton:GetEnvironmentTemplateVersion",
        "proton:GetEnvironmentTemplate",
        "proton:DeleteServiceTemplateVersion",
        "proton:DeleteEnvironmentTemplateVersion",
        "proton:CreateServiceTemplateVersion",
        "proton:CreateServiceTemplate",
        "proton:CreateEnvironmentTemplateVersion",
        "proton:CreateEnvironmentTemplate",
        "proton:ListEnvironmentTemplateVersions",
```

```
        "proton:ListServiceTemplateVersions",
        "proton:CreateEnvironmentTemplateMajorVersion",
        "proton:CreateServiceTemplateMajorVersion"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AccessGitRepos",
      "Effect": "Allow",
      "Action": [
        "codestar-connections:UseConnection"
      ],
      "Resource": "arn:aws:codestar-connections:*:*:connection/*"
    }
  ]
}
```

有关 AWSProtonSyncServiceRolePolicy 的信息，请参阅 AWS 托管策略：AWSProtonSyncServiceRolePolicy。

AWSServiceRoleForProtonServiceSync 服务相关角色信任以下服务担任该角色：

- `service-sync.proton.amazonaws.com`

名为 AWSServiceRoleForProtonServiceSync 的角色权限策略允许 AWS Proton 对指定资源完成以下操作：

- 操作：针对 AWS Proton 服务和服务实例的创建、管理和读取

AwsProtonServiceGitSyncServiceRolePolicy

该策略包含以下权限：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "ProtonServiceSync",
   "Effect": "Allow",
   "Action": [
    "proton:GetService",
```

```
    "proton:UpdateService",
    "proton:UpdateServicePipeline",
    "proton:CreateServiceInstance",
    "proton:GetServiceInstance",
    "proton:UpdateServiceInstance",
    "proton:ListServiceInstances",
    "proton:GetComponent",
    "proton:CreateComponent",
    "proton:ListComponents",
    "proton:UpdateComponent",
    "proton:GetEnvironment",
    "proton:CreateEnvironment",
    "proton:ListEnvironments",
    "proton:UpdateEnvironment"
   ],
   "Resource": "*"
  }
 ]
}
```

有关 AwsProtonServiceSyncServiceRolePolicy 的信息，请参阅 AWS 托管策略：AwsProtonServiceSyncServiceRolePolicy。

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 AWS Proton 创建服务相关角色

您无需手动创建服务相关角色。当您通过 AWS Management Console、AWS CLI 或 AWS API 在 AWS Proton 中配置存储库或服务以进行同步时，AWS Proton 为您创建服务相关角色。

如果您删除此服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您在 AWS Proton 中配置存储库或服务以进行同步时，AWS Proton 再次为您创建服务相关角色。

要重新创建 AWSServiceRoleForProtonSync 服务相关角色，您可能需要配置一个存储库以进行同步；要重新创建 AWSServiceRoleForProtonServiceSync，您可能需要配置一个服务以进行同步。

为 AWS Proton 编辑服务相关角色

AWS Proton 不允许您编辑 AWSServiceRoleForProtonSync 服务相关角色。在创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。不过，您可以使用 IAM 编辑角色的说明。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除 AWS Proton 的服务相关角色

您不需要手动删除 AWSServiceRoleForProtonSync 角色。在您删除 AWS Management Console、AWS CLI 或 AWS API 中用于存储库同步的所有 AWS Proton 链接存储库时，AWS Proton 清理资源并为您删除服务相关角色。

AWS Proton 服务相关角色的受支持区域

AWS Proton 支持在该服务可用的所有 AWS 区域中使用服务相关角色。有关更多信息，请参阅 AWS 一般参考 中的 AWS Proton endpoints and quotas。

使用角色进行基于 CodeBuild 的预置

AWS Proton 使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 AWS Proton 直接相关。服务相关角色由 AWS Proton 预定义，并包含该服务代表您调用其他 AWS 服务所需的一切权限。

服务相关角色使 AWS Proton 的设置更轻松，因为您不必手动添加必要的权限。AWS Proton 定义其服务相关角色的权限，除非另行定义，否则仅 AWS Proton 可以担任其角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这将保护您的 AWS Proton 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的 AWS 服务，并查找 Service-linked roles（服务相关角色）列中显示为 Yes（是）的服务。请选择是与查看该服务的服务相关角色文档的链接。

AWS Proton 的服务相关角色权限

AWS Proton 使用名为 AWSServiceRoleForProtonCodeBuildProvisioning 的服务相关角色 - 用于 AWS Proton CodeBuild 预置的服务相关角色。

AWSServiceRoleForProtonCodeBuildProvisioning 服务相关角色信任以下服务担任该角色：

- `codebuild.proton.amazonaws.com`

名为 AWSProtonCodeBuildProvisioningServiceRolePolicy 的角色权限策略允许 AWS Proton 对指定资源完成以下操作：

- 操作：针对 AWS CloudFormation 堆栈和转换的创建、管理和读取

- 操作：针对 CodeBuild 项目和构建的创建、管理和读取

AWSProtonCodeBuildProvisioningServiceRolePolicy

该策略包含以下权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CreateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeStacks",
        "cloudformation:DescribeStackEvents",
        "cloudformation:ListStackResources"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/AWSProton-CodeBuild-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:CreateProject",
        "codebuild:DeleteProject",
        "codebuild:UpdateProject",
        "codebuild:StartBuild",
        "codebuild:StopBuild",
        "codebuild:RetryBuild",
        "codebuild:BatchGetBuilds",
        "codebuild:BatchGetProjects"
      ],
      "Resource": "arn:aws:codebuild:*:*:project/AWSProton*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
```

```
      "Condition": {
        "StringEqualsIfExists": {
          "iam:PassedToService": "codebuild.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
          "servicequotas:GetServiceQuota"
      ],
      "Resource": "*"
    }
  ]
}
```

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 AWS Proton 创建服务相关角色

您无需手动创建服务相关角色。在 AWS Proton AWS Management Console、AWS CLI 或 AWS API 中创建使用基于 CodeBuild 的预置的环境时，AWS Proton 为您创建服务相关角色。

如果您删除此服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。在 AWS Proton 中创建使用基于 CodeBuild 的预置的环境时，AWS Proton 再次为您创建服务相关角色。

为 AWS Proton 编辑服务相关角色

AWS Proton 不允许您编辑 AWSServiceRoleForProtonCodeBuildProvisioning 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除 AWS Proton 的服务相关角色

如果不再需要使用某个需要服务相关角色的特征或服务，建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。不过，您必须删除 AWS Proton 中使用基于 CodeBuild 的预置的所有环境和服务（实例和管道），然后才能手动删除该角色。

手动删除服务相关角色

使用 IAM 控制台、AWS CLI 或 AWS API 删除 AWSServiceRoleForProtonCodeBuildProvisioning 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

AWS Proton 服务相关角色的受支持区域

AWS Proton 支持在该服务可用的所有 AWS 区域中使用服务相关角色。有关更多信息，请参阅 AWS 一般参考 中的 AWS Proton endpoints and quotas。

对 AWS Proton 身份和访问进行故障排除

使用以下信息可帮助您诊断和修复在使用 AWS Proton 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 AWS Proton 中执行操作
- 我无权执行 iam:PassRole
- 我希望允许我的 AWS 账户以外的人访问我的 AWS Proton 资源

我无权在 AWS Proton 中执行操作

如果 AWS Management Console 告诉您，您无权执行某个操作，则必须联系您的管理员寻求帮助。管理员是向您提供登录凭证的人。

当 `mateojackson` IAM 用户尝试使用控制台查看有关虚构 *`my-example-widget`* 资源的详细信息，但不拥有虚构 `proton:`*`GetWidget`* 权限时，会发生以下示例错误。

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

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 `proton:`*`GetWidget`* 操作访问 *`my-example-widget`* 资源。

我无权执行 iam:PassRole

如果您收到一个错误，表明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 AWS Proton。

有些 AWS 服务允许您将现有角色传递到该服务，而不是创建新服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的 IAM 用户尝试使用控制台在 AWS Proton 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

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

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系 AWS 管理员。您的管理员是提供登录凭证的人。

我希望允许我的 AWS 账户以外的人访问我的 AWS Proton 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACL) 的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 AWS Proton 是否支持这些特征，请参阅 AWS Proton 如何与 IAM 协同工作。
- 要了解如何为您拥有的 AWS 账户中的资源提供访问权限，请参阅《IAM 用户指南》中的为您拥有的另一个 AWS 账户中的 IAM 用户提供访问权限。
- 要了解如何为第三方 AWS 账户 提供您的资源的访问权限，请参阅《IAM 用户指南》中的为第三方拥有的 AWS 账户 提供访问权限。
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

AWS Proton 中的配置和漏洞分析

AWS Proton 不会为客户提供的代码提供补丁或更新。客户负责更新并将补丁应用于自己的代码，包括在 AWS Proton 上运行的服务和应用程序的源代码以及在其服务和环境模板捆绑包中提供的代码。

客户负责更新和修补其环境和服务中的基础设施资源。AWS Proton 不会自动更新或修补任何资源。客户应查阅其架构中的资源的文档，以了解他们的相应修补策略。

除了为服务和环境模板次要版本提供客户请求的环境和服务更新以外，AWS Proton 不会为客户在其服务和环境模板以及模板捆绑包中定义的资源提供补丁或更新。

有关更多详细信息，请参阅以下资源：

- 责任共担模式
- Amazon Web Services：安全过程概述

AWS Proton 中的数据保护

AWS Proton 符合 AWS 责任共担模式，其中包括数据保护法规和准则。AWS 负责保护运行所有 AWS 服务的全球基础设施。AWS 保持对该基础设施上托管的数据的控制，包括用于处理客户内容和个人数据的安全配置控制。作为数据控制者或数据处理者，AWS 客户和 APN 合作伙伴对他们放入 AWS Cloud的任何个人数据承担责任。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS Identity and Access Management (IAM) 设置各个用户账户，以便仅为每个用户提供履行其工作职责所需的权限。我们还建议您通过以下方式保护您的数据：

- 对每个账户使用 multi-factor authentication（MFA）。
- 使用 SSL/TLS 与 AWS 资源进行通信。建议使用 TLS 1.2 或更高版本。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务 中的所有默认安全控制。

我们强烈建议您切勿将敏感的可识别信息（例如您的客户的账号）放入自由格式文本字段中，例如名称字段。这包括使用控制台、API、AWS CLI 或 AWS SDK 处理 AWS Proton 或其他 AWS 服务时。对于您在资源标识符或与 AWS 资源管理相关的类似项目的自由格式文本字段中输入的任何数据，可能会选择这些数据以包含在诊断日志中。当您向外部服务器提供 URL 时，请勿在 URL 中包含凭证信息来验证您对该服务器的请求。

有关数据保护的更多信息，请参阅AWS安全性博客上的AWS责任共担模式和 GDPR 博客文章。

服务器端静态加密

如果您选择在存储模板捆绑包的 S3 存储桶中静态加密模板捆绑包中的敏感数据，您必须使用 SSE-S3 或 SSE-KMS 密钥以允许 AWS Proton 检索模板捆绑包，以便将其附加到注册的 AWS Proton 模板。

传输中加密

所有服务到服务通信都使用 SSL/TLS 进行传输中加密。

AWS Proton 加密密钥管理

在 AWS Proton 中，所有客户数据默认使用 AWS Proton 拥有的密钥进行加密。如果您提供客户拥有和管理的 AWS KMS 密钥，则使用客户提供的密钥加密所有客户数据，如以下段落中所述。

在创建 AWS Proton 模板时，您需要指定您的密钥，并且 AWS Proton 使用您的凭证创建允许 AWS Proton 使用您的密钥的授权。

如果您手动停用授权，或者禁用或删除指定的密钥，则 AWS Proton 无法读取由指定的密钥加密的数据并引发 `ValidationException`。

AWS Proton 加密上下文

AWS Proton 支持加密上下文标头。加密上下文是一组可选的键值对，可以包含有关数据的其他上下文信息。有关加密上下文的一般信息，请参阅 AWS Key Management Service 开发人员指南中的 AWS Key Management Service 概念 - 加密上下文。

加密上下文是一组包含任意非机密数据的键值对。在加密数据的请求中包含加密上下文时，AWS KMS 以加密方式将加密上下文绑定到加密的数据。要解密数据，您必须传入相同的加密上下文。

客户可以使用加密上下文在审核记录和日志中确定客户托管密钥的使用情况。它还会以明文形式出现在日志中，例如 AWS CloudTrail 和 Amazon CloudWatch Logs。

AWS Proton 不接受任何客户指定的加密上下文或外部指定的加密上下文。

AWS Proton 添加以下加密上下文。

```
{
  "aws:proton:template": "<proton-template-arn>",
  "aws:proton:resource": "<proton-resource-arn>"
}
```

第一个加密上下文标识与资源关联的 AWS Proton 模板，并且还作为客户托管密钥权限和授权的限制。

第二加密上下文标识加密的 AWS Proton 资源。

以下示例显示 AWS Proton 加密上下文的用途。

开发人员创建服务实例。

```
{
```

```
  "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"
}
```

管理员创建模板。

```
{
  "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"
}
```

中的基础设施安全 AWS Proton

作为一项托管服务 AWS Proton ，受 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用 AWS Proton 通过网络进行访问。客户端必须支持以下内容：

- 传输层安全性协议 (TLS) 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

要改善网络隔离，可以按照下一节所述使用 AWS PrivateLink 。

AWS Proton 和接口 VPC 终端节点 (AWS PrivateLink)

您可以通过创建接口 VPC 终端节点在您 AWS Proton 的 VPC 之间建立私有连接。接口端点由一项技术提供支持 AWS PrivateLink，该技术使您无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接即可私密访问 AWS Proton API。您的 VPC 中的实例不需要公有 IP 地址即可与 AWS Proton API 通信。您的 VPC 和 VPC 之间的流量 AWS Proton 不会离开亚马逊网络。

每个接口端点均由子网中的一个或多个弹性网络接口表示。

有关更多信息，请参阅《Amazon VPC 用户指南》中的接口 VPC 端点 (AWS PrivateLink)。

AWS Proton VPC 终端节点的注意事项

在为设置接口 VPC 终端节点之前 AWS Proton，请务必查看 Amazon VPC 用户指南中的接口终端节点属性和限制。

AWS Proton 支持从您的 VPC 调用其所有 API 操作。

支持 VPC 终端节点策略 AWS Proton。默认情况下，允许通过终端节点进行完全访问。 AWS Proton 有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 端点控制对服务的访问权限。

为创建接口 VPC 终端节点 AWS Proton

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 AWS Proton 服务创建 VPC 终端节点。有关更多信息，请参阅《Amazon VPC 用户指南》中的创建接口端点。

AWS Proton 使用以下服务名称创建 VPC 终端节点：

- com.amazonaws.*region*.proton

例如，如果您为终端节点启用私有 DNS，则可以使用该终端节点的默认 DNS 名称向 AWS Proton 发出 API 请求`proton.`*region*`.amazonaws.com`。

有关更多信息，请参阅《Amazon VPC 用户指南》中的通过接口端点访问服务。

为创建 VPC 终端节点策略 AWS Proton

您可以为 VPC 端点附加控制对 AWS Proton的访问的端点策略。该策略指定以下信息：

- 可执行操作的主体。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 端点控制对服务的访问。

示例：用于 AWS Proton 操作的 VPC 终端节点策略

以下是的终端节点策略示例 AWS Proton。当连接到终端节点时，此策略授予所有委托人对所有资源 AWS Proton 执行所列操作的访问权限。

```
{
  "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": "*"
    }
  ]
}
```

AWS Proton 中的日志记录和监控

监控是保持 AWS Proton 和其他 AWS 解决方案的可靠性、可用性和性能的重要组成部分。AWS 提供以下监控工具以监控 AWS Proton 中运行的实例，在出现问题时进行报告，并在适当的时候自动采取措施。

目前，AWS Proton 本身没有与 Amazon CloudWatch Logs 或 AWS Trusted Advisor 集成在一起。管理员可以配置和使用 CloudWatch 监控其服务和环境模板中定义的其他 AWS 服务。AWS Proton 与 AWS CloudTrail 集成在一起。

- Amazon CloudWatch 实时监控您的 AWS 资源以及在 AWS 上运行的应用程序。您可以收集和跟踪指标，创建自定义的控制面板，以及设置警报以在指定的指标达到您指定的阈值时通知您或采取措施。例如，您可以具有 Amazon EC2 实例的 CloudWatch 跟踪 CPU 使用率或其他指标并且在需要时自动启动新实例。有关更多信息，请参阅 Amazon CloudWatch 用户指南。
- 通过使用 Amazon CloudWatch Logs，您可以监控、存储和访问来自 Amazon EC2 实例、CloudTrail 和其他来源的日志文件。CloudWatch Logs 可以监控日志文件中的信息，并在达到特定阈值时通知您。您还可以在高持久性存储中检索您的日志数据。有关更多信息，请参阅 Amazon CloudWatch Logs 用户指南。
- AWS CloudTrail 捕获由您的 AWS 账户 或代表该账户发出的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 存储桶。您可以标识哪些用户和账户调用了 AWS、从中发出调用的源 IP 地址以及调用的发生时间。有关更多信息，请参阅 AWS CloudTrail 用户指南。
- Amazon EventBridge 是一种无服务器事件总线服务，可以轻松地将应用程序与来自各种来源的数据相连接。EventBridge 提供来自您自己的应用程序、软件即服务 (SaaS) 应用程序和 AWS 服务的实时数据流，并将该数据路由到 Lambda 等目标。这使您能够监控服务中发生的事件，并构建事件驱动的架构。有关更多信息，请参阅 使用实现自动 AWS Proton 化 EventBridge 和 EventBridge User Guide。

AWS Proton 中的故障恢复能力

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。AWS 区域提供多个物理上分离和隔离的可用区，这些可用区是通过低延迟、高吞吐量和高冗余性网络连接的。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关AWS 区域和可用区的更多信息，请参阅AWS全球基础设施。

除了 AWS 全球基础设施之外，AWS Proton 还提供了相应功能来帮助支持您的数据弹性和备份需求。

AWS Proton 备份

AWS Proton 保留所有客户数据的备份。在完全中断的情况下，可以使用该备份从以前的有效状态还原 AWS Proton 和客户数据。

AWS Proton 的安全最佳实践

AWS Proton 提供了您在开发和实施自己的安全策略时考虑的安全功能。以下最佳实践是一般指导原则，并不代表完整安全解决方案。这些最佳实践可能不适合您的环境或不满足您的环境要求，请将其视为有用的考虑因素而不是惯例。

主题

使用 IAM 控制访问

IAM 是一项 AWS 服务，可用于管理 AWS 中的用户及其权限。您可以将 IAM 与 AWS Proton 一起使用以指定管理员和开发人员可以执行的 AWS Proton 操作，例如管理模板、环境或服务。您可以使用 IAM 服务角色允许 AWS Proton 代表您调用其他服务。

有关 适用于 AWS Proton 的 Identity and Access Management 和 IAM 角色的更多信息，请参阅AWS Proton。

实施最低权限访问。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的 IAM 中的策略和权限。

不要将凭证嵌入到您的模板和模板捆绑包中

我们建议您在堆栈模板中使用动态引用，而不是在 AWS CloudFormation 模板和模板捆绑包中嵌入敏感信息。

动态引用为您提供了一种简洁且强大的方法，以引用在其他服务（例如 AWS Systems Manager Parameter Store 或 AWS Secrets Manager）中存储和管理的外部值。当您使用动态引用时，CloudFormation 会在堆栈和更改集合操作期间根据需要检索指定引用的值，并将值传递到相应的资源。但是，CloudFormation 从不存储实际引用值。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的使用动态引用以指定模板值。

AWS Secrets Manager 帮助您安全地加密、存储和检索数据库和其他服务的凭证。AWS Systems Manager Parameter Store 提供安全的分层存储以管理配置数据。

有关定义模板参数的更多信息，请参阅《AWS CloudFormation 用户指南》中的 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html。

使用加密以保护敏感数据

在 AWS Proton 中，所有客户数据默认使用 AWS Proton 拥有的密钥进行加密。

作为平台团队的成员，您可以向 AWS Proton 提供客户托管密钥以加密和保护您的敏感数据。静态加密 S3 存储桶中的敏感数据。有关更多信息，请参阅AWS Proton 中的数据保护。

使用 AWS CloudTrail 查看和记录 API 调用

AWS CloudTrail 跟踪在您的 AWS 账户 中进行 API 调用的任何人。每次任何人使用 AWS Proton API、AWS Proton 控制台或 AWS ProtonAWS CLI 命令时，都会记录 API 调用。启用日志记录并指定用于存储日志的 Amazon S3 存储桶。这样，如果需要，您可以审核谁在您的账户中进行了 AWS Proton 调用。有关更多信息，请参阅AWS Proton 中的日志记录和监控。

跨服务混淆代理问题防范

混淆代理问题是一个安全性问题，即不具有操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在 AWS 中，跨服务模拟可能会导致混淆代理问题。一个服务（呼叫服务）调用另一项服务（所谓的服务)时，可能会发生跨服务模拟。可以操纵调用服务，使用其权限以在其他情况下该服务不应有访问权限的方式对另一个客户的资源进行操作。为防止这种情况，AWS 提供可帮助您保护所有服务的数据的工具，而这些服务中的服务主体有权限访问账户中的资源。

我们建议使用资源策略中的 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键，限制 AWS Proton 为另一项服务提供的资源访问权限。如果 `aws:SourceArn` 值不包含账户 ID，例如 Amazon S3 存储桶 ARN，您必须使用两个全局条件上下文密钥来限制权限。如果同时使用全局条件上下文密钥和包含账户 ID 的 `aws:SourceArn` 值，则 `aws:SourceAccount` 值和 `aws:SourceArn` 值中的账户在同一策略语句中使用时，必须使用相同的账户 ID。如果您只希望将一个资源与跨服务访问相关联，请使用 `aws:SourceArn`。如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用 `aws:SourceAccount`。

`aws:SourceArn` 的值必须是 AWS Proton 存储的资源。

防范混淆代理问题最有效的方法是使用 `aws:SourceArn` 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符 (*) 的 `aws:SourceArn` 全局上下文条件键。例如，`arn:aws::proton:*:123456789012:environment/*`。

以下示例演示如何使用 AWS Proton 中的 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键来防范混淆代理问题。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "ExampleProtonConfusedDeputyPreventionPolicy",
        "Effect": "Allow",
        "Principal": {"Service": "proton.amazonaws.com"},
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "123456789012"
            },
            "ArnLike": {
                "aws:SourceArn": "arn:aws::proton:*:123456789012:environment/*"
            }
        }
    }
}
```

CodeBuild 预置自定义 Amazon VPC 支持

AWS Proton CodeBuild 预置在位于 AWS Proton 环境账户的 CodeBuild 项目中执行客户提供的任意 CLI 命令。这些命令通常使用基础设施即代码 (IaC) 工具（例如 CDK）管理资源。如果您在 Amazon VPC 中具有资源，CodeBuild 可能无法访问这些资源。为了解决该问题，CodeBuild 支持在特定 Amazon VPC 中运行。一些示例使用案例包括：

- 从自托管的内部构件存储库中检索依赖项，例如适用于 Python 的 PyPI、适用于 Java 的 Maven 和适用于 Node.js 的 npm。
- CodeBuild 需要访问特定 Amazon VPC 中的 Jenkins 服务器以注册管道。
- 访问配置为仅允许通过 Amazon VPC 终端节点访问的 Amazon S3 存储桶中的对象。
- 针对在私有子网上隔离的 Amazon RDS 数据库中的数据，从您的构建中运行集成测试。

有关更多信息，请参阅 CodeBuild 和 VPC 文档。

如果您希望 CodeBuild 预置在自定义 VPC 中运行，AWS Proton 提供了一种简单的解决方案。首先，您必须将 VPC ID、子网和安全组添加到环境模板中。接下来，您将这些值输入到环境规范中。这导致创建一个针对给定 VPC 的 CodeBuild 项目。

更新环境模板

架构

需要将 VPC ID、子网和安全组添加到模板架构中，以便将其包含在环境规范中。

一个示例 `schema.yaml`：

```
schema:
  format:
    openapi: "3.0.0"
  environment_input_type: "EnvironmentInputType"
  types:
    EnvironmentInputType:
      type: object
      properties:
        codebuild_vpc_id:
          type: string
        codebuild_subnets:
          type: array
          items:
            type: string
        codebuild_security_groups:
          type: array
          items:
            type: string
```

这添加了清单使用的三个新属性：

- `codebuild_vpc_id`
- `codebuild_subnets`
- `codebuild_security_groups`

清单

为了在 CodeBuild 中配置 Amazon VPC 设置，在模板清单中提供了一个名为 `project_properties` 的可选属性。`project_properties` 内容将添加到创建 CodeBuild 项目的 AWS CloudFormation 堆栈中。这样，不仅可以添加 Amazon VPC AWS CloudFormation 属性，而且还可以添加任何支持的 CodeBuild CloudFormation 属性，例如构建超时。为 `proton-inputs.json` 提供的相同数据也可以作为 `project_properties` 的值。

将该部分添加到 manifest.yaml 中：

```
project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

生成的 manifest.yaml 可能如下所示：

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
        provision:
          - npm install
          - npm run build
          - npm run cdk bootstrap
          - npm run cdk deploy -- --require-approval never
        deprovision:
          - npm install
          - npm run build
          - npm run cdk destroy -- --force
        project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

创建环境

在您使用启用了 VPC 的 CodeBuild 预置模板创建环境时，您必须提供 Amazon VPC ID、子网和安全组。

要获取您的区域中的所有 Amazon VPC ID 的列表，请运行以下命令：

```
aws ec2 describe-vpcs
```

要获取所有子网 ID 的列表，请运行：

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=vpc-id"
```

> **Important**
>
> 仅包括私有子网。如果您提供公有子网，CodeBuild 将失败。公有子网具有到互联网网关的默认路由，而私有子网没有。

运行以下命令以获取安全组 ID。也可以通过 AWS Management Console 获取这些 ID：

```
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=vpc-id"
```

这些值将类似于：

```
vpc-id: vpc-045ch35y28dec3a05
subnets:
  - subnet-04029a82e6ae46968
  - subnet-0f500a9294fc5f26a
 security-groups:
  - sg-03bc4c4ce32d67e8d
```

确保 CodeBuild 权限

Amazon VPC 支持需要具有某些权限，例如，能够创建弹性网络接口。

如果在控制台中创建环境，请在执行环境创建向导期间输入这些值。如果要以编程方式创建环境，您的 `spec.yaml` 如下所示：

```
proton: EnvironmentSpec

spec:
  codebuild_vpc_id: vpc-045ch35y28dec3a05
  codebuild_subnets:
    - subnet-04029a82e6ae46968
    - subnet-0f500a9294fc5f26a
  codebuild_security_groups:
    - sg-03bc4c4ce32d67e8d
```

AWS Proton 资源和标记

分配了 Amazon 资源名称 (ARN) 的 AWS Proton 资源包括环境模板及其主要和次要版本、服务模板及其主要和次要版本、环境、服务、服务实例、组件和存储库。您可以标记这些资源以帮助组织和识别它们。您可使用标签，按用途、所有者、环境或其他标准对资源进行分类。有关更多信息，请参阅 标记策略。要跟踪和管理您的 AWS Proton 资源，您可以使用以下几节中所述的标记功能。

AWS 标记

可以将自己的元数据以标签形式分配给 AWS 资源。每个标签包含客户定义的键和可选的值。标签可帮助您管理、识别、组织、搜索和筛选资源。

> **Important**
>
> 请勿在标签中添加个人身份信息（PII）或其他机密或敏感信息。标签可供许多 AWS 服务 访问，包括计费。标签不适合用于私有或敏感数据。

每个 标签具有两个部分。

- 标签键（例如，`CostCenter`、`Environment` 或 `Project`）。标签键区分大小写。
- 标签值（可选）（例如 `111122223333` 或 `Production`）。与标签键一样，标签值区分大小写。

以下基本命名和用法要求适用于标签。

- 每个资源最多可以有 50 个用户创建的标签。

> **Note**
>
> 以 `aws:` 前缀开头的系统创建标签是为 AWS 使用而保留的，并且不计入该限制。您无法编辑或删除以 `aws:` 前缀开头的标签。

- 对于每个资源，每个标签键都必须是唯一的，每个标签键只能有一个值。
- 标签键必须包含 1 到 128 个 Unicode 字符，并且以 UTF-8 格式表示。
- 标签值必须最少为 1 个 Unicode 字符，最多为 256 个 Unicode 字符，采用 UTF-8 格式。
- 允许在标签中使用的字符包括以 UTF-8 表示的字母、数字和空格以及以下字符：* _ . : / = + - @。

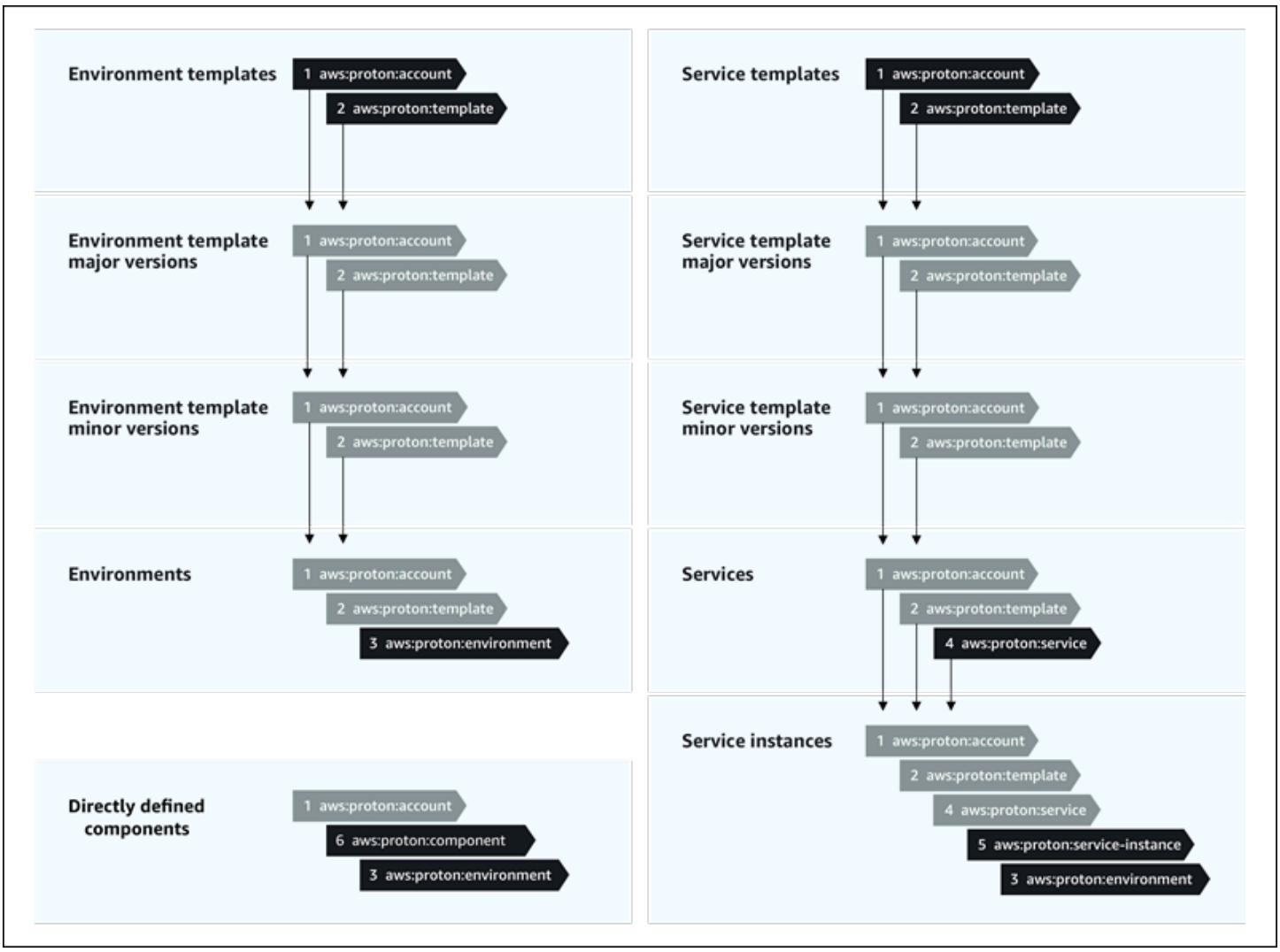
AWS Proton 标记

对于 AWS Proton，可以使用您创建的标签以及 AWS Proton 自动为您生成的标签。

AWS Proton AWS 托管标签

在您创建 AWS Proton 资源时，AWS Proton 自动为新资源生成 AWS 托管标签，如下图所示。AWS 托管标签稍后传播到基于新资源的其他 AWS Proton 资源。例如，环境模板中的托管标签传播到其版本，服务中的托管标签传播到其服务实例。

> **Note**
>
> 不会 为环境账户连接生成 AWS 托管标签。有关更多信息，请参阅the section called “账户连接”。



标签传播到预置的资源

如果预置的资源（例如服务和环境模板中定义的资源）支持 AWS 标记，AWS 托管标签将作为客户托管标签传播到预置的资源。这些标签不会传播到不支持 AWS 标记的预置资源。

AWS Proton 按 AWS Proton 账户、注册的模板和部署的环境以及服务和服务实例将标签应用于您的资源，如下表中所述。您可以使用 AWS 托管标签查看和管理您的 AWS Proton 资源，但无法修改这些资源。

AWS 托管标签键	传播的客户托管键	说明
`aws:proton:account`	`proton:account`	创建和部署 AWS Proton 资源的 AWS 账户。

AWS 托管标签键	传播的客户托管键	说明
aws:proton:template	proton:template	选定的模板的 ARN。
aws:proton:environment	proton:environment	选定的环境的 ARN。
aws:proton:service	proton:service	选定的服务的 ARN。
aws:proton:service-instance	proton:service-instance	选定的服务实例的 ARN。
aws:proton:component	proton:component	选定的组件的 ARN。

以下是 AWS Proton 资源的 AWS 托管标签示例。

```
"aws:proton:template" = "arn:aws:proton:region-id:account-id:environment-template/env-template"
```

以下是应用于预置资源的客户托管标签示例，该标签是从 AWS 托管标签传播的。

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

对于 AWS 托管式预置，AWS Proton 将传播的标签直接应用于预置的资源。

对于自托管式预置，AWS Proton 将传播的标签与它在预置拉取请求 (PR) 中提交的渲染 IaC 文件一起提供。标签是在字符串映射变量 proton_tags 中提供的。我们建议您在 Terraform 配置中引用该变量，以在 default_tags 中包含 AWS Proton 标签。这会将 AWS Proton 标签传播到所有预置的资源。

以下示例说明了环境 Terraform 模板中的这种标签传播方法。

以下是 proton_tags 变量定义：

proton.environment.variables.tf：

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

```
  })
}

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

以下是如何将标签值分配给该变量：

proton.auto.tfvars.json：

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

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

以下是如何将 AWS Proton 标签添加到 Terraform 配置中，以便将它们添加到预置的资源中：

```
# Configure the AWS Provider
provider "aws" {
  region = var.aws_region
  default_tags {
    tags = var.proton_tags
  }
}
```

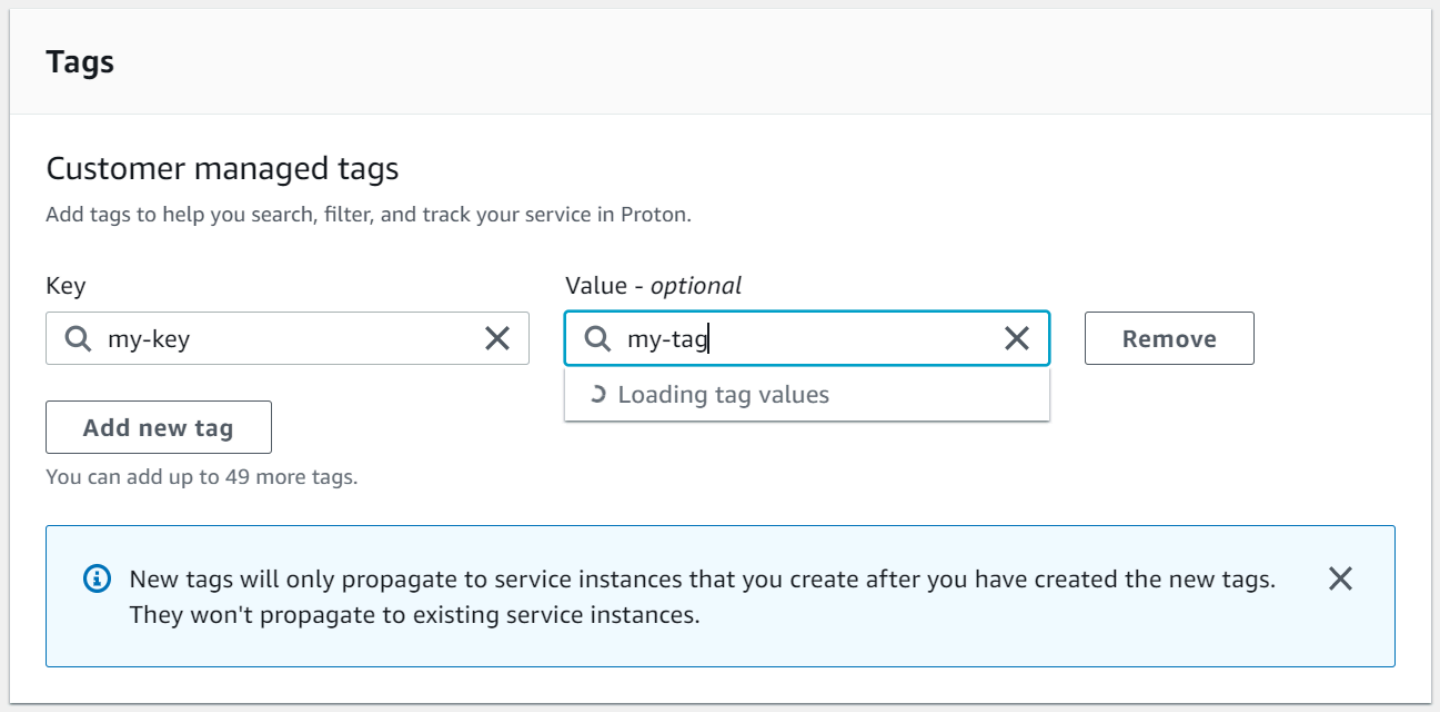
客户托管标签

每个 AWS Proton 资源的最大配额为 50 个客户托管标签。客户托管标签传播到子 AWS Proton 资源的方式与 AWS 托管标签相同，但它们不会传播到现有的 AWS Proton 资源或预置的资源。如果您将新标

签应用于具有现有子资源的 AWS Proton 资源，并且希望使用新标签标记现有的子资源，则需要使用控制台或 AWS CLI 手动标记每个现有的子资源。

使用控制台和 CLI 创建标签

在您使用控制台创建 AWS Proton 资源时，您有机会在创建过程的第一页或第二页上创建客户托管标签，如以下控制台快照中所示。选择添加新标签，输入键和值并继续。



在使用 AWS Proton 控制台创建新资源后，您可以从详细信息页面中查看它的 AWS 托管标签和客户托管标签列表。

创建或编辑标签

1. 在 AWS Proton 控制台中，打开 AWS Proton 资源详细信息页面，您可以在其中看到标签列表。
2. 选择管理标签。
3. 在管理标签页面中，您可以查看、创建、删除和编辑标签。您无法修改顶部列出的 AWS 托管标签。不过，您可以使用编辑字段添加和修改客户托管标签，这些字段在 AWS 托管标签后面列出。

   选择添加新标签以创建新标签。
4. 输入新标签的键和值。
5. 要编辑标签，请在选定键的标签值字段中输入一个值。
6. 要删除标签，请为选定的标签选择删除。

7. 在完成更改后，选择保存更改。

使用 AWS Proton AWS CLI 创建标签

您可以使用 AWS Proton AWS CLI 查看、创建、删除和编辑标签。

您可以创建或编辑资源的标签，如以下示例中所示。

```
$ 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"}]'
```

您可以删除资源的标签，如以下示例中所示。

```
$ aws proton untag-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice" \
    --tag-keys '["mykey1","mykey2"]'
```

您可以列出资源的标签，如最后一个示例中所示。

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice"
```

排除 AWS Proton 的故障

了解如何解决 AWS Proton 问题。

主题

引用 AWS CloudFormation 动态参数的部署错误

如果您看到引用 CloudFormation 动态变量的部署错误，请验证它们是否经过 Jinja 转义。这些错误可能是由 Jinja 错误地解释动态变量引起的。CloudFormation 动态参数语法与您在 AWS Proton 参数中使用的 Jinja 语法非常相似。

示例 CloudFormation 动态变量语法：

`'{{resolve:secretsmanager:MySecret:SecretString:password:EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE}}'`.

示例 AWS Proton 参数 Jinja 语法：

`'{{ service_instance.environment.outputs.env-outputs }}'`.

为了避免这些解释错误，Jinja 转义了您的 CloudFormation 动态参数，如以下示例中所示。

该示例来自于《AWS CloudFormation 用户指南》。可以使用 AWS Secrets Manager secret-name 和 json-key 分段检索密钥中存储的登录凭证。

```
MyRDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: 'MyRDSInstance'
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
      MasterUserPassword:
 '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
```

要转义 CloudFormation 动态参数，您可以使用两种不同的方法：

- 将一个块放在 {% raw %} and {% endraw %} 之间：

```
'{% raw %}'
MyRDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: 'MyRDSInstance'
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
      MasterUserPassword:
 '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
'{% endraw %}'
```

- 将一个参数放在 "{{ }}" 之间：

```
MyRDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: 'MyRDSInstance'
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername:
 "{{ '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}' }}"
      MasterUserPassword:
 "{{ '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}' }}"
```

有关信息，请参阅 Jinja escaping。

AWS Proton 配额

下表列出了 AWS Proton 配额。所有值是按 AWS 账户和支持的 AWS 区域计算的。

资源配额	默认限制	可调整?
模板捆绑包的最大大小	10 MB	✗ 否
模板清单文件的最大大小	2MB	✗ 否
模板架构文件的最大大小	2MB	✗ 否
每个模板文件的最大大小	2MB	✗ 否
每个模板名称的最大长度	100 个字符	✗ 否
每个捆绑包的最大 CloudFormation 模板文件数	1	✗ 否
每个账户、服务和环境模板组合的最大注册模板数	1000	✓ 是
每个模板注册的最大模板版本数	1000	✓ 是
每个 CodeBuild 预置捆绑包的最大文件数	500	✗ 否
每个账户的最大环境数	1000	✓ 是
每个账户的最大服务数	1000	✓ 是
每个服务的最大服务实例数	20	✓ 是
每个账户的最大组件数量	1000	✓ 是
每个环境账户的最大环境账户连接数	1000	✓ 是

文档历史记录

下表介绍了对与最新版本的 AWS Proton 和客户反馈相关的文档进行的重要更改。如需对此文档更新的通知，您可以订阅 RSS 源。

- API 版本：2020-07-20

变更	说明	日期
托管式策略更新	更新了 AWSProtonCodeBuildProvisioningServiceRolePolicy 策略。	2023 年 5 月 12 日
服务同步配置。	AWS Proton 添加了对服务同步配置的支持。	2023 年 3 月 31 日
CodeBuild	AWS Proton 添加了对 CodeBuild 预置的支持。	2022 年 11 月 16 日
托管式策略更新	添加了 AWSProtonCodeBuildProvisioningBasicAccess 策略，以向 CodeBuild 提供为 AWS Proton CodeBuild 预置运行构建所需的权限。	2022 年 11 月 11 日
Terraform 标签传播	在标记章节中添加了 Terraform 标签传播。	2022 年 9 月 16 日
API 迁移指南	删除了 GA 前 API 迁移指南。	2022 年 8 月 12 日
AWS Proton 对象	添加了有关 AWS Proton 对象以及与其他 AWS 对象和第三方对象的关系的主题。请参阅 AWS Proton 对象。	2022 年 7 月 29 日

链接的存储库澄清	在整个指南中澄清了链接（注册）的存储库的用途及其用法。	2022 年 7 月 18 日
指南合并	将两个单独的管理员指南和用户指南合并为一个指南，即《AWS Proton 用户指南》。	2022 年 6 月 30 日
托管式策略更新	更新了托管策略以提供对新 AWS Proton API 操作的访问，并修复某些 AWS Proton 控制台操作的权限问题。请参阅适用于 AWS Proton 的 AWS 托管策略。	2022 年 6 月 20 日
CLI 入门	通过使用新模板库的新教程更新了 AWS CLI 入门。	2022 年 6 月 14 日
直接定义的组件	添加了组件章节，并在整个指南中进行了相关修改。	2022 年 6 月 1 日
AWS Proton 模板库	添加了 AWS Proton 模板库主题。	2022 年 5 月 6 日
Terraform 正式发行版 (GA)	将拉取请求预置 重命名为自托管式预置。添加了预置方法主题。	2022 年 3 月 23 日
存储库标记	添加了对存储库资源标记的支持。请参阅创建存储库的链接。	2022 年 3 月 23 日
文档更新	添加了环境账户连接标记。	2021 年 11 月 26 日

模板同步和 Terraform 预览	在正式发行版中添加了具有模板同步功能的自动化模板版本控制，并在预览版中添加了使用 Terraform 进行拉取请求预置功能。重新添加了 API 迁移指南。	2021 年 11 月 24 日
文档更新	添加了 EventBridge 教程、入门工作流、AWS Proton 的工作方式以及模板捆绑包部分改进内容。	2021 年 9 月 17 日
AWS Proton 控制台帮助面板发行版	在控制台中添加了帮助面板。控制台模板版本删除不再删除较低的版本。已删除 API 迁移指南。	2021 年 9 月 8 日
AWS Proton 正式发行版 (GA)	添加了跨账户环境、EventBridge 监控、IAM 条件键、幂等性支持和增加的配额。	2021 年 6 月 9 日
添加和删除服务的服务实例，并针对具有 AWS Proton 的环境使用现有的外部基础设施	通过使用该公开预览版包含的更新，您可以在服务中添加和删除服务实例，在 AWS Proton 环境中使用现有的外部基础设施以及取消环境、服务实例和管道部署。AWS Proton 现在支持 PrivateLink。添加了额外的删除验证，以防止在资源使用某个次要版本时错误地将其删除。	2021 年 4 月 27 日
AWS Proton 中的标记	公开预览版 2 包括 AWS Proton 标记以及在不使用服务管道的情况下启动服务的功能。	2021 年 3 月 5 日

初始版本	公开预览版现已在选定的区域中推出。	2020 年 12 月 1 日

AWS 术语表

有关最新的 AWS 术语，请参阅《AWS 词汇表参考》中的 AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。