aws

开发人员指南

截止日期云

截止日期云: 开发人员指南

Table of Contents

什么是 AWS 截止日期云？

AWS Deadline Cloud 是一项完全托管的 AWS 服务，可让您在几分钟内启动并运行可扩展的处理场。它提供了一个管理控制台，用于管理用户、服务器场、用于调度作业的队列以及负责处理的工作人员队伍。

本开发人员指南适用于各种用例中的管道、工具和应用程序开发人员，包括以下用例：

- 管道开发人员和技术主管可以将 Deadline Cloud APIs 和功能集成到他们的自定义制作管道中。
- 独立软件供应商可以将 Deadline Cloud 集成到他们的应用程序中，使数字内容创作艺术家和用户能够从其工作站无缝提交 Deadline Cloud 渲染作业。
- 网络和基于云的服务开发人员可以将Deadline Cloud渲染集成到其平台中，从而使客户能够提供虚拟查看产品的资产。

我们提供的工具使您能够直接处理流程中的任何步骤：

- 可以直接使用或通过脚本使用的命令行界面。
- AWS 适用于 11 种流行编程语言的 SDK。
- 一个基于 REST 的 Web 界面，你可以从应用程序中调用。

您也可以在自定义应用程序 AWS 服务 中使用其他。例如，你可以使用：

- AWS CloudFormation自动创建和删除服务器场、队列和队列。
- 亚马逊 CloudWatch将收集就业指标。
- Amazon 简单存储服务，用于存储和管理数字资产和工作产出。
- AWS IAM Identity Center管理农场的用户和群组。

打开 Job 描述

Deadline Cloud 使用 OpenJD 职位描述 (OpenJD) 规范来指定作业的详细信息。OpenJD 的开发是为了定义解决方案之间可移植的作业。你可以用它来定义一个作业，该作业是一组在工作主机上运行的命令。

你可以使用 Deadline Cloud 提供的提交者创建 OpenJD 作业模板，也可以使用任何你想要创建模板的工具。创建模板后，将其发送到 Deadline Cloud。如果您使用提交者，它会负责发送模板。如果您以

其他方式创建了模板，则可以调用 Deadline Cloud 命令行操作，也可以使用其中一个 AWS SDKs 来发送作业。无论哪种方式，Deadline Cloud 都会将任务添加到指定的队列中并安排工作。

截止日期云的概念和术语

为了帮助您开始使用 De AWS adline Cloud，本主题解释了其一些关键概念和术语。

农场资源

此图显示了 Deadline Cloud 场资源是如何协同工作的。

服务器农场

服务器场包含与提交和运行作业相关的所有其他资源。农场相互独立，因此可用于分离生产环境。

队列

队列中包含用于在关联舰队上调度的作业。用户可以向队列提交作业，并在队列中管理其优先级和状态。队列必须与具有队列队列关联的队列关联才能运行其作业，并且队列可以与多个队列相关联。

实例集

队列包含用于运行作业的计算容量。舰队可以是服务管理的，也可以是客户管理的。服务管理的队列在 Deadline Cloud 中运行，包括自动扩展、许可和软件访问等内置功能。客户管理的队列在您自己的计算资源（例如 Amazon EC2 实例或本地服务器）上运行。

Budget

预算为您的工作活动设置支出阈值，并允许您在达到阈值时采取行动，例如停止工作安排。

队列环境

队列环境定义了在每个工作器上运行的脚本，用于设置或关闭工作负载环境。它们对于设置环境变量、安装软件和配置资产存储非常有用。

存储配置文件

存储配置文件是一组主机和工作站的配置，它告诉数据在文件系统上的位置。Deadline Cloud 使用存储配置文件来映射在不同配置的主机上运行作业（例如从中提交Windows并在其上运行的作业）上的路径Linux。

限制

限制允许您跟踪共享资源（例如浮动许可证）的使用情况，并控制它们在任务之间的分配方式。限制与具有队列限制关联的队列相关联。

监控

监控器配置 Deadline Cloud 监控器 Web 应用程序的 URL，允许最终用户监控和管理作业。它可以在浏览器中或通过 Deadline Cloud 监视器桌面应用程序进行访问。

Job 执行资源

此图显示了 Deadline Cloud 作业资源是如何协同工作的。

作业

作业是用户提交给 Deadline Cloud 的一组工作，以便在可用的工作人员上安排和运行。作业可以渲染 3D 场景或运行模拟。作业由可重复使用的作业模板创建，这些模板定义了运行时环境和流程，以及特定于作业的参数。作业包含定义要执行的工作的步骤和任务，并且可以对它们进行优先级、最大工作人员数量和重试设置进行配置。

作业优先级

任务优先级是 Deadline Cloud 在队列中处理任务的大致顺序。您可以将作业优先级设置在 1 到 100 之间，数字优先级较高的作业通常会先处理。优先级相同的任务按收到的顺序处理。

作业属性

Job 属性是您在提交渲染作业时定义的设置。一些示例包括帧范围、输出路径、作业附件、可渲染摄像机等。属性因提交渲染的 DCC 而异。

步骤

步骤是作业的一部分，它为运行许多任务提供了模板，这些任务除任务参数值之外完全相同。步骤可能依赖于其他步骤，从而允许您创建具有顺序或并行执行路径的复杂工作流程。在渲染作业中，步骤通常定义用于渲染帧的命令并使用帧号作为任务参数。

Task

任务是 Deadline Cloud 中最小的工作单位。任务是步骤的一部分，由工作人员执行，代表需要作为作业的一部分执行的单个操作。可以用特定的参数配置任务，并根据工作人员的能力和可用性将其分配给他们。在渲染作业中，任务通常会渲染单帧。

工作线程

工人是车队的一部分，他们执行工作中的任务。可以为工作人员配置特定功能，例如 GPU 加速器、CPU 架构和操作系统。在服务管理的车队中，工作人员是在车队向外扩展和缩小规模时自动创建的。

实例

舰队使用实例获取 CPU 资源。实例是 Amazon EC2 性能实例。截止日期云使用按需实例和竞价实例。

按需实例

按需实例按秒计价，没有长期承诺，也不会中断。

竞价型实例

竞价型实例是非预留容量，您可以以折扣价使用，但可能会被按需请求中断。

等待并保存

Wait and Save 功能以较低的成本提供延迟作业调度，并且可以被按需请求和竞价请求中断。“等待并保存” 仅在 Deadline Cloud 服务管理的舰队中可用。

Wait and Save 用于在 De AWS adline Cloud 中管理视觉计算工作负载的执行。详情请参阅AWS服务条款。

会话

会话代表工作人员在工作中的工作顺序。在单个会话中，可能会为工作人员分配多个任务，这些任务一个接一个地运行。会话通常具有设置操作，这些操作可以在运行任务操作之前配置环境和加载资产。

会话操作

会话操作表示会话期间执行的特定操作，例如设置环境、运行任务和同步资产。

其他重要概念和术语

使用情况浏览器

使用情况浏览器是 Deadline Cloud 监控器的一项功能。它提供了对您的成本和使用量的近似估计。

预算经理

预算经理是 Deadline Cloud 监控器的一部分。使用预算管理器来创建和管理预算。您还可以使用它来限制活动以保持在预算范围内。

截止日期云客户端库

开源客户端库包括用于管理 Deadline Cloud 的命令行界面和库。功能包括根据 Open Job Description 规范向 Deadline Cloud 提交工作捆绑包、下载作业附件输出以及使用命令行界面 (CLI) 监控您的农场。

数字内容创作应用程序 (DCC)

数字内容创作应用程序 (DCCs) 是您在其中创建数字内容的第三方产品。Deadline Cloud 内置了与 Autodesk Maya、Blender 和 Maxon Cinema 4D DCCs 等许多系统的集成，允许你从 DCC 内部提交作业，并使用预先配置的软件和许可在服务管理的队列上进行渲染。

Job 附件

Job 附件是 Deadline Cloud 的一项功能，您可以将其作为工作的一部分上传和下载资源，例如纹理、3D 模型和灯光装备。Job 附件存储在 Amazon S3 中，无需共享网络存储。

作业模板

作业模板定义运行时环境以及作为 Deadline Cloud 作业的一部分运行的所有进程。

截止日期云提交者

Deadline Cloud 提交器是 DCC 的插件，它允许用户轻松地从 DCC 内部提交作业。

许可证端点

许可证端点使得 Deadline Cloud 基于使用量的第三方产品许可在您的 VPC 内可用。此模式按使用量付费，按使用的小时数和分钟数向您收费。许可证端点未连接到服务器场，可以独立使用。

标记

标签是您可以分配给 AWS 资源的标签。每个标签都包含您所定义的一个键和可选值。使用标签，您可以按不同的方式对 AWS 资源进行分类，例如按用途、所有者或环境进行分类。

基于使用的许可 (UBL)

基于使用量的许可 (UBL) 是一种按需许可模式，适用于部分第三方产品。此模式按使用量付费，您需要按使用的小时数和分钟数付费。

截止日期云架构指南

本主题为使用 Deadline Cloud 为您的工作负载设计和构建可靠、安全、高效且经济实惠的渲染农场提供了指导和最佳实践。使用本指导可以帮助您构建稳定、高效的工作负载，从而使您能够专注于创新、降低成本并改善客户体验。

本内容面向首席技术官 (CTOs)、架构师、开发人员和运营团队成员。

end-to-end渲染工作流程需要在流程的多个层面提供解决方案，例如任务生成、资产访问和作业监控。Deadline Cloud 为渲染过程的每一层提供了多种解决方案。通过从每个图层的 Deadline Cloud 选项中进行选择，您可以设计与您的用例相匹配的工作流程。

对于每一层，你都需要决定哪种方法最适合你的用例。这些不是严格的场景定义，也不是使用 Deadline Cloud 的唯一方法。相反，这些是一组高级概念，可帮助您了解Deadline Cloud如何适应您的业务或工作流程。您可以将 Deadline Cloud 工作负载分为以下几层：任务来源、作业提交、应用程序管理、应用程序许可、资产访问、输出管理和工作器基础设施管理。

通常，您可以将一个层中的 mix-and-match任何场景与另一个图层中的任何其他场景一起使用，但下面指定的特定组合除外。


Job Source
Interactive Workflow
Automated Workflow
Job Submission
Integrated Submitter
Custom Job Definition
Application Management
Conda Application Management
Custom Application Management
Application Licensing
Deadline Cloud UBL
Custom Licensing
Asset Access
Job Attachments
Custom Storage
Job Monitoring
Deadline Cloud monitor
Custom Monitor Application
Automated Monitoring Solution
Worker Infrastructure

Job 来源

任务源是接入点，新作业将进入系统，由 Deadline Cloud 呈现。总体而言，有两个主要的工作来源：人际交互和自动化计算机系统。

交互式工作流程

在这种情况下，艺术家或其他创作角色是Deadline Cloud农场中要处理的作品的主要生成者。通常，这些作业的输出是大型项目或团队的主要产物。他们使用诸如行业标准数字内容创作 (DCC) 工具之类的软件来执行工作。他们正在手动向 Deadline Cloud 场提交作业，然后查看输出以进行审查。工作站本身不是由管理的 AWS。

在大多数情况下，这些艺术家在工作负载应用程序和监控层中使用 Deadline Cloud 集成提交者和 Deadline Cloud 监视器。

自动化工作流程

在这种情况下，客户拥有的编程系统是 Deadline Cloud 场中的主要工作生成器。这可能是零售渠道中的资产生成，例如通过三维模型或扫描生成的转盘视频。这可以是自动合成用于体育的广播图形和球员卡。此场景的主题是，个人不是手动将每份作业提交到 Deadline Cloud，而是作为大型系统的一部分生成作业。

对于自动化作业，使用 Deadline Cloud 集成提交者和 Deadline Cloud 监控器的情况并不常见。通常，任务定义将由您编写的自定义应用程序开发，任务输出将自动流入数字资产管理 (DAM) 系统或媒体资产管理 (MAM) 系统进行批准和分发。

Job 提交

作业使用OpenJobDescription模板提交到 Deadline Cloud。 OpenJobDescription 是一种灵活的开放规范，用于定义可在不同调度系统部署之间移植的批处理作业。作业定义文件描述了作业的参数、作业的步骤、如何根据作业输入对步骤进行参数化，以及将在工作器上运行以执行处理的实际脚本。Workload Submition 的概念是如何创建这些任务定义、由谁创建它们以及如何提交这些定义。

将提交器与 DCC 集成

Deadline Cloud 集成提交器是一款将 Deadline Cloud 与行业标准 DCC 或软件包结合在一起的软件。集成的提交者决定如何将渲染、合成或其他工作负载的数据和配置转换为作业模板，Deadline Cloud 可以理解。许多集成提交者是由Deadline Cloud团队或软件包的创建者创建和维护的，但是如果所需应

用程序尚不存在集成提交者，则可以创建和维护自己的提交者。Deadline Cloud 团队只支持有限的一组。 DCCs

交互式工作流程通常涉及集成的提交者，但并非总是如此。对于模板化的自动化工作流程，常见的工作流程是美工人员在自己的 DCC 中设置模板作业，然后一次性导出作业捆绑包。此任务包定义了如何以参数化的方式在 Deadline Cloud 上运行该特定类型的作业。此任务包可以集成到自动化工作流程场景中以实现自动化。

自定义任务定义

对于自定义应用程序和工作流程，可以完全控制如何创建这些任务定义并将其提交到 Deadline Cloud。例如，电子商务网站可能会要求卖家上传他们销售的物品的 3D 模型。上传后，电子商务平台可以动态生成任务定义以提交给 Deadline Cloud，从而使用常用光照在公共背景上自动生成转盘动画，以匹配网站上可用的其他 3D 对象。在电子商务平台的开发过程中，软件开发人员将创建任务定义，使用卖家最终提供的参数将其嵌入到电子商务平台中，然后对平台进行编码，以便在平台产品上传工作流程中提交此作业。

Deadline Cloud 在 github 的示例存储库中提供了许多示例作业定义。

应用程序管理

将作业提交到 Deadline Cloud 并分配给工作人员后，将在该工作人员上执行作业定义中的脚本。在大多数情况下，此脚本将调用应用程序来执行实际处理，例如渲染器、合成、编码、过滤或任何其他计算密集型任务。应用程序管理的概念是确保员工可以使用所需软件的必要版本。

您可以使用自己喜欢的任何软件包管理系统来管理应用程序，但是 Deadline Cloud 提供了许多工具来轻松启用 conda 软件包。Conda 是一个开源、跨平台、与语言无关的软件包管理器和环境管理系统。

面向服务管理车队 (SMF) 的 Deadline 云托管 conda 频道

使用服务托管队列时，系统会自动设置和配置由 Deadline Cloud 管理的 conda 频道，供您的作业使用。Deadline Cloud 服务在这个 conda 频道中提供了许多合作伙伴 DCC 应用程序和渲染。有关更多信息，请参阅 D eadline Cloud 用户指南中的创建队列环境。Deadline Cloud 服务会自动更新这些软件包，无需您进行维护。此 conda 频道仅在使用服务管理车队时可用，而在使用客户管理的车队时不可用。

自管 conda 频道

如果您无法使用 Deadline Cloud 管理的 conda 频道，则必须确定如何在 Deadline Cloud 队列上安装、修补和以其他方式管理应用程序。一种选择是创建一个由您设置和维护的 conda 频道。这将与

Deadline Cloud 管理的 conda 频道进行最密切的互操作。例如，你可以使用 Deadline Cloud 管理的 conda 频道中的 DCC，但要自带包含特定 DCC 插件的软件包。有关此过程的更多信息，请参阅使用 S3 创建 conda 频道。

自定义应用程序管理

对于应用程序管理，Deadline Cloud 的要求是，在工作器上执行作业脚本时，应用程序必须在 PATH 中可用。

如果您已经构建和维护 Rez 软件包，则可以使用队列环境从 Rez 存储库安装应用程序。可以在 De AWS adline Cloud GitHub 组织上找到队列环境示例。

如果您已经在客户管理的队列中使用长期使用的工作人员或系统映像管理应用程序，则无需队列环境即可进行应用程序管理。确保应用程序出现在作业用户的路径上并提交作业。

Application licensing

Deadline Cloud 上通常运行的许多工作负载都需要软件供应商的软件许可。这些应用程序通常按席位、每个 CPU 或每台主机进行许可。您有责任确保您在Deadline Cloud上使用第三方软件时遵守第三方许可协议。如果您使用的是开源软件、自定义软件或其他免许可证软件，则无需配置此层。请记住，Deadline Cloud 仅支持渲染许可，不支持工作站许可。

服务托管车队和基于使用量的许可

使用 Deadline Cloud 服务托管队列时，系统会自动为支持的软件配置基于使用量的许可 (UBL)。在服务管理队列上运行的作业会自动为支持的应用程序设置环境变量，以指导它们使用 Deadline Cloud 许可证服务器。使用 Deadline Cloud UBL 时，您只需按使用许可应用程序的小时数付费。

客户管理的车队和基于使用量的许可

不使用服务托管队列时，Deadline Cloud 基于使用量的许可 (UBL) 也可用。在这种情况下，您将设置 Deadline Cloud 许可端点，这些端点在您选定的 VPC 子网中提供 IP 地址，这些子网提供对 Deadline Cloud 许可服务器的访问权限。在工作人员上配置适当的软件特定环境变量并配置从工作人员到这些许可证端点 IP 地址的网络连接后，工作人员就可以签出和签入受支持软件的许可证。您按小时支付的许可证费用与在服务托管车队中使用 UBL 时相同。

自定义许可

您可能使用不受 Deadline Cloud UBL 支持的应用程序，或者您之前可能有仍然有效的许可证。在这种情况下，您负责配置从您的员工（客户或服务管理）到许可证服务器的网络路径。有关自定义许可的更多信息，请参阅Connect 将服务管理的车队连接到自定义许可服务器。

资产访问权限

将作业提交给工作人员并配置应用程序后，必须将该工作人员配置为访问该作业所需的资产数据。这可能是 3D 数据、纹理数据、动画数据、视频帧或工作中使用的任何其他类型的数据。

首先要考虑您的数据当前存储在哪里。这可能位于工作站硬盘、用户协作工具、源代码控制、本地或云端的共享文件系统、Amazon S3 或任意数量的其他位置。

接下来，考虑工作人员访问这些数据所需的条件。这些数据是否仅在您的公司网络上可用？访问数据需要什么身份或凭证？数据源是否按预期处理作业的工作人员数量进行扩展，以支持作业？

Job 附件

最简单的资产访问机制是 Deadline Cloud 作业附件。使用任务附件提交任务时，该任务所需的数据将与一份清单文件一起上传到 Amazon S3 存储桶，其中指定任务需要哪些文件。使用作业附件，无需复杂的联网或共享存储设置。文件仅上传一次，因此后续上传的完成速度更快。工作人员完成任务处理后，输出数据将上传到 Amazon S3，以便美工或其他客户可以下载。Job 附件适用于任何规模的车队，并且可以简单快速地装载和使用。

Job 附件并不是所有情况的最佳工具。如果您的数据已开启 AWS，则作业附件会添加您的数据的其他副本，包括相关的传输时间和存储成本。Job 附件要求作业能够在提交时完全指定所需的数据，以便可以上传数据。

要使用任务附件，您的 Deadline Cloud 队列必须具有关联的任务附件存储桶，并且必须使用队列角色来提供对该存储桶的访问权限。默认情况下，Deadline Cloud 集成提交者都支持作业附件。如果您没有使用 Deadline Cloud 集成提交器，则可以通过集成 De adline Cloud python 库，将作业附件与您的自定义软件一起使用。

自定义存储访问权限

如果您不使用作业附件，则有责任确保工作人员可以访问工作所需的数据。Deadline Cloud 提供了许多工具来支持这一点并保持作业的可移植性。如果您已经为艺术家和工作人员提供了共享的网络存储，您更喜欢使用外部服务（例如或出于其他原因） LucidLink，则可能需要使用自定义存储解决方案。

使用存储配置文件对工作站和工作主机上的文件系统进行建模。每个存储配置文件都描述了其中一个系统配置的操作系统和文件系统布局。使用存储配置文件，当使用 Windows 工作站的艺术家提交由Linux 工作人员处理的作业时，Deadline Cloud 可确保进行路径映射，以便工作人员可以访问您配置的数据存储。

使用 Deadline Cloud 服务管理的队列时，主机配置脚本和 VPC 资源端点使工作人员能够直接挂载和访问您的 VPC 中可用的共享存储或其他服务。

Job 监控和输出管理

成功完成提交到 Deadline Cloud 的作业后，个人或流程将下载任务输出，以便在 Deadline Cloud 之外的业务工作流程中使用。任务失败后，作业日志和监控信息有助于诊断问题。

截止日期云监控

Deadline Cloud 监控应用程序可在网络和桌面上使用。该解决方案最适合使用交互式工作流程的工作室，以便广泛 DCCs 使用作业附件进行存储。该监控器仅在使用 IAM 身份中心时支持您。IAM Identity Center 是一种员工身份产品，而不是消费者身份 (B2C) 解决方案，因此它不适用于许多 B2C 场景。

自定义监视器应用程序

如果您想自定义用户的监控体验，或者正在构建 B2C 产品，或者使用 Deadline Cloud 构建高度专业化的系统，则可以选择创建自定义监控应用程序。您可以使用 De AWS adline Cloud API 来创建此自定义应用程序，将整个工作流程的上下文与 Deadline Cloud 概念相结合。例如，您的B2C产品可能有自己的项目概念，用户可以设置该概念，并且您的应用程序可以在同一个界面中嵌套Deadline Cloud作业。

自动监控解决方案

在某些情况下，Deadline Cloud 不需要专用的监控应用程序。这种情况在自动化工作流程中很常见，在这种工作流程中，Deadline Cloud 用于自动渲染管道中的资产，例如体育或新闻的广播图片。在这种情况下，Deadline Cloud API 和 EventBridge 事件用于与外部媒体资产管理系统集成，以进行审批，并将数据转移到流程的下一步中。

工作人员基础设施管理

Deadline Cloud 队列是一组服务器（工作人员），它们能够处理提交到 Deadline Cloud 队列的作业，是任何 Deadline Cloud 场的核心基础架构。

服务管理车队

在服务管理的队列中，Deadline Cloud 对运行渲染农场的工作主机、操作系统、网络、补丁、自动缩放和其他因素负责。您可以指定所需的最小和最大工作人员数量，以及应用程序所需的系统规格，剩下的就交给 Deadline Cloud。服务管理舰队是唯一可以使用 Deadline Cloud 托管 conda 通道轻松管理行业 DCC 应用程序的车队选项。此外，Deadline Cloud UBL 会自动配置服务管理队列。只有使用服务托管队列才能使用成本更低、具有延迟容忍能力的工作负载的 Wait and Save 队列。

客户管理的车队

当您需要对工作人员主机及其环境进行更多控制时，您可以使用客户管理的队列。客户管理的车队最适合在本地使用 Deadline Cloud。要了解更多信息，请参阅创建和使用 Deadline Cloud 客户管理的车队。

架构示例

传统制作工作室

传统的制作工作室需要大量的计算、存储和网络基础架构，这些基础架构可以跨越多个物理位置，为渲染工作负载提供服务。每个软件包和供应商都有独特的硬件、软件、网络和许可要求，在解决版本控制、兼容性和资源冲突时必须满足这些要求。

通常对艺术家工作站、渲染节点、网络存储、许可证服务器、作业队列系统、监控工具和资产管理有单独的基础架构要求。工作室通常需要维护多个版本的 DCC 工具、渲染器、插件和自定义工具，同时管理整个渲染农场的复杂许可安排。考虑到开发、质量保证和制作环境，您的工作室基础设施会变得更加复杂。

使用服务托管选项的典型Deadline Cloud部署可通过以下方式解决或减少其中的许多挑战：

- 通过集成的 DCC 提交者提交交互式工作流程作业
- 通过 Deadline Cloud 托管 conda 渠道
- 为支持的软件自动配置基于使用情况的许可
- 通过工作附件进行资产管理
- 通过 Deadline 云监控应用程序进行监控
- 通过服务管理的队列管理基础架构

通过这种方法，美术师可以直接从他们熟悉的 DCC 工具向可扩展的云渲染农场提交作业，而无需管理复杂的基础架构。该服务自动处理软件部署、许可、数据传输和基础架构扩展。艺术家可以通过网页界面或桌面应用程序监控他们的工作，输出内容会自动存储在 Amazon S3 中，便于访问。

通过这种配置，工作室可以在几分钟内创建开发和制作环境，只需为他们使用的计算和许可付费，并且可以专注于创造性工作而不是基础设施管理。服务管理方法为采用云渲染提供了最快的途径，同时保持了艺术家熟悉的工作流程。


Interactive Workflow
Automated Workflow
Integrated Submitter
Custom Job Definition
Conda Application Management
Custom Application Management
Deadline Cloud UBL
Custom Licensing
Job Attachments
Custom Storage
Deadline Cloud monitor
Custom Monitor Application
Automated Monitoring Solution

云端工作室

现代视觉效果和动画工作室越来越多地将其整个流程转移到云端，包括艺术家工作站。这种方法消除了对本地基础设施的需求，实现了全球协作，并为交互式工作和渲染提供了无缝扩展。但是，它也在管理云资源、确保低延迟访问数据以及将基于云的工作站与渲染农场集成方面带来了新的挑战。

典型的云原生工作室需要采用统一的方法来管理云工作站、共享存储、渲染基础架构以及所有这些组件的软件部署。传统方法通常会导致复杂的手动管理系统，难以在性能、成本和灵活性之间取得平衡。

可以通过以下方式实现云原生工作室的 Deadline Cloud 部署：

- 通过云工作站上的集成 DCC 提交者进行交互式工作流程作业提交
- 通过 Deadline Cloud 管理的 conda 通道管理应用程序渲染节
- 为支持的软件自动配置基于使用情况的许可
- 使用 Windows File Server FSx 对共享项目数据进行自定义存储访问权限
- 通过 Deadline 云监控应用程序进行监控
- 使用服务托管队列管理基础架构

这种方法允许艺术家在基于云的工作站上工作，直接访问高性能共享存储，并将作业无缝提交到 Deadline Cloud农场。工作室可以使用相同的 conda 通道管理两个工作站和渲染节点之间的软件部署，从而确保一致性并减少维护开销。

此配置的主要优点包括：

- 与能够从任何地方访问工作站的艺术家进行全球合作
- 跨工作站和渲染节点的软件环境保持一致
- 工作站和渲染节点均可访问的高性能共享存储
- 灵活扩展交互式和批处理计算资源
- 在云端集中管理所有工作室基础架构

此场景中的存储配置通常包括：

- FSx Windowsfor File Server 用于存储项目数据，云工作站和 Deadline Cloud 工作人员均可访问
- Deadline Cloud 中的存储配置文件用于管理工作站和渲染节点之间的路径映射
- 使用 VPC 资源端点和主机配置脚本在 Deadline Cloud 工作线程上直接挂载 FSx 共享

这种云原生方法使工作室可以省去本地基础架构，从而可以快速扩展任何规模的项目，同时保持熟悉的艺术家工作流程。它提供了混合使用服务管理资源和客户管理资源的灵活性，并针对易管理性和特定的性能要求进行了优化。

通过利用云工作站和 Deadline Cloud，工作室可以实现完全集成、全球可访问的制作流程，从小型团队无缝扩展到大型制作。

ECommerce 自动化

现代电子商务平台需要大规模自动生成资产，才能为数百万个商品提供丰富的产品可视化。传统方法需要大量的基础设施投资才能将大量 3D 模型处理成标准化产品介质，这通常会导致系统配置不足，从而导致处理积压，或者系统过度配置闲置容量。

典型的自动化电子商务工作流程需要处理产品上传处理、3D 模型验证、渲染场管理、输出处理以及与产品信息系统的集成。传统上，管理这些工作流程需要协调多个渲染应用程序、计算资源和数据处理管道，同时确保质量一致并保持大规模成本效益。

可以使用以下方法实现电子商务自动化的 Deadline Cloud 部署：

- 通过在现有电子商务摄取应用程序中集成自定义 API，自动提交工作流程
- 为标准化产品可视化量身定制的自定义任务定义
- 通过 Deadline Cloud 托管 conda 渠道
- 为支持的软件自动配置基于使用情况的许可
- 直接与 Amazon S3 集成，用于资产管理
- 与现有产品管理系统集成的定制监控应用程序
- 服务托管队列可实现弹性扩展

这种方法可以每天处理数千种产品，自动生成标准化的产品可视化效果，例如转盘动画。服务管理的基础架构可自动扩展以满足不断变化的需求，同时通过员工重复使用和优化的应用程序部署保持成本效率。


eCommerce
Interactive Workflow
Automated Workflow
Integrated Submitter
Custom Job Definition
Conda Application Management
Custom Application Management
Deadline Cloud UBL
Custom Licensing
Job Attachments
Custom Storage
Deadline Cloud monitor
Custom Monitor Application
Automated Monitoring Solution

Whitelabel/OEM/B2C 顾客

传统的数字内容创作 (DCC) 软件通常要求用户维护自己的渲染基础架构或在工作站上本地处理渲染，这会导致大量的硬件投资或漫长的等待时间，从而中断创作工作流程。对于软件供应商而言，提供云渲染功能传统上需要构建和维护复杂的基础架构和计费系统。

集成到B2C软件中的Deadline Cloud部署可直接在用户熟悉的界面中实现无缝云渲染。这种集成结合了：

- DCC 应用程序中嵌入的交互式工作流程作业提交
- 用于部署渲染应用程序的 Deadline 云托管 conda 频道
- 自动配置基于使用情况的许可
- 通过带有供应商管理存储的任务附件进行资产管理
- 直接集成在 DCC 界面中的自定义监控
- 服务管理的队列在用户之间共享

这种方法允许最终用户在软件中单击一下即可将渲染图提交到云端，而无需管理帐户、基础架构或复杂的设置。软件供应商维护一个多租户环境，其中：

- 用户通过其现有的软件凭据进行身份验证
- 任务会自动路由到每用户专用的队列
- 使用 IAM 控制的存储前缀安全地隔离资产
- 账单通过供应商的现有系统处理
- Job 状态和输出直接流回用户的应用程序

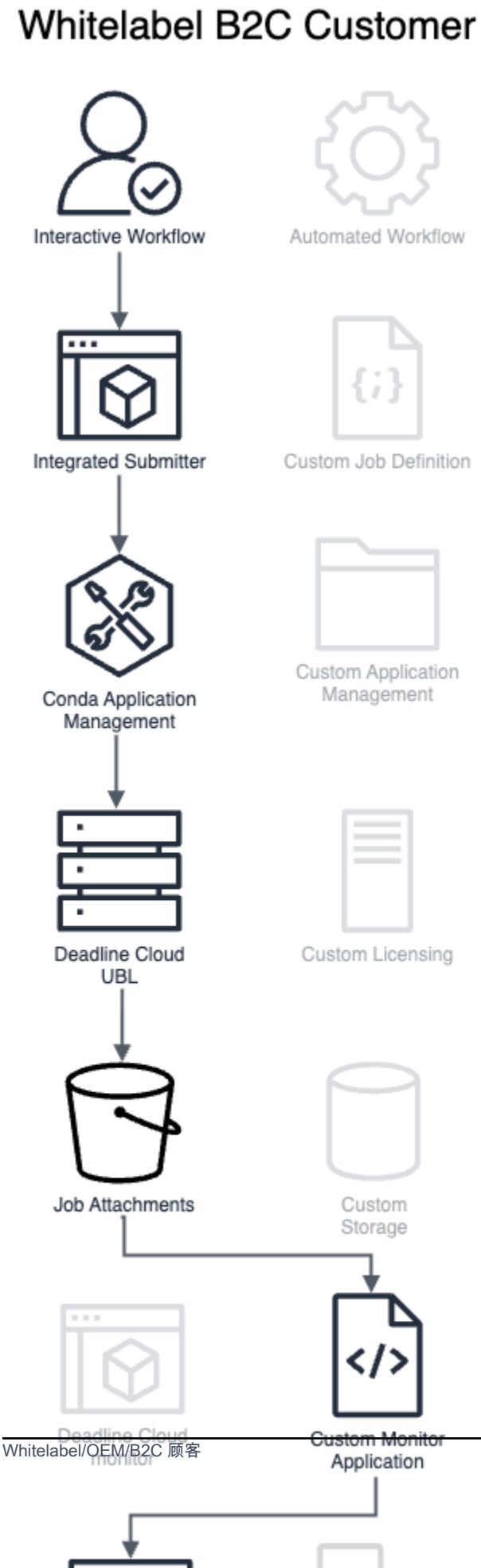
共享车队方法通过保持员工队伍的温暖库来确保最佳性能，最大限度地减少启动时间，同时最大限度地提高整个用户群的资源利用率。这种配置允许软件供应商将云渲染作为无缝产品功能提供，而不是需要额外设置或帐户的单独服务。

最终用户将受益于：

- 从他们熟悉的界面中一键提交
- Pay-as-you-go 无需基础架构管理即可定价
- 通过共享基础架构缩短作业启动时间
- 自动下载和整理已完成的渲染图

- 在所有平台上获得一致的体验

这种集成模式使软件供应商能够为其整个用户群提供企业级渲染功能，同时保持其应用程序原生的简单、消费者友好型体验。


Whitelabel B2C Customer
Interactive Workflow
Automated Workflow
Integrated Submitter
Custom Job Definition
Conda Application Management
Custom Application Management
Deadline Cloud UBL
Custom Licensing
Job Attachments
Custom Storage
Deadline Cloud monitor
Custom Monitor Application
Automated Monitoring Solution

什么是截止日期云工作负载

借 AWS 助 Deadline Cloud，您可以提交作业以在云端运行应用程序，并处理数据，以生成对您的业务至关重要的内容或见解。Deadline Cloud 使用开放作业描述 (OpenJD) 作为作业模板的语法，该规范专为视觉计算管道的需求而设计，但适用于许多其他用例。一些示例工作负载包括计算机图形渲染、物理模拟和摄影测量。

工作负载可以从用户使用 CLI 或自动生成的 GUI 提交到队列的简单任务捆绑包，到为应用程序定义的工作负载动态生成任务包的集成提交者插件。

工作负载是如何从生产中产生的

要了解生产环境中的工作负载以及如何使用 Deadline Cloud 为它们提供支持，请考虑它们是如何形成的。制作可能涉及创建视觉效果、动画、游戏、产品目录图像、用于建筑信息建模 (BIM) 的 3D 重建等。这些内容通常由运行各种软件应用程序和自定义脚本的艺术或技术专家团队创作。团队成员使用生产管道在彼此之间传递数据。管道执行的许多任务都涉及密集型计算，如果在用户的工作站上运行，则需要数天时间。

这些生产流程中的一些任务示例包括：

- 使用摄影测量应用程序处理拍摄的电影场景中的照片，以重建带纹理的数字网格。
- 在 3D 场景中运行粒子模拟，为电视节目的爆炸视觉效果添加层次细节。
- 将游戏关卡的数据转换为外部发布所需的形式，并应用优化和压缩设置。
- 为产品目录渲染一组图像，包括颜色、背景和光照的变化。
- 在 3D 模型上运行定制开发的脚本，以应用电影导演定制和批准的外观。

这些任务涉及许多需要调整的参数，以获得艺术效果或微调输出质量。通常会有一个 GUI 来选择这些参数值，通过按钮或菜单在应用程序中本地运行该过程。当用户运行该进程时，应用程序以及可能的主机本身不能用于执行其他操作，因为它使用内存中的应用程序状态，并且可能会消耗主机的所有 CPU 和内存资源。

在许多情况下，这个过程很快。在生产过程中，当对质量和复杂性的要求提高时，过程的速度就会减慢。在开发过程中花费30秒的角色测试在应用于最终制作角色时，很容易变成3小时。通过这种进展，在 GUI 中开始存在的工作负载可能会变得太大而无法容纳。将其移植到Deadline Cloud可以提高运行这些流程的用户的工作效率，因为他们可以重新完全控制自己的工作站，并且可以从Deadline Cloud监视器跟踪更多迭代。

在 Deadline Cloud 中开发对工作负载的支持时，需要达到两个级别的支持：

- 将工作负载从用户工作站转移到 Deadline Cloud 场中，无需并行处理或加速。这可能未充分利用服务器场中可用的计算资源，但是将长时间操作转移到批处理系统的能力使用户能够使用自己的工作站完成更多工作。
- 优化工作负载的并行性，使其利用 Deadline Cloud 场的水平规模快速完成。

有时候，如何使工作负载并行运行是显而易见的。例如，计算机图形渲染的每一帧都可以独立完成。但是，重要的是不要被这种并行性所困扰。相反，要明白，将长期运行的工作负载转移到Deadline Cloud可以带来显著的好处，即使没有明显的方法可以将工作负载分开。

工作负载的要素

要指定 Deadline Cloud 工作负载，请使用 De adline Cloud CLI 实现用户提交到队列的任务捆绑包。创建任务包的大部分工作是编写作业模板，但还有更多因素，例如如何提供工作负载所需的应用程序。在定义 Deadline Cloud 的工作负载时，需要考虑以下基本事项：

- 要运行的应用程序。该作业必须能够启动应用程序进程，因此需要安装可用的应用程序以及应用程序使用的任何许可，例如访问浮动许可证服务器。这通常是服务器场配置的一部分，而不是嵌入到任务包本身中。
  - 使用队列环境配置作业
  - Connect 将客户管理的车队连接到许可证端点
- Job 参数定义。提交作业的用户体验在很大程度上受到其提供的参数的影响。示例参数包括数据文件、目录和应用程序配置。
  - 任务捆绑包的参数值元素
- 文件数据流。作业运行时，它会从用户提供的文件中读取输入，然后将其输出写为新文件。要使用作业附件和路径映射功能，作业必须为这些输入和输出指定目录或特定文件的路径。
  - 在作业中使用文件
- 步骤脚本。步骤脚本使用正确的命令行选项运行应用程序二进制文件，以应用提供的作业参数。如果工作负载数据文件包含绝对路径引用而不是相对路径引用，它还会处理路径映射等细节。
  - 作业捆绑包的作业模板元素

工作负载可移植性

如果工作负载可以在多个不同的系统中运行，而无需在每次提交作业时都对其进行更改，则该工作负载是可移植的。例如，它可能在安装了不同共享文件系统的不同渲染农场上运行，也可以在不同的操作系统（如Linux或）上运行Windows。当您实现便携式作业包时，用户可以更轻松地在其特定服务器场上运行作业，或者对其进行调整以适应其他用例。

您可以通过以下几种方法让您的工作捆绑包变得便于携带。

- 使用作业捆绑包中的PATH作业参数和资产引用，完全指定工作负载所需的输入数据文件。这种方法使作业可以移植到基于共享文件系统的服务器场和制作输入数据副本的场中，例如 Deadline Cloud 作业附件功能。
- 使作业输入文件的文件路径引用可重定位，并在不同的操作系统上使用。例如，当用户从工作Windows站提交任务以在Linux队列上运行时。
  - 使用相对文件路径引用，因此，如果将包含它们的目录移到其他位置，则引用仍然可以解析。某些应用程序，例如 Blender，支持在相对路径和绝对路径之间进行选择。
  - 如果您不能使用相对路径，请支持 OpenJD 路径映射元数据，并根据 Deadline Cloud 为作业提供文件的方式转换绝对路径。
- 使用便携式脚本在作业中实现命令。Python 和 bash 是两个可以用这种方式使用的脚本语言示例。您应该考虑在车队的所有工作人员主机上同时提供这两者。
  - 使用脚本解释器二进制文件`bash`，例如`python`或，并将脚本文件名作为参数。与使用设置了执行位的脚本文件相比Windows，这种方法适用于所有操作系统，包括操作系统Linux。
  - 通过应用以下做法来编写便携式 bash 脚本：
    - 用单引号展开模板路径参数以处理带有空格和路径分隔符的Windows路径。
    - 运行时Windows，请注意与 minGW 自动路径转换有关的问题。例如，它将类似的 AWS CLI 命令`aws logs tail /aws/deadline/...`转换为类似的命令`aws logs tail "C:/Program Files/Git/aws/deadline/..."`，但不会正确跟踪日志。设置变量`MSYS_NO_PATHCONV=1`以关闭此行为。
    - 在大多数情况下，相同的代码适用于所有操作系统。当代码需要不同时，请使用`if/else`构造来处理案例。

```
if [[ "$(uname)" == MINGW* || "$(uname -s)" == MSYS_NT* ]]; then
    # Code for Windows
elif [[ "$(uname)" == Darwin ]]; then
    # Code for MacOS
else
    # Code for Linux and other operating systems
fi
```

  - 您可以使用编写可移植的 Python 脚本`pathlib`来处理文件系统路径差异并避免使用特定于操作的功能。Python 文档包括对此的注释，例如在信号库文档中。 Linux特定功能支持标记为 “可用性：Linux”。
- 使用作业参数来指定应用程序要求。使用服务器场管理员可以在队列环境中应用的一致约定。

- 例如，您可以在作业中使用CondaPackages和/或RezPackages参数，其默认参数值列出作业所需的应用程序包名称和版本。然后，您可以使用其中一个示例 conda 或 Rez 队列环境为作业提供虚拟环境。

Deadline Cloud 资源入门

要开始为 De AWS adline Cloud 创建自定义解决方案，您必须设置资源。其中包括农场、服务器场的至少一个队列以及为队列提供服务的至少一个工作人员车队。您可以使用 Deadline Cloud 控制台创建资源，也可以使用 AWS Command Line Interface。

在本教程中，您将使用 AWS CloudShell 创建一个简单的开发者群组并运行工作器代理。然后，您可以提交并运行带有参数和附件的简单作业，添加服务托管队列，并在完成后清理农场资源。

以下各节将向您介绍 Deadline Cloud 的不同功能，以及它们是如何运作和协同工作的。遵循这些步骤对于开发和测试新的工作负载和自定义项非常有用。

有关使用控制台设置服务器场的说明，请参阅 De adline Cloud 用户指南中的入门。

主题

- 创建截止日期云场
- 运行 Deadline 云端工作者代理
- 使用截止日期云提交
- 在 Deadline Cloud 中提交带有作业附件的工作
- 在 Deadline Cloud 中向你的开发者群添加服务管理队列
- 在 Deadline Cloud 中清理农场资源

创建截止日期云场

要在 De AWS adline Cloud 中创建开发者群和队列资源，请使用 AWS Command Line Interface (AWS CLI)，如以下过程所示。您还将创建一个 AWS Identity and Access Management (IAM) 角色和一个客户管理的队列 (CMF)，并将队列与您的队列关联。然后，您可以配置 AWS CLI 并确认您的服务器场已按指定设置并正常运行。

您可以使用此服务器场来探索 Deadline Cloud 的功能，然后开发和测试新的工作负载、自定义项和管道集成。

创建农场

1. 打开会 AWS CloudShell 话。您将使用 CloudShell 窗口输入 AWS Command Line Interface (AWS CLI) 命令来运行本教程中的示例。继续操作时，请保持 CloudShell 窗口处于打开状态。

2. 为您的农场创建一个名称，然后将该农场名称添加到~/.bashrc。这将使其可用于其他终端会话。

```
echo "DEV_FARM_NAME=DeveloperFarm" >> ~/.bashrc
source ~/.bashrc
```

3. 创建服务器场资源，并将其场 ID 添加到~/.bashrc。

```
aws deadline create-farm \
    --display-name "$DEV_FARM_NAME"

echo "DEV_FARM_ID=\$(aws deadline list-farms \
        --query \"farms[?displayName=='\$DEV_FARM_NAME'].farmId \
        | [0]\" --output text)" >> ~/.bashrc
source ~/.bashrc
```

4. 创建队列资源，并将其队列 ID 添加到 ~/.bashrc.

```
aws deadline create-queue \
    --farm-id $DEV_FARM_ID \
    --display-name "$DEV_FARM_NAME Queue" \
    --job-run-as-user '{"posix": {"user": "job-user", "group": "job-group"},
 "runAs":"QUEUE_CONFIGURED_USER"}'

echo "DEV_QUEUE_ID=\$(aws deadline list-queues \
        --farm-id \$DEV_FARM_ID \
        --query \"queues[?displayName=='\$DEV_FARM_NAME Queue'].queueId \
        | [0]\" --output text)" >> ~/.bashrc
source ~/.bashrc
```

5. 为舰队创建 IAM 角色。此角色为队列中的工作人员主机提供必要的安全证书，以便在队列中运行作业。

```
aws iam create-role \
    --role-name "${DEV_FARM_NAME}FleetRole" \
    --assume-role-policy-document \
        '{
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
```

```
                        "Service": "credentials.deadline.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }'
aws iam put-role-policy \
    --role-name "${DEV_FARM_NAME}FleetRole" \
    --policy-name WorkerPermissions \
    --policy-document \
        '{
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Action": [
                        "deadline:AssumeFleetRoleForWorker",
                        "deadline:UpdateWorker",
                        "deadline:DeleteWorker",
                        "deadline:UpdateWorkerSchedule",
                        "deadline:BatchGetJobEntity",
                        "deadline:AssumeQueueRoleForWorker"
                    ],
                    "Resource": "*",
                    "Condition": {
                        "StringEquals": {
                            "aws:PrincipalAccount": "${aws:ResourceAccount}"
                        }
                    }
                },
                {
                    "Effect": "Allow",
                    "Action": [
                        "logs:CreateLogStream"
                    ],
                    "Resource": "arn:aws:logs:*:*:*:/aws/deadline/*",
                    "Condition": {
                        "StringEquals": {
                            "aws:PrincipalAccount": "${aws:ResourceAccount}"
                        }
                    }
                },
                {
                    "Effect": "Allow",
```

```
                    "Action": [
                        "logs:PutLogEvents",
                        "logs:GetLogEvents"
                    ],
                    "Resource": "arn:aws:logs:*:*:*:/aws/deadline/*",
                    "Condition": {
                        "StringEquals": {
                            "aws:PrincipalAccount": "${aws:ResourceAccount}"
                        }
                    }
                }
            ]
        }'
```

6. 创建客户管理的队列 (CMF)，并将其队列 ID 添加到。`~/.bashrc`

```
FLEET_ROLE_ARN="arn:aws:iam::$(aws sts get-caller-identity \
        --query "Account" --output text):role/${DEV_FARM_NAME}FleetRole"
aws deadline create-fleet \
    --farm-id $DEV_FARM_ID \
    --display-name "$DEV_FARM_NAME CMF" \
    --role-arn $FLEET_ROLE_ARN \
    --max-worker-count 5 \
    --configuration \
        '{
            "customerManaged": {
                "mode": "NO_SCALING",
                "workerCapabilities": {
                    "vCpuCount": {"min": 1},
                    "memoryMiB": {"min": 512},
                    "osFamily": "linux",
                    "cpuArchitectureType": "x86_64"
                }
            }
        }'

echo "DEV_CMF_ID=\$(aws deadline list-fleets \
        --farm-id \$DEV_FARM_ID \
        --query \"fleets[?displayName=='\$DEV_FARM_NAME CMF'].fleetId \
        | [0]\" --output text)" >> ~/.bashrc
source ~/.bashrc
```

7. 将 CMF 与您的队列关联。

```
aws deadline create-queue-fleet-association \
    --farm-id $DEV_FARM_ID \
    --queue-id $DEV_QUEUE_ID \
    --fleet-id $DEV_CMF_ID
```

8. 安装 Deadline Cloud 命令行界面。

```
pip install deadline
```

9. 要将默认场设置为场 ID，将队列设置为之前创建的队列 ID，请使用以下命令。

```
deadline config set defaults.farm_id $DEV_FARM_ID
deadline config set defaults.queue_id $DEV_QUEUE_ID
```

10. （可选）要确认您的服务器场是否按照您的规格进行设置，请使用以下命令：

- 列出所有农场 — **`deadline farm list`**
- 列出默认服务器场中的所有队列 — **`deadline queue list`**
- 列出默认服务器场中的所有舰队 — **`deadline fleet list`**
- 获取默认农场 — **`deadline farm get`**
- 获取默认队列 — **`deadline queue get`**
- 获取所有与默认队列关联的舰队 — **`deadline fleet get`**

后续步骤

创建服务器场后，您可以在队列中的主机上运行 Deadline Cloud 工作代理来处理作业。请参阅运行 Deadline 云端工作者代理。

运行 Deadline 云端工作者代理

必须先在工作服务器主机上以开发者模式运行 De AWS adline Cloud 工作器代理，然后才能在开发者群中运行提交到队列的作业。

在本教程的其余部分中，您将使用两个 AWS CloudShell 选项卡在开发者群中执行 AWS CLI 操作。在第一个选项卡中，您可以提交作业。在第二个选项卡中，您可以运行工作器代理。

> **Note**
>
> 如果您的 CloudShell 会话闲置时间超过 20 分钟，则会话将超时并停止工作器代理。要重新启动工作器代理，请按照以下过程中的说明进行操作。

在启动工作人员代理之前，必须先设置 Deadline Cloud 场、队列和队列。请参阅创建截止日期云场。

在开发者模式下运行工作器代理

1. 当您的农场在第一个 CloudShell 选项卡中仍处于打开状态时，打开第二个 CloudShell 选项卡，然后创建`demoenv-logs`和`demoenv-persist`目录。

```
mkdir ~/demoenv-logs
mkdir ~/demoenv-persist
```

2. 从 PyPI 下载并安装 Deadline Cloud 工作器代理包：

> **Note**
>
> On Windows，则需要将代理文件安装到 Python 的全局站点包目录中。目前不支持 Python 虚拟环境。

```
python -m pip install deadline-cloud-worker-agent
```

3. 要允许工作器代理为正在运行的作业创建临时目录，请创建一个目录：

```
sudo mkdir /sessions
sudo chmod 750 /sessions
sudo chown cloudshell-user /sessions
```

4. 使用您添加到的变量`DEV_FARM_ID`在开发者模式下运行 Deadline Cloud 工作器代理`~/.bashrc`。`DEV_CMF_ID`

```
deadline-worker-agent \
    --farm-id $DEV_FARM_ID \
    --fleet-id $DEV_CMF_ID \
    --run-jobs-as-agent-user \
    --logs-dir ~/demoenv-logs \
```

```
    --persistence-dir ~/demoenv-persist
```

当工作代理初始化然后轮询 UpdateWorkerSchedule API 操作时，将显示以下输出：

```
INFO    Worker Agent starting
[2024-03-27 15:51:01,292][INFO    ] # Worker Agent starting
[2024-03-27 15:51:01,292][INFO    ] AgentInfo
Python Interpreter: /usr/bin/python3
Python Version: 3.9.16 (main, Sep  8 2023, 00:00:00)  - [GCC 11.4.1 20230605 (Red
 Hat 11.4.1-2)]
Platform: linux
...
[2024-03-27 15:51:02,528][INFO    ] # API.Resp # [deadline:UpdateWorkerSchedule]
(200) params={'assignedSessions': {}, 'cancelSessionActions': {},
 'updateIntervalSeconds': 15} ...
[2024-03-27 15:51:17,635][INFO    ] # API.Resp # [deadline:UpdateWorkerSchedule]
(200) params=(Duplicate removed, see previous response) ...
[2024-03-27 15:51:32,756][INFO    ] # API.Resp # [deadline:UpdateWorkerSchedule]
(200) params=(Duplicate removed, see previous response) ...
...
```

5. 选择您的第一个 CloudShell 选项卡，然后列出车队中的员工。

```
deadline worker list --fleet-id $DEV_CMF_ID
```

将显示如下输出：

```
Displaying 1 of 1 workers starting at 0

- workerId: worker-8c9af877c8734e89914047111f
  status: STARTED
  createdAt: 2023-12-13 20:43:06+00:00
```

在生产配置中，Deadline Cloud 工作器代理需要在主机上以管理用户的身份设置多个用户和配置目录。您可以覆盖这些设置，因为您在自己的开发场中运行作业，只有您可以访问这些开发场。

后续步骤

现在，您的工作器主机上正在运行工作器代理，您可以向您的工作人员发送作业。您可以：

- 使用截止日期云提交使用一个简单的 OpenJD 工作包。
- 在 Deadline Cloud 中提交带有作业附件的工作它们在使用不同操作系统的工作站之间共享文件。

使用截止日期云提交

要在工作服务器主机上运行 Deadline Cloud 作业，您需要创建并使用 OpenJD 作业描述 (OpenJD) 任务包来配置作业。捆绑包配置作业，例如，通过指定作业的输入文件以及将作业输出写入何处。本主题包括配置任务捆绑包的方法示例。

在按照本节中的步骤进行操作之前，必须完成以下操作：

- 创建截止日期云场
- 运行 Deadline 云端工作者代理

要使用 De AWS adline Cloud 运行作业，请按以下步骤操作。使用第一个 AWS CloudShell 选项卡向您的开发者群提交作业。使用第二个 CloudShell 选项卡查看工作器代理的输出。

主题

提交simple_job样本

创建服务器场并运行工作器代理后，您可以将simple_job示例提交到 Deadline Cloud。

将simple_job样本提交到 Deadline Cloud

1. 选择您的第一个 CloudShell 选项卡。
2. 从中下载示例 GitHub。

```
cd ~
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
```

3. 导航到任务捆绑包示例目录。

```
cd ~/deadline-cloud-samples/job_bundles/
```

4. 提交simple_job样本。

```
deadline bundle submit simple_job
```

5. 选择第二个 CloudShell 选项卡可查看有关呼叫BatchGetJobEntities、获取会话和运行会话操作的日志输出。

```
...
[2024-03-27 16:00:21,846][INFO    ] # Session.Starting
 # [session-053d77cef82648fe2] Starting new Session.
 [queue-3ba4ff683ff54db09b851a2ed8327d7b/job-d34cc98a6e234b6f82577940ab4f76c6]
[2024-03-27 16:00:21,853][INFO    ] # API.Req # [deadline:BatchGetJobEntity]
 resource={'farm-id': 'farm-3e24cfc9bbcd423e9c1b6754bc1',
 'fleet-id': 'fleet-246ee60f46d44559b6cce010d05', 'worker-id':
 'worker-75e0fce9c3c344a69bff57fcd83'} params={'identifiers': [{'jobDetails':
 {'jobId': 'job-d34cc98a6e234b6f82577940ab4'}}]} request_url=https://
scheduling.deadline.us-west-2.amazonaws.com/2023-10-12/farms/
farm-3e24cfc9bbcd423e /fleets/fleet-246ee60f46d44559b1 /workers/worker-
 75e0fce9c3c344a69b /batchGetJobEntity
[2024-03-27 16:00:22,013][INFO    ] # API.Resp # [deadline:BatchGetJobEntity](200)
 params={'entities': [{'jobDetails': {'jobId': 'job-d34cc98a6e234b6f82577940ab6',
 'jobRunAsUser': {'posix': {'user': 'job-user', 'group': 'job-group'},
 'runAs': 'QUEUE_CONFIGURED_USER'}, 'logGroupName': '/aws/deadline/
farm-3e24cfc9bbcd423e9c1b6754bc1/queue-3ba4ff683ff54db09b851a2ed83', 'parameters':
 '*REDACTED*', 'schemaVersion': 'jobtemplate-2023-09'}}], 'errors': []}
 request_id=a3f55914-6470-439e-89e5-313f0c6
[2024-03-27 16:00:22,013][INFO    ] # Session.Add #
 [session-053d77cef82648fea9c69827182] Appended new SessionActions.
 (ActionIds: ['sessionaction-053d77cef82648fea9c69827182-0'])
 [queue-3ba4ff683ff54db09b851a2ed8b/job-d34cc98a6e234b6f82577940ab6]
[2024-03-27 16:00:22,014][WARNING ] # Session.User #
 [session-053d77cef82648fea9c69827182] Running as the Worker Agent's
 user. (User: cloudshell-user) [queue-3ba4ff683ff54db09b851a2ed8b/job-
d34cc98a6e234b6f82577940ac6]
[2024-03-27 16:00:22,015][WARNING ] # Session.AWSCreds #
 [session-053d77cef82648fea9c69827182] AWS Credentials are not available: Queue has
 no IAM Role. [queue-3ba4ff683ff54db09b851a2ed8b/job-d34cc98a6e234b6f82577940ab6]
[2024-03-27 16:00:22,026][INFO    ] # Session.Logs #
 [session-053d77cef82648fea9c69827182] Logs streamed to: AWS CloudWatch
 Logs. (LogDestination: /aws/deadline/farm-3e24cfc9bbcd423e9c1b6754bc1/
```

```
queue-3ba4ff683ff54db09b851a2ed83/session-053d77cef82648fea9c69827181)
 [queue-3ba4ff683ff54db09b851a2ed83/job-d34cc98a6e234b6f82577940ab4]
[2024-03-27 16:00:22,026][INFO    ] # Session.Logs #
 [session-053d77cef82648fea9c69827182] Logs streamed to: local
 file. (LogDestination: /home/cloudshell-user/demoenv-logs/
queue-3ba4ff683ff54db09b851a2ed8b/session-053d77cef82648fea9c69827182.log)
 [queue-3ba4ff683ff54db09b851a2ed83/job-d34cc98a6e234b6f82577940ab4]
...
```

> **Note**
>
> 仅显示工作器代理的日志输出。运行作业的会话有一个单独的日志。

6. 选择第一个选项卡，然后检查工作器代理写入的日志文件。

   a. 导航到工作器代理日志目录并查看其内容。

   ```
   cd ~/demoenv-logs
   ls
   ```

   b. 打印工作器代理创建的第一个日志文件。

   ```
   cat worker-agent-bootstrap.log
   ```

   此文件包含工作人员代理输出，说明它如何调用 Deadline Cloud API 在您的队列中创建工作人员资源，然后担任队列角色。

   c. 打印工作器代理加入队列时的日志文件输出。

   ```
   cat worker-agent.log
   ```

   此日志包含有关工作器代理执行的所有操作的输出，但不包含有关其运行作业的队列 IDs 的输出，但这些资源除外。

   d. 在与队列资源 ID 同名的目录中打印每个会话的日志文件。

   ```
   cat $DEV_QUEUE_ID/session-*.log
   ```

   如果作业成功，则日志文件输出将类似于以下内容：

   ```
   cat $DEV_QUEUE_ID/$(ls -t $DEV_QUEUE_ID | head -1)
   ```

```
2024-03-27 16:00:22,026 WARNING Session running with no AWS Credentials.
2024-03-27 16:00:22,404 INFO
2024-03-27 16:00:22,405 INFO ================================================
2024-03-27 16:00:22,405 INFO --------- Running Task
2024-03-27 16:00:22,405 INFO ================================================
2024-03-27 16:00:22,406 INFO ------------------------------------------------
2024-03-27 16:00:22,406 INFO Phase: Setup
2024-03-27 16:00:22,406 INFO ------------------------------------------------
2024-03-27 16:00:22,406 INFO Writing embedded files for Task to disk.
2024-03-27 16:00:22,406 INFO Mapping: Task.File.runScript -> /sessions/
session-053d77cef82648fea9c698271812a/embedded_fileswa_gj55_/tmp2u9yqtsz
2024-03-27 16:00:22,406 INFO Wrote: runScript -> /sessions/
session-053d77cef82648fea9c698271812a/embedded_fileswa_gj55_/tmp2u9yqtsz
2024-03-27 16:00:22,407 INFO ------------------------------------------------
2024-03-27 16:00:22,407 INFO Phase: Running action
2024-03-27 16:00:22,407 INFO ------------------------------------------------
2024-03-27 16:00:22,407 INFO Running command /sessions/
session-053d77cef82648fea9c698271812a/tmpzuzxpslm.sh
2024-03-27 16:00:22,414 INFO Command started as pid: 471
2024-03-27 16:00:22,415 INFO Output:
2024-03-27 16:00:22,420 INFO Welcome to AWS Deadline Cloud!
2024-03-27 16:00:22,571 INFO
2024-03-27 16:00:22,572 INFO ================================================
2024-03-27 16:00:22,572 INFO --------- Session Cleanup
2024-03-27 16:00:22,572 INFO ================================================
2024-03-27 16:00:22,572 INFO Deleting working directory: /sessions/
session-053d77cef82648fea9c698271812a
```

7. 打印有关作业的信息。

```
deadline job get
```

提交作业时，系统会将其保存为默认值，因此您无需输入作业 ID。

提交simple_job带有参数的

您可以提交带有参数的作业。在以下步骤中，您可以编辑simple_job模板以包含自定义消息，提交simple_job，然后打印会话日志文件以查看消息。

提交带参数的simple_job示例

1. 选择您的第一个 CloudShell 选项卡，然后导航到任务捆绑包示例目录。

```
cd ~/deadline-cloud-samples/job_bundles/
```

2. 打印simple_job模板的内容。

```
cat simple_job/template.yaml
```

带有Message参数的parameterDefinitions部分应如下所示：

```
parameterDefinitions:
- name: Message
  type: STRING
  default: Welcome to AWS Deadline Cloud!
```

3. 提交带有参数值的simple_job示例，然后等待任务完成运行。

```
deadline bundle submit simple_job \
    -p "Message=Greetings from the developer getting started guide."
```

4. 要查看自定义消息，请查看最新的会话日志文件。

```
cd ~/demoenv-logs
cat $DEV_QUEUE_ID/$(ls -t $DEV_QUEUE_ID | head -1)
```

创建带有文件 I/O 的 simple_file_job 任务捆绑包

渲染作业需要读取场景定义，从中渲染图像，然后将该图像保存到输出文件中。您可以通过让作业计算输入的哈希值而不是渲染图像来模拟此操作。

创建带有文件 I/O 的 simple_file_job 任务捆绑包

1. 选择您的第一个 CloudShell 选项卡，然后导航到任务捆绑包示例目录。

```
cd ~/deadline-cloud-samples/job_bundles/
```

2. 用新名称制作一份副本simple_file_job。simple_job

```
cp -r simple_job simple_file_job
```

3. 按如下方式编辑作业模板：

> **Note**
>
> 我们建议您使用nano这些步骤。如果您更喜欢使用Vim，则必须使用设置其粘贴模式`:set paste`。

a. 在文本编辑器中打开模板。

```
nano simple_file_job/template.yaml
```

b. 添加以下内容`type`、`objectType`和`dataFlow``parameterDefinitions`。

```
- name: InFile
  type: PATH
  objectType: FILE
  dataFlow: IN
- name: OutFile
  type: PATH
  objectType: FILE
  dataFlow: OUT
```

c. 将以下bash脚本命令添加到文件末尾，该命令从输入文件读取并写入输出文件。

```
        # hash the input file, and write that to the output
        sha256sum "{{Param.InFile}}" > "{{Param.OutFile}}"
```

更新后的内容`template.yaml`应与以下内容完全匹配：

```
specificationVersion: 'jobtemplate-2023-09'
name: Simple File Job Bundle Example
parameterDefinitions:
- name: Message
  type: STRING
  default: Welcome to AWS Deadline Cloud!
- name: InFile
  type: PATH
```

```
  objectType: FILE
  dataFlow: IN
- name: OutFile
  type: PATH
  objectType: FILE
  dataFlow: OUT
steps:
- name: WelcomeToDeadlineCloud
  script:
    actions:
      onRun:
        command: '{{Task.File.Run}}'
    embeddedFiles:
    - name: Run
      type: TEXT
      runnable: true
      data: |
        #!/usr/bin/env bash
        echo "{{Param.Message}}"

        # hash the input file, and write that to the output
        sha256sum "{{Param.InFile}}" > "{{Param.OutFile}}"
```

> **Note**
>
> 如果要调整中的间距`template.yaml`，请确保使用空格而不是缩进。

d. 保存文件，然后退出文本编辑器。

4. 为输入和输出文件提供参数值以提交 simple_file_job。

```
deadline bundle submit simple_file_job \
    -p "InFile=simple_job/template.yaml" \
    -p "OutFile=hash.txt"
```

5. 打印有关作业的信息。

```
deadline job get
```

- 您将看到如下输出：

```
parameters:
```

```
  Message:
    string: Welcome to AWS Deadline Cloud!
  InFile:
    path: /local/home/cloudshell-user/BundleFiles/JobBundle-Examples/simple_job/
template.yaml
  OutFile:
    path: /local/home/cloudshell-user/BundleFiles/JobBundle-Examples/hash.txt
```

- 尽管您只提供了相对路径，但参数设置了完整路径。将当前工作目录与作为参数提供的任何路径 AWS CLI 连接起来，而这些路径的类型为该路径PATH。
- 在另一个终端窗口中运行的工作器代理接起并运行作业。此操作将创建hash.txt文件，您可以使用以下命令查看该文件。

  ```
  cat hash.txt
  ```

  此命令将打印类似于以下内容的输出。

  ```
  eaa2df5d34b54be5ac34c56a24a8c237b8487231a607eaf530a04d76b89c9cd3 /local/home/
  cloudshell-user/BundleFiles/JobBundle-Examples/simple_job/template.yaml
  ```

后续步骤

在学习了如何使用 Deadline Cloud CLI 提交简单作业后，您可以探索：

- 在 Deadline Cloud 中提交带有作业附件的工作学习如何在运行不同操作系统的主机上运行作业。
- 在 Deadline Cloud 中向你的开发者群添加服务管理队列在由 Deadline Cloud 管理的主机上运行作业。
- 在 Deadline Cloud 中清理农场资源关闭您在本教程中使用的资源。

在 Deadline Cloud 中提交带有作业附件的工作

许多服务器场使用共享文件系统在提交作业的主机和运行作业的主机之间共享文件。例如，在前面的simple_file_job示例中，本地文件系统在终端窗口之间共享， AWS CloudShell 终端窗口在您提交作业的选项卡一和运行工作代理的选项卡二中运行。

当提交者工作站和工作主机位于同一个局域网中时，共享文件系统更具优势。如果您将数据存储在本地访问数据的工作站附近，那么使用基于云的服务器场意味着您必须通过高延迟 VPN 共享文件系统或在云中同步文件系统。这两个选项都不容易设置或操作。

AWS Deadline Cloud 提供了一个带有作业附件的简单解决方案，类似于电子邮件附件。使用作业附件，您可以将数据附加到作业中。然后，Deadline Cloud 会处理在亚马逊简单存储服务 (Amazon S3) 存储桶中传输和存储任务数据的细节。

内容创建工作流程通常是迭代的，这意味着用户提交作业时会包含一小部分修改过的文件。由于 Amazon S3 存储桶将任务附件存储在内容可寻址的存储中，因此每个对象的名称都基于对象数据的哈希值，并且目录树的内容以任务所附的清单文件格式存储。

在按照本节中的步骤进行操作之前，必须完成以下操作：

- 创建截止日期云场
- 运行 Deadline 云端工作者代理

要运行带有作业附件的作业，请完成以下步骤。

主题

向队列中添加作业附件配置

要在队列中启用作业附件，请向账户中的队列资源添加作业附件配置。

向队列中添加作业附件配置

1. 选择您的第一个 CloudShell 选项卡，然后输入以下命令之一，使用 Amazon S3 存储桶存储任务附件。
   - 如果您没有现有的私有 Amazon S3 存储桶，则可以创建和使用新的 S3 存储桶。

```
DEV_FARM_BUCKET=$(echo $DEV_FARM_NAME \
    | tr '[:upper:]' '[:lower:]')-$(xxd -l 16 -p /dev/urandom)
if [ "$AWS_REGION" == "us-east-1" ]; then LOCATION_CONSTRAINT=
```

```
else LOCATION_CONSTRAINT="--create-bucket-configuration \
    LocationConstraint=${AWS_REGION}"
fi
aws s3api create-bucket \
    $LOCATION_CONSTRAINT \
    --acl private \
    --bucket ${DEV_FARM_BUCKET}
```

- 如果您已经拥有私有 Amazon S3 存储桶，则可以通过将其*MY_BUCKET_NAME*替换为存储桶的名称来使用它。

```
DEV_FARM_BUCKET=MY_BUCKET_NAME
```

2. 创建或选择 Amazon S3 存储桶后，将存储桶名称添加到~/.bashrc，以使该存储桶可用于其他终端会话。

```
echo "DEV_FARM_BUCKET=$DEV_FARM_BUCKET" >> ~/.bashrc
 source ~/.bashrc
```

3. 为队列创建 AWS Identity and Access Management (IAM) 角色。

```
aws iam create-role --role-name "${DEV_FARM_NAME}QueueRole" \
    --assume-role-policy-document \
        '{
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "credentials.deadline.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }'
aws iam put-role-policy \
    --role-name "${DEV_FARM_NAME}QueueRole" \
    --policy-name S3BucketsAccess \
    --policy-document \
            '{
                "Version": "2012-10-17",
                "Statement": [
                {
```

```
                "Action": [
                    "s3:GetObject*",
                    "s3:GetBucket*",
                    "s3:List*",
                    "s3:DeleteObject*",
                    "s3:PutObject",
                    "s3:PutObjectLegalHold",
                    "s3:PutObjectRetention",
                    "s3:PutObjectTagging",
                    "s3:PutObjectVersionTagging",
                    "s3:Abort*"
                ],
                "Resource": [
                    "arn:aws:s3:::'$DEV_FARM_BUCKET'",
                    "arn:aws:s3:::'$DEV_FARM_BUCKET'/*"
                ],
                "Effect": "Allow"
            }
        ]
        }'
```

4. 更新您的队列以包含任务附件设置和 IAM 角色。

```
QUEUE_ROLE_ARN="arn:aws:iam::$(aws sts get-caller-identity \
        --query "Account" --output text):role/${DEV_FARM_NAME}QueueRole"
aws deadline update-queue \
    --farm-id $DEV_FARM_ID \
    --queue-id $DEV_QUEUE_ID \
    --role-arn $QUEUE_ROLE_ARN \
    --job-attachment-settings \
        '{
            "s3BucketName": "'$DEV_FARM_BUCKET'",
            "rootPrefix": "JobAttachments"
        }'
```

5. 确认您已更新队列。

```
deadline queue get
```

输出如下所示：

```
...
jobAttachmentSettings:
```

```
  s3BucketName: DEV_FARM_BUCKET
  rootPrefix: JobAttachments
roleArn: arn:aws:iam::ACCOUNT_NUMBER:role/DeveloperFarmQueueRole
...
```

提交simple_file_job时附上工作附件

使用作业附件时，任务捆绑包必须为 Deadline Cloud 提供足够的信息，以确定作业的数据流，例如使用PATH参数。如果是simple_file_job，您编辑`template.yaml`文件是为了告诉 Deadline Cloud 数据流在输入文件和输出文件中。

将作业附件配置添加到队列后，您可以提交带有作业附件的 simple_file_job 示例。完成此操作后，您可以查看日志记录和作业输出，以确认simple_file_job带有作业附件的运行正常。

提交带有作业附件的 simple_file_job 任务捆绑包

1. 选择您的第一个 CloudShell 选项卡，然后打开该`JobBundle-Samples`目录。
2.
```
cd ~/deadline-cloud-samples/job_bundles/
```
3. 将 simple_file_job 提交到队列。当系统提示您确认上传时，请输入**y**。

```
deadline bundle submit simple_file_job \
    -p InFile=simple_job/template.yaml \
    -p OutFile=hash-jobattachments.txt
```
4. 要查看作业附件数据传输会话日志输出，请运行以下命令。

```
JOB_ID=$(deadline config get defaults.job_id)
SESSION_ID=$(aws deadline list-sessions \
        --farm-id $DEV_FARM_ID \
        --queue-id $DEV_QUEUE_ID \
        --job-id $JOB_ID \
        --query "sessions[0].sessionId" \
        --output text)
cat ~/demoenv-logs/$DEV_QUEUE_ID/$SESSION_ID.log
```
5. 列出在会话中运行的会话操作。

```
aws deadline list-session-actions \
    --farm-id $DEV_FARM_ID \
```

```
    --queue-id $DEV_QUEUE_ID \
    --job-id $JOB_ID \
    --session-id $SESSION_ID
```

输出如下所示：

```
{
    "sessionactions": [
        {
            "sessionActionId": "sessionaction-123-0",
            "status": "SUCCEEDED",
            "startedAt": "<timestamp>",
            "endedAt": "<timestamp>",
            "progressPercent": 100.0,
            "definition": {
                "syncInputJobAttachments": {}
            }
        },
        {
            "sessionActionId": "sessionaction-123-1",
            "status": "SUCCEEDED",
            "startedAt": "<timestamp>",
            "endedAt": "<timestamp>",
            "progressPercent": 100.0,
            "definition": {
                "taskRun": {
                    "taskId": "task-abc-0",
                    "stepId": "step-def"
                }
            }
        }
    ]
}
```

第一个会话操作下载了输入作业附件，而第二个操作则像前面的步骤一样运行任务，然后上传了输出作业附件。

6. 列出输出目录。

```
ls *.txt
```

例如，输出hash.txt存在于目录中，但hash-jobattachments.txt由于作业的输出文件尚未下载，因此不存在。

7. 下载最近作业的输出。

```
deadline job download-output
```

8. 查看已下载文件的输出。

```
cat hash-jobattachments.txt
```

输出如下所示：

```
eaa2df5d34b54be5ac34c56a24a8c237b8487231a607eaf530a04d76b89c9cd3  /tmp/openjd/
session-123/assetroot-abc/simple_job/template.yaml
```

了解任务附件在 Amazon S3 中的存储方式

您可以使用 AWS Command Line Interface (AWS CLI) 上传或下载任务附件的数据，这些数据存储在 Amazon S3 存储桶中。了解 Deadline Cloud 如何在 Amazon S3 上存储作业附件将有助于您开发工作负载和管道集成。

检查 Deadline Cloud 作业附件在 Amazon S3 中的存储方式

1. 选择您的第一个 CloudShell 选项卡，然后打开任务捆绑包示例目录。

```
cd ~/deadline-cloud-samples/job_bundles/
```

2. 检查作业属性。

```
deadline job get
```

输出如下所示：

```
parameters:
  Message:
    string: Welcome to AWS Deadline Cloud!
  InFile:
```

```
    path: /home/cloudshell-user/deadline-cloud-samples/job_bundles/simple_job/
template.yaml
  OutFile:
    path: /home/cloudshell-user/deadline-cloud-samples/job_bundles/hash-
jobattachments.txt
attachments:
  manifests:
  - rootPath: /home/cloudshell-user/deadline-cloud-samples/job_bundles/
    rootPathFormat: posix
    outputRelativeDirectories:
    - .
    inputManifestPath: farm-3040c59a5b9943d58052c29d907a645d/queue-
cde9977c9f4d4018a1d85f3e6c1a4e6e/Inputs/
f46af01ca8904cd8b514586671c79303/0d69cd94523ba617c731f29c019d16e8_input.xxh128
    inputManifestHash: f95ef91b5dab1fc1341b75637fe987ee
  fileSystem: COPIED
```

附件字段包含清单结构列表，这些清单结构描述了作业运行时使用的输入和输出数据路径。查看`rootPath`提交作业的计算机上的本地目录路径。要查看包含清单文件的 Amazon S3 对象后缀，请查看。`inputManifestFile`清单文件包含任务输入数据的目录树快照的元数据。

3. 漂亮地打印 Amazon S3 清单对象以查看任务的输入目录结构。

```
MANIFEST_SUFFIX=$(aws deadline get-job \
     --farm-id $DEV_FARM_ID \
     --queue-id $DEV_QUEUE_ID \
     --job-id $JOB_ID \
     --query "attachments.manifests[0].inputManifestPath" \
     --output text)
 aws s3 cp s3://$DEV_FARM_BUCKET/JobAttachments/Manifests/$MANIFEST_SUFFIX - | jq .
```

输出如下所示：

```
{
     "hashAlg": "xxh128",
     "manifestVersion": "2023-03-03",
     "paths": [
     {
         "hash": "2ec297b04c59c4741ed97ac8fb83080c",
         "mtime": 1698186190000000,
         "path": "simple_job/template.yaml",
         "size": 445
```

```
    }
    ],
    "totalSize": 445
}
```

4. 构造用于保存输出任务附件清单的 Amazon S3 前缀，并在其下列出对象。

```
SESSION_ACTION=$(aws deadline list-session-actions \
    --farm-id $DEV_FARM_ID \
    --queue-id $DEV_QUEUE_ID \
    --job-id $JOB_ID \
    --session-id $SESSION_ID \
    --query "sessionActions[?definition.taskRun != null] | [0]")
STEP_ID=$(echo $SESSION_ACTION | jq -r .definition.taskRun.stepId)
TASK_ID=$(echo $SESSION_ACTION | jq -r .definition.taskRun.taskId)
TASK_OUTPUT_PREFIX=JobAttachments/Manifests/$DEV_FARM_ID/$DEV_QUEUE_ID/$JOB_ID/
$STEP_ID/$TASK_ID/
aws s3api list-objects-v2 --bucket $DEV_FARM_BUCKET --prefix $TASK_OUTPUT_PREFIX
```

输出任务附件不是直接从任务资源中引用的，而是根据服务器场资源放置在 Amazon S3 存储桶中 IDs。

5. 获取特定会话操作 ID 的最新清单对象密钥，然后漂亮地打印清单对象。

```
SESSION_ACTION_ID=$(echo $SESSION_ACTION | jq -r .sessionActionId)
 MANIFEST_KEY=$(aws s3api list-objects-v2 \
    --bucket $DEV_FARM_BUCKET \
    --prefix $TASK_OUTPUT_PREFIX \
    --query "Contents[*].Key" --output text \
    | grep $SESSION_ACTION_ID \
    | sort | tail -1)
 MANIFEST_OBJECT=$(aws s3 cp s3://$DEV_FARM_BUCKET/$MANIFEST_KEY -)
 echo $MANIFEST_OBJECT | jq .
```

您将在输出hash-jobattachments.txt中看到该文件的属性，如下所示：

```
{
    "hashAlg": "xxh128",
    "manifestVersion": "2023-03-03",
    "paths": [
    {
        "hash": "f60b8e7d0fabf7214ba0b6822e82e08b",
```

```
        "mtime": 1698785252554950,
        "path": "hash-jobattachments.txt",
        "size": 182
    }
    ],
    "totalSize": 182
}
```

每次运行任务时，您的作业只会有一个清单对象，但一般而言，每次运行的任务可能会有更多的对象。

6. 在前缀下查看可寻址内容的 Amazon S3 存储输出。Data

```
FILE_HASH=$(echo $MANIFEST_OBJECT | jq -r .paths[0].hash)
FILE_PATH=$(echo $MANIFEST_OBJECT | jq -r .paths[0].path)
aws s3 cp s3://$DEV_FARM_BUCKET/JobAttachments/Data/$FILE_HASH -
```

输出如下所示：

```
eaa2df5d34b54be5ac34c56a24a8c237b8487231a607eaf530a04d76b89c9cd3  /tmp/openjd/
session-123/assetroot-abc/simple_job/template.yaml
```

后续步骤

在学习了如何使用 Deadline Cloud CLI 提交带有附件的作业后，您可以探索：

- 使用截止日期云提交学习如何在工作主机上使用 OpenJD 捆绑包运行作业。
- 在 Deadline Cloud 中向你的开发者群添加服务管理队列在由 Deadline Cloud 管理的主机上运行作业。
- 在 Deadline Cloud 中清理农场资源关闭您在本教程中使用的资源。

在 Deadline Cloud 中向你的开发者群添加服务管理队列

AWS CloudShell 无法提供足够的计算容量来测试更大的工作负载。它也未配置为处理在多个工作主机上分配任务的作业。

您可以将 A CloudShell uto Scaling 服务托管队列 (SMF) 添加到您的开发者群中，而不必使用。SMF 为较大的工作负载提供了足够的计算容量，并且可以处理需要在多个工作主机上分配作业任务的作业。

在添加 SMF 之前，必须设置 Deadline Cloud 场、队列和队列。请参阅创建截止日期云场。

将服务托管队列添加到您的开发者群中

1. 选择您的第一个 AWS CloudShell 选项卡，然后创建服务管理的队列并将其队列 ID 添加到。`.bashrc`此操作使其可用于其他终端会话。

```
FLEET_ROLE_ARN="arn:aws:iam::$(aws sts get-caller-identity \
        --query "Account" --output text):role/${DEV_FARM_NAME}FleetRole"
 aws deadline create-fleet \
     --farm-id $DEV_FARM_ID \
     --display-name "$DEV_FARM_NAME SMF" \
     --role-arn $FLEET_ROLE_ARN \
     --max-worker-count 5 \
     --configuration \
         '{
             "serviceManagedEc2": {
                 "instanceCapabilities": {
                     "vCpuCount": {
                         "min": 2,
                         "max": 4
                     },
                     "memoryMiB": {
                         "min": 512
                     },
                     "osFamily": "linux",
                     "cpuArchitectureType": "x86_64"
                 },
                 "instanceMarketOptions": {
                     "type": "spot"
                 }
             }
         }'

 echo "DEV_SMF_ID=$(aws deadline list-fleets \
         --farm-id $DEV_FARM_ID \
         --query "fleets[?displayName=='$DEV_FARM_NAME SMF'].fleetId \
         | [0]" --output text)" >> ~/.bashrc
 source ~/.bashrc
```

2. 将 SMF 与您的队列关联。

```
aws deadline create-queue-fleet-association \
     --farm-id $DEV_FARM_ID \
     --queue-id $DEV_QUEUE_ID \
     --fleet-id $DEV_SMF_ID
```

3. 提交simple_file_job到队列。当系统提示您确认上传时，请输入**y**。

```
deadline bundle submit simple_file_job \
    -p InFile=simple_job/template.yaml \
    -p OutFile=hash-jobattachments.txt
```

4. 确认 SMF 工作正常。

```
deadline fleet get
```

   - 工作人员可能需要几分钟才能开始。重复该`deadline fleet get`命令，直到您可以看到舰队正在运行。
   - `queueFleetAssociationsStatus`用于服务管理的舰队将是。`ACTIVE`
   - SMF `autoScalingStatus` 将从变`GROWING`为。`STEADY`

   您的状态将类似于以下内容：

```
fleetId: fleet-2cc78e0dd3f04d1db427e7dc1d51ea44
farmId: farm-63ee8d77cdab4a578b685be8c5561c4a
displayName: DeveloperFarm SMF
description: ''
status: ACTIVE
autoScalingStatus: STEADY
targetWorkerCount: 0
workerCount: 0
minWorkerCount: 0
maxWorkerCount: 5
```

5. 查看您提交的作业的日志。此日志存储在 Amazon Logs 的 CloudWatch 日志中，而不是 CloudShell 文件系统中。

```
 JOB_ID=$(deadline config get defaults.job_id)
 SESSION_ID=$(aws deadline list-sessions \
```

```
        --farm-id $DEV_FARM_ID \
        --queue-id $DEV_QUEUE_ID \
        --job-id $JOB_ID \
        --query "sessions[0].sessionId" \
        --output text)
aws logs tail /aws/deadline/$DEV_FARM_ID/$DEV_QUEUE_ID \
    --log-stream-names $SESSION_ID
```

后续步骤

创建并测试服务托管队列后，应删除创建的资源，以避免不必要的费用。

- 在 Deadline Cloud 中清理农场资源关闭您在本教程中使用的资源。

在 Deadline Cloud 中清理农场资源

要开发和测试新的工作负载和管道集成，您可以继续使用为本教程创建的 Deadline Cloud 开发者群组。如果您不再需要开发者群组，则可以删除其资源，包括场、队列、队列、 AWS Identity and Access Management (IAM) 角色和 Amazon Logs 中的 CloudWatch 日志。删除这些资源后，您需要重新开始本教程才能使用这些资源。有关更多信息，请参阅 Deadline Cloud 资源入门。

清理开发者农场资源

1. 选择第一个 CloudShell 选项卡，然后停止队列的所有队列队列关联。

```
FLEETS=$(aws deadline list-queue-fleet-associations \
        --farm-id $DEV_FARM_ID \
        --queue-id $DEV_QUEUE_ID \
        --query "queueFleetAssociations[].fleetId" \
        --output text)
for FLEET_ID in $FLEETS; do
    aws deadline update-queue-fleet-association \
        --farm-id $DEV_FARM_ID \
        --queue-id $DEV_QUEUE_ID \
        --fleet-id $FLEET_ID \
        --status STOP_SCHEDULING_AND_CANCEL_TASKS
done
```

2. 列出队列队列关联。

```
aws deadline list-queue-fleet-associations \
     --farm-id $DEV_FARM_ID \
     --queue-id $DEV_QUEUE_ID
```

在输出报告之前，您可能需要重新运行该命令"status": "STOPPED"，然后才能继续下一步。此过程可能需要几分钟才能完成。

```
{
    "queueFleetAssociations": [
        {
            "queueId": "queue-abcdefgh01234567890123456789012id",
            "fleetId": "fleet-abcdefgh01234567890123456789012id",
            "status": "STOPPED",
            "createdAt": "2023-11-21T20:49:19+00:00",
            "createdBy": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/
MySessionName",
            "updatedAt": "2023-11-21T20:49:38+00:00",
            "updatedBy": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/
MySessionName"
        },
        {
            "queueId": "queue-abcdefgh01234567890123456789012id",
            "fleetId": "fleet-abcdefgh01234567890123456789012id",
            "status": "STOPPED",
            "createdAt": "2023-11-21T20:32:06+00:00",
            "createdBy": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/
MySessionName",
            "updatedAt": "2023-11-21T20:49:39+00:00",
            "updatedBy": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/
MySessionName"
        }
    ]
}
```

3. 删除队列的所有队列队列关联。

```
for FLEET_ID in $FLEETS; do
     aws deadline delete-queue-fleet-association \
         --farm-id $DEV_FARM_ID \
         --queue-id $DEV_QUEUE_ID \
         --fleet-id $FLEET_ID
```

```
done
```

4. 删除与您的队列关联的所有舰队。

```
for FLEET_ID in $FLEETS; do
     aws deadline delete-fleet \
         --farm-id $DEV_FARM_ID \
         --fleet-id $FLEET_ID
 done
```

5. 删除队列。

```
aws deadline delete-queue \
     --farm-id $DEV_FARM_ID \
     --queue-id $DEV_QUEUE_ID
```

6. 删除农场。

```
aws deadline delete-farm \
     --farm-id $DEV_FARM_ID
```

7. 删除农场的其他 AWS 资源。

   a. 删除舰队 AWS Identity and Access Management (IAM) 角色。

```
aws iam delete-role-policy \
     --role-name "${DEV_FARM_NAME}FleetRole" \
     --policy-name WorkerPermissions
aws iam delete-role \
     --role-name "${DEV_FARM_NAME}FleetRole"
```

   b. 删除队列 IAM 角色。

```
aws iam delete-role-policy \
     --role-name "${DEV_FARM_NAME}QueueRole" \
     --policy-name S3BucketsAccess
aws iam delete-role \
     --role-name "${DEV_FARM_NAME}QueueRole"
```

   c. 删除 Amazon CloudWatch 日志组。每个队列和队列都有自己的日志组。

```
aws logs delete-log-group \
     --log-group-name "/aws/deadline/$DEV_FARM_ID/$DEV_QUEUE_ID"
```

```
aws logs delete-log-group \
     --log-group-name "/aws/deadline/$DEV_FARM_ID/$DEV_CMF_ID"
aws logs delete-log-group \
     --log-group-name "/aws/deadline/$DEV_FARM_ID/$DEV_SMF_ID"
```

创建要提交到截止日期云的作业

您可以使用任务捆绑包向 Deadline Cloud 提交作业。作业捆绑包是文件的集合，包括开放式作业描述 (OpenJD) 作业模板和呈现作业所需的任何资产文件。

作业模板描述了工作人员如何处理和访问资产，并提供了工作人员运行的脚本。Job bundle 使美术师、技术总监和管道开发人员能够轻松地从本地工作站或本地渲染农场向 Deadline Cloud 提交复杂作业。Job bundle 对于从事需要可扩展的按需计算资源的大型视觉效果、动画或其他媒体渲染项目的团队特别有用。

您可以使用本地文件系统来存储文件，使用文本编辑器来创建作业模板。创建捆绑包后，使用 Deadline Cloud CLI 或 Deadline Cloud 提交者之类的工具将任务提交到 Deadline Cloud

您可以将资产存储在工作人员之间共享的文件系统中，也可以使用 Deadline Cloud 作业附件自动将资产移动到 S3 存储桶，让工作人员可以在那里访问它们。Job 附件还有助于将作业的输出移回工作站。

以下各节提供了有关创建工作捆绑包并将其提交到 Deadline Cloud 的详细说明。

主题

Deadline Cloud 的打开职位描述 (OpenJD) 模板

任务捆绑包是你用来为 De AWS adline Cloud 定义作业的工具之一。他们将 Ope n Job Description (OpenJD) 模板与附加信息分组，例如您的作业与作业附件一起使用的文件和目录。您可以使用 Deadline Cloud 命令行界面 (CLI) 使用任务捆绑包提交任务以供队列运行。

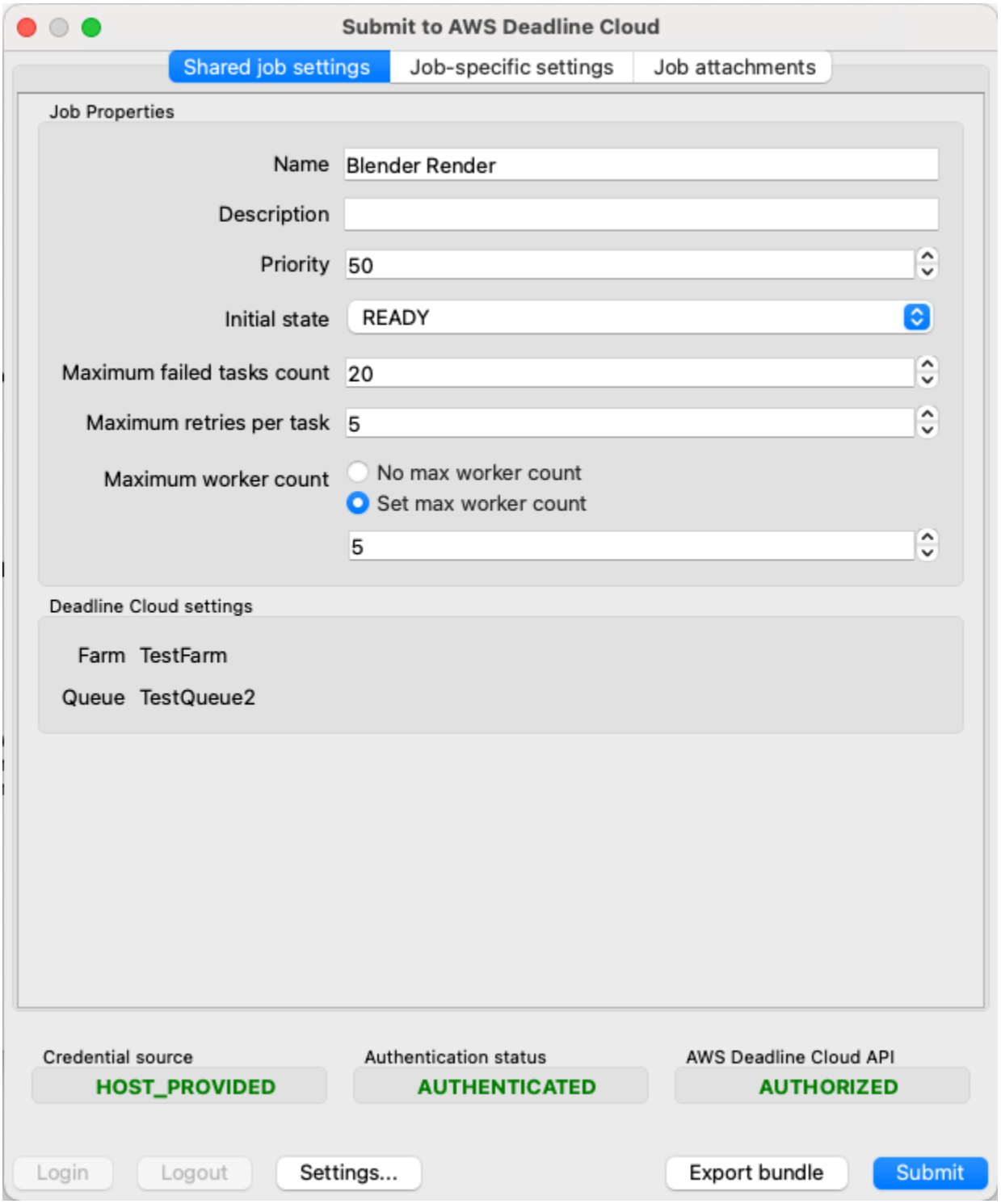
作业捆绑包是一种目录结构，其中包含 OpenJD 作业模板、定义作业的其他文件以及作业输入所需的特定于作业的文件。您可以将定义任务的文件指定为 YAML 或 JSON 文件。

唯一需要的文件是template.yaml或template.json。您还可以包含以下文件：

```
/template.yaml (or template.json)
/asset_references.yaml (or asset_references.json)
/parameter_values.yaml (or parameter_values.json)
/other job-specific files and directories
```

使用作业捆绑包通过 Deadline Cloud CLI 和作业附件提交自定义作业，也可以使用图形提交界面。例如，以下是来自 Blender 的示例 GitHub。在 Blender 示例目录中使用以下命令运行示例：

```
deadline bundle gui-submit blender_render
```



特定于作业的设置面板由作业模板中定义的作业参数的`userInterface`属性生成。

要使用命令行提交作业，您可以使用类似于以下内容的命令

```
deadline bundle submit \
```

```
    --yes \
    --name Demo \
     -p BlenderSceneFile=location of scene file \
     -p OutputDir=file pathe for job output \
      blender_render/
```

或者你可以使用 deadline Python 包中的deadline.client.api.create_job_from_job_bundle函数。

Deadline Cloud 提供的所有作业提交者插件，例如 Autodesk Maya 插件，都会生成一个供你提交的作业包，然后使用 Deadline Cloud Python 包将你的作业提交到 Deadline Cloud。您可以在工作站的作业历史目录中查看已提交的作业捆绑包，也可以使用提交者查看。您可以使用以下命令找到您的作业历史目录：

```
deadline config get settings.job_history_dir
```

当您的作业在 Deadline Cloud 工作线程上运行时，它可以访问环境变量，这些变量为其提供有关该作业的信息。环境变量是：

变量名称	可用
DEADLINE_FARM_ID	所有操作
DEADLINE_FLEET_ID	所有操作
DEADLINE_WORKER_ID	所有操作
截止日期_队列_ID	所有操作
截止日期_JOB_ID	所有操作
截止日期_STEP_ID	任务操作
截止日期_会话_ID	所有操作
DEADLINE_TASK_ID	任务操作
截止日期会话操作_ID	所有操作

主题

作业捆绑包的作业模板元素

作业模板定义了运行时环境和作为 Deadline Cloud 作业的一部分运行的进程。你可以在模板中创建参数，这样它就可以用来创建只在输入值上有所不同的作业，就像编程语言中的函数一样。

当您向 Deadline Cloud 提交作业时，该任务将在应用于该队列的任何队列环境中运行。队列环境是使用 Open Job Description (OpenJD) 外部环境规范构建的。有关详细信息，请参阅 OpenJD GitHub 存储库中的环境模板。

有关使用 OpenJD 作业模板创建作业的简介，请参阅在 OpenJD GitHub 存储库中创建作业简介。更多信息可以在作业的运行方式中找到。OpenJD GitHub 存储库的`samples`目录中有作业模板示例。

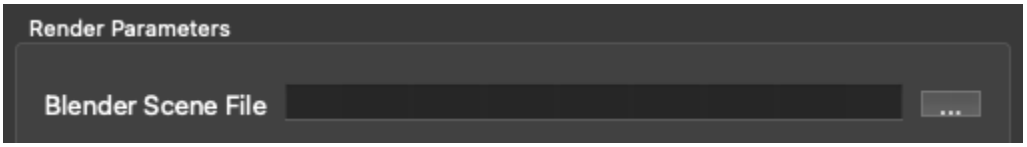
您可以用 YAML 格式 (`template.yaml`) 或 JSON 格式 (`template.json`) 定义作业模板。本节中的示例以 YAML 格式显示。

例如，该`blender_render`示例的作业模板将输入参数定义`BlenderSceneFile`为文件路径：

```
- name: BlenderSceneFile
  type: PATH
  objectType: FILE
  dataFlow: IN
  userInterface:
    control: CHOOSE_INPUT_FILE
    label: Blender Scene File
    groupLabel: Render Parameters
    fileFilters:
    - label: Blender Scene Files
      patterns: ["*.blend"]
    - label: All Files
      patterns: ["*"]
  description: >
    Choose the Blender scene file to render. Use the 'Job Attachments' tab
    to add textures and other files that the job needs.
```

该userInterface属性定义了使用命令的命令行自动生成的用户界面的行为，也定义了在 Autodesk Maya 等应用程序的作业提交插件中自动生成的用户界面的行为。deadline bundle gui-submit

在此示例中，用于输入BlenderSceneFile参数值的 UI 控件是一个仅显示文件的文件选择对话框。.blend



有关使用该userInteface元素的更多示例，请参阅存储库中的 gui_control_showcase 示例。deadline-cloud-samples GitHub

objectType和dataFlow属性控制从作业捆绑包提交作业时作业附件的行为。在本例中objectType: FILE，and dataFlow:IN 表示的值BlenderSceneFile是作业附件的输入文件。

相比之下，OutputDir参数的定义有objectType: DIRECTORY和dataFlow: OUT：

```
- name: OutputDir
  type: PATH
  objectType: DIRECTORY
  dataFlow: OUT
  userInterface:
    control: CHOOSE_DIRECTORY
    label: Output Directory
    groupLabel: Render Parameters
  default: "./output"
  description: Choose the render output directory.
```

作业附件使用该OutputDir参数的值作为作业写入输出文件的目录。

有关objectType和dataFlow属性的更多信息，请参阅 Ope n Job Description 规范JobPathParameterDefinition中的

blender_render作业模板示例的其余部分将作业的工作流程定义为单个步骤，动画中的每一帧都呈现为单独的任务：

```
steps:
- name: RenderBlender
  parameterSpace:
```

```
    taskParameterDefinitions:
    - name: Frame
      type: INT
      range: "{{Param.Frames}}"
  script:
    actions:
      onRun:
        command: bash
        # Note: {{Task.File.Run}} is a variable that expands to the filename on the
worker host's
        # disk where the contents of the 'Run' embedded file, below, is written.
        args: ['{{Task.File.Run}}']
    embeddedFiles:
      - name: Run
        type: TEXT
        data: |
          # Configure the task to fail if any individual command fails.
          set -xeuo pipefail

          mkdir -p '{{Param.OutputDir}}'

          blender --background '{{Param.BlenderSceneFile}}' \
                  --render-output '{{Param.OutputDir}}/{{Param.OutputPattern}}' \
                  --render-format {{Param.Format}} \
                  --use-extension 1 \
                  --render-frame {{Task.Param.Frame}}
```

例如，如果`Frames`参数的值为`1-10`，则它定义 10 个任务。每个 has task 的`Frame`参数值都不同。要运行任务，请执行以下操作：

1. 例如，嵌入式文件`data`属性中的所有变量引用都会被展开`--render-frame 1`。
2. 该`data`属性的内容将写入磁盘上会话工作目录中的一个文件中。
3. 任务的`onRun`命令解析为，`bash` *`location of embedded file`*然后运行。

有关嵌入文件、会话和路径映射位置的更多信息，请参阅 Open Job Description 规范中的作业运行方式。

deadline-cloud-samples/job_bundles 存储库中还有更多作业模板示例，以及随开放职位描述规范提供的模板示例。

作业模板的任务分块

任务分块允许您将多个任务分组为一个工作单元，称为块。例如，在渲染作业中，这意味着 Deadline Cloud 可以一起调度多个帧，而不是每次命令调用一帧。这减少了为每个任务启动应用程序的开销，并缩短了作业的总运行时间。有关详细信息，请参阅 OpenJD 维基中的一次运行多个帧。

OpenJD 支持为作业模板添加可选功能的扩展。通过添加TASK_CHUNKING扩展来启用任务分块。要使用分块，请将扩展添加到您的作业模板中，然后使用CHUNK[INT]任务参数类型。使用相同的`deadline bundle submit`命令提交分块作业。例如，以下作业模板以 10 个区块呈现帧：

```
specificationVersion: 'jobtemplate-2023-09'
extensions:
  - TASK_CHUNKING
name: Blender Render with Contiguous Chunking
parameterDefinitions:
  - name: BlenderSceneFile
    type: PATH
    objectType: FILE
    dataFlow: IN
  - name: Frames
    type: STRING
    default: "1-100"
  - name: OutputDir
    type: PATH
    objectType: DIRECTORY
    dataFlow: OUT
    default: "./output"
steps:
  - name: RenderBlender
    parameterSpace:
      taskParameterDefinitions:
        - name: Frame
          type: CHUNK[INT]
          range: "{{Param.Frames}}"
          chunks:
            defaultTaskCount: 10
            rangeConstraint: CONTIGUOUS
    script:
      actions:
        onRun:
          command: bash
          args: ["{{Task.File.Run}}"]
      embeddedFiles:
```

```
    - name: Run
      type: TEXT
      data: |
        set -xeuo pipefail

        mkdir -p '{{Param.OutputDir}}'

        # Parse the chunk range (e.g., "1-10") into start and end frames
        START_FRAME="$(echo '{{Task.Param.Frame}}' | cut -d- -f1)"
        END_FRAME="$(echo '{{Task.Param.Frame}}' | cut -d- -f2)"

        blender --background '{{Param.BlenderSceneFile}}' \
                --render-output '{{Param.OutputDir}}/output_####' \
                --render-format PNG \
                --use-extension 1 \
                -s "$START_FRAME" \
                -e "$END_FRAME" \
                --render-anim
```

在此示例中，Deadline Cloud 将 100 个帧分成诸如1-1011-20、之类的块。{{Task.Param.Frame}}变量会扩展为范围表达式，例如1-10。因为设置rangeConstraint为CONTIGUOUS，所以范围始终是start-end格式化的。该脚本解析此范围，并使用和-e选项将起始帧和结束帧传递给 Blender。-s --render-anim

该chunks属性支持以下字段：

- defaultTaskCount— （必填）要将多少任务合并成一个块。最大值为 150。
- rangeConstraint— （必需）如果CONTIGUOUS，则区块始终是一个连续的范围，例如。1-10如果NONCONTIGUOUS，则区块可以是任意集合，比如1,3,7-10。
- targetRuntimeSeconds— （可选）每个区块的目标运行时间（以秒为单位）。在某些区块完成后，Deadline Cloud 可以动态调整区块大小以接近该目标。

有关更多任务分块示例，包括包含连续和非连续区块的基本示例和 Blender 示例，请参阅 Deadline Cloud 示例存储库中的任务分块示例。 GitHub

> **客户管理的车队要求**
>
> 任务分块需要兼容的工作器代理版本。如果您使用客户管理的队列，请确保在提交带有分块的作业之前更新您的工作人员代理。服务托管队列始终使用兼容的工作代理版本。

> **正在下载分块作业的输出**
>
> 当你下载分块作业中单个任务的输出时，Deadline Cloud 会下载整个区块的输出。例如，如果将第 1-10 帧一起处理，则下载第 3 帧的输出将包括所有 1-10 帧。此功能需要deadline-cloud版本 0.53.3 或更高版本。

任务捆绑包的参数值元素

您可以使用参数文件来设置任务模板中某些任务参数的值或任务捆绑包中的CreateJob操作请求参数的值，这样在提交任务时就无需设置值。提交作业的用户界面允许您修改这些值。

您可以用 YAML 格式 (parameter_values.yaml) 或 JSON 格式 (parameter_values.json) 定义作业模板。本节中的示例以 YAML 格式显示。

在 YAML 中，文件的格式为：

```
parameterValues:
- name: <string>
  value: <integer>, <float>, or <string>
- name: <string>
  value: <integer>, <float>, or <string>ab
... repeating as necessary
```

parameterValues列表中的每个元素都必须是以下元素之一：

- 在作业模板中定义的作业参数。
- 在队列环境中为你提交作业的队列定义的作业参数...
- 创建作业时传递给CreateJob操作的特殊参数。
  - deadline:priority— 该值必须为整数。它作为优先级参数传递给CreateJob操作。
  - deadline:targetTaskRunStatus— 该值必须是字符串。它作为targetTaskRun状态参数传递给CreateJob操作。
  - deadline:maxFailedTasksCount— 该值必须为整数。它作为 C maxFailedTasksount 参数传递给CreateJob操作。
  - deadline:maxRetriesPerTask— 该值必须为整数。它作为 T maxRetriesPerask 参数传递给CreateJob操作。
  - deadline:maxWorkercount— 该值必须为整数。它作为maxWorkerCount参数传递给CreateJob操作。

作业模板始终是一个模板，而不是要运行的特定作业。如果某些参数在此文件中未定义值，则参数值文件使作业捆绑包可以充当模板，或者如果所有参数都有值，则可以用作特定的作业提交。

例如，blender_render 示例没有参数文件，其作业模板定义的参数没有默认值。此模板必须用作创建作业的模板。使用此任务捆绑包创建任务后，Deadline Cloud 会将新的任务捆绑包写入作业历史记录目录。

例如，当您使用以下命令提交作业时：

```
deadline bundle gui-submit blender_render/
```

新的任务捆绑包包含一个包含指定参数的parameter_values.yaml文件：

```
% cat ~/.deadline/job_history/\(default\)/2024-06/2024-06-20-01-JobBundle-Demo/
parameter_values.yaml
parameterValues:
- name: deadline:targetTaskRunStatus
  value: READY
- name: deadline:maxFailedTasksCount
  value: 10
- name: deadline:maxRetriesPerTask
  value: 5
- name: deadline:priority
  value: 75
- name: BlenderSceneFile
  value: /private/tmp/bundle_demo/bmw27_cpu.blend
- name: Frames
  value: 1-10
- name: OutputDir
  value: /private/tmp/bundle_demo/output
- name: OutputPattern
  value: output_####
- name: Format
  value: PNG
- name: CondaPackages
  value: blender
- name: RezPackages
  value: blender
```

您可以使用以下命令创建相同的作业：

```
deadline bundle submit ~/.deadline/job_history/\(default\)/2024-06/2024-06-20-01-
JobBundle-Demo/
```

> **Note**
>
> 您提交的任务包将保存到您的作业历史记录目录中。你可以使用以下命令找到该目录的位置：
>
> ```
> deadline config get settings.job_history_dir
> ```

作业捆绑包的资产参考元素

你可以使用 Deadline Cloud 作业附件在工作站和 Deadline Cloud 之间来回传输文件。资产参考文件列出了输入文件和目录，以及附件的输出目录。如果您没有列出此文件中的所有文件和目录，则可以在使用`deadline bundle gui-submit`命令提交作业时选择它们。

如果您不使用作业附件，则此文件无效。

您可以用 YAML 格式 (`asset_references.yaml`) 或 JSON 格式 (`asset_references.json`) 定义作业模板。本节中的示例以 YAML 格式显示。

在 YAML 中，文件的格式为：

```
assetReferences:
    inputs:
        # Filenames on the submitting workstation whose file contents are needed as
        # inputs to run the job.
        filenames:
        - list of file paths
        # Directories on the submitting workstation whose contents are needed as inputs
        # to run the job.
        directories:
        - list of directory paths

    outputs:
        # Directories on the submitting workstation where the job writes output files
        # if running locally.
        directories:
        - list of directory paths
```

```
    # Paths referenced by the job, but not necessarily input or output.
    # Use this if your job uses the name of a path in some way, but does not explicitly
need
    # the contents of that path.
    referencedPaths:
    - list of directory paths
```

在选择要上传到 Amazon S3 的输入或输出文件时，Deadline Cloud 会将文件路径与您的存储配置文件中列出的路径进行比较。存储配置SHARED文件中的每个文件系统位置都抽象出安装在工作站和工作主机上的网络文件共享。Deadline Cloud 仅上传不在其中一个文件共享中的文件。

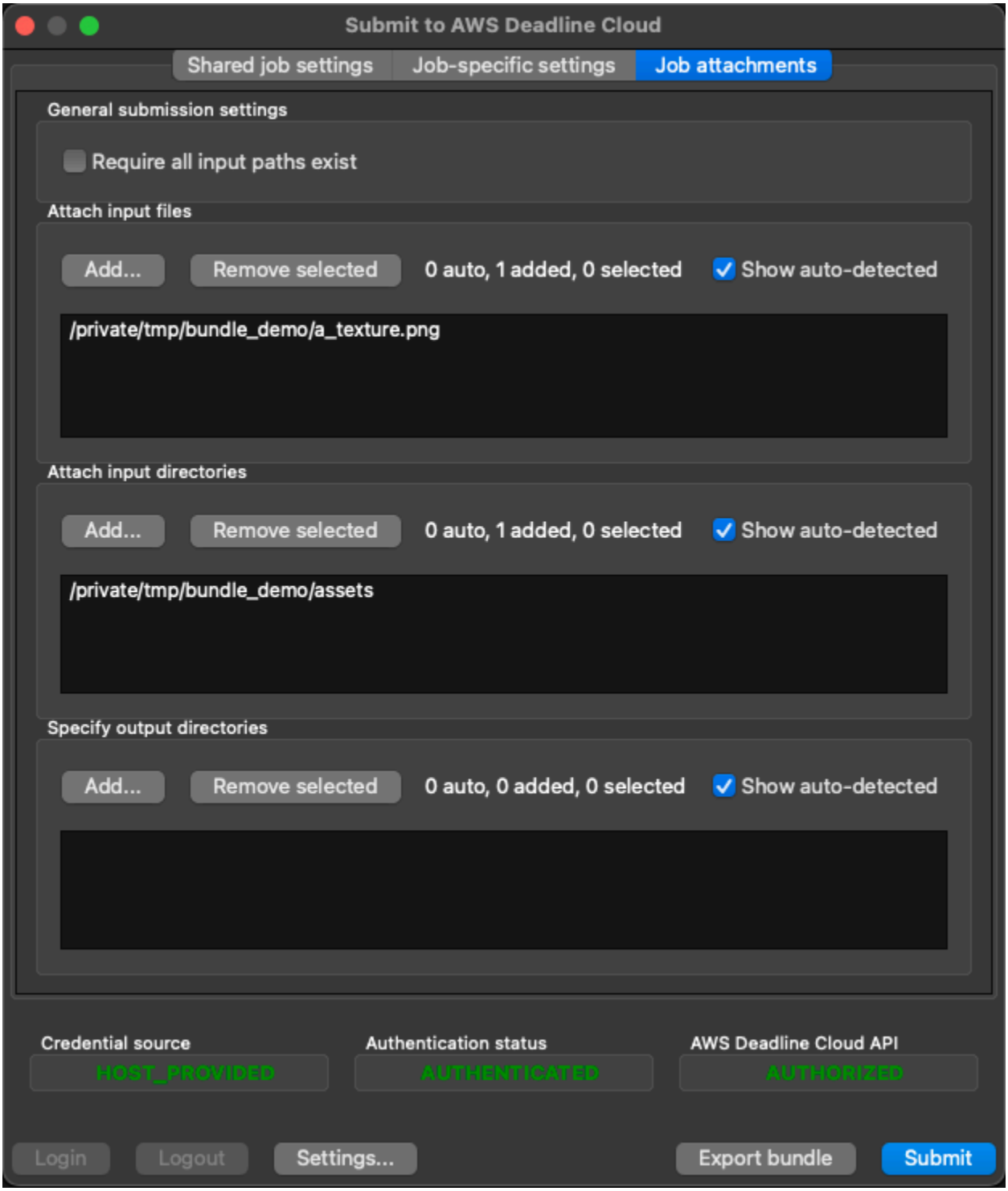
有关创建和使用存储配置文件的更多信息，请参阅 Dead line Cloud 用户指南中的 Deadline Cloud 中的共享存储。AWS

Example-由 Deadline Cloud GUI 创建的资产参考文件

使用以下命令通过 blender_rend er 示例提交作业。

```
deadline bundle gui-submit blender_render/
```

在 Job 附件选项卡上向作业添加一些其他文件：



提交作业后，您可以查看作业历史目录中任务捆绑包中的`asset_references.yaml`文件，以查看 YAML 文件中的资产：

```
% cat ~/.deadline/job_history/\(default\)/2024-06/2024-06-20-01-JobBundle-Demo/
asset_references.yaml
```

```
assetReferences:
  inputs:
    filenames:
    - /private/tmp/bundle_demo/a_texture.png
    directories:
    - /private/tmp/bundle_demo/assets
  outputs:
    directories: []
  referencedPaths: []
```

在作业中使用文件

您提交给 De AWS adline Cloud 的许多作业都有输入和输出文件。您的输入文件和输出目录可能位于共享文件系统和本地驱动器的组合上。工作需要在这些位置找到内容。Deadline Cloud 提供两种功能，即作业附件和存储配置文件，它们可以协同工作，帮助您的作业找到所需的文件。

Job 附件有多项好处

- 使用 Amazon S3 在主机之间移动文件
- 将文件从工作站传输到工作主机，反之亦然
- 适用于已启用该功能的队列中的作业
- 主要用于服务管理的车队，但也与客户管理的车队兼容。

使用存储配置文件映射工作站和工作主机上共享文件系统位置的布局。当您的工作站和工作主机的位置不同时，此映射可以帮助您的作业找到共享文件和目录，例如使用Windows基于工作站和Linux基于工作主机的跨平台设置。任务附件还使用存储配置文件中的文件系统配置映射来识别通过 Amazon S3 在主机之间穿梭所需的文件。

如果您不使用作业附件，并且不需要在工作站和工作主机之间重新映射文件和目录位置，则无需使用存储配置文件对文件共享进行建模。

主题

- 示例项目基础架构
- 存储配置文件和路径映射

示例项目基础架构

要演示如何使用作业附件和存储配置文件，请设置一个包含两个独立项目的测试环境。您可以使用 Deadline Cloud 控制台来创建测试资源。

1. 如果您还没有，请创建一个测试场。要创建场，请按照创建场中的步骤进行操作。
2. 在这两个项目中分别为作业创建两个队列。要创建队列，请按照创建队列中的步骤进行操作。

   a. 创建第一个名为的队列**Q1**。使用以下配置，对所有其他项目使用默认配置。

      - 对于任务附件，请选择创建新的 Amazon S3 存储桶。
      - 选择 “启用与客户管理的车队的关联”。
      - 对于以用户身份运行，请同时输入 **jobuser** POSIX 用户和群组。
      - 对于队列服务角色，创建一个名为的新角色 **AssetDemoFarm-Q1-Role**
      - 清除默认 conda 队列环境复选框。

   b. 创建第二个名为的队列**Q2**。使用以下配置，对所有其他项目使用默认配置。

      - 对于任务附件，请选择创建新的 Amazon S3 存储桶。
      - 选择 “启用与客户管理的车队的关联”。
      - 对于以用户身份运行，请同时输入 **jobuser** POSIX 用户和群组。
      - 对于队列服务角色，创建一个名为的新角色 **AssetDemoFarm-Q2-Role**
      - 清除默认 conda 队列环境复选框。

3. 创建一个由客户管理的队列来运行两个队列中的作业。要创建队列，请按照创建客户管理的队列中的步骤进行操作。使用以下配置：

   - 对于 “名称”，使用**DemoFleet**。
   - 对于舰队类型，请选择客户管理
   - 对于舰队服务角色，创建一个名为 AssetDemoFarm-Fleet-R ole 的新角色。
   - 请勿将队列与任何队列关联。

测试环境假设主机之间使用网络文件共享共享三个文件系统。在此示例中，这些地点的名称如下：

- FSCommon-包含两个项目通用的输入作业资产。
- FS1-包含项目 1 的输入和输出作业资产。
- FS2-包含项目 2 的输入和输出作业资产。

测试环境还假设有三个工作站，如下所示：

- WSAll-开发人员在所有项目中使用的Linux基于工作站。共享文件系统的位置是：
  - FSCommon: /shared/common
  - FS1: /shared/projects/project1
  - FS2: /shared/projects/project2
- WS1-用于项目 1 的Windows基于工作站。共享文件系统的位置是：
  - FSCommon: S:\
  - FS1: Z:\
  - FS2: 不可用
- WS1-用于项目 2 的macOS基于工作站。共享文件系统位置为：
  - FSCommon: /Volumes/common
  - FS1: 不可用
  - FS2: /Volumes/projects/project2

最后，为队列中的工作人员定义共享文件系统位置。以下示例将此配置称为WorkerConfig。共享位置是：

- FSCommon: /mnt/common
- FS1: /mnt/projects/project1
- FS2: /mnt/projects/project2

您无需设置任何与此配置相匹配的共享文件系统、工作站或工作服务器。演示不需要存在共享地点。

存储配置文件和路径映射

使用存储配置文件对工作站和工作主机上的文件系统进行建模。每个存储配置文件都描述了其中一个系统配置的操作系统和文件系统布局。本主题介绍如何使用存储配置文件对主机的文件系统配置进行建模，以便 Deadline Cloud 可以为您的作业生成路径映射规则，以及如何根据存储配置文件生成这些路径映射规则。

当你向 Deadline Cloud 提交任务时，你可以为该任务提供一个可选的存储配置文件 ID。此存储配置文件描述了提交工作站的文件系统。它描述了作业模板中的文件路径使用的原始文件系统配置。

您也可以将存储配置文件与队列关联。存储配置文件描述了队列中所有工作主机的文件系统配置。如果您的工作服务器具有不同的文件系统配置，则必须将这些工作人员分配到服务器场中的不同队列。

路径映射规则描述了应如何将路径从作业中的指定方式重新映射到工作主机上路径的实际位置。Deadline Cloud 将作业存储配置文件中描述的文件系统配置与运行该任务的队列的存储配置文件进行比较，以得出这些路径映射规则。

主题

使用存储配置文件对共享文件系统位置进行建模

存储配置文件对其中一个主机配置的文件系统配置进行建模。示例项目基础架构中有四种不同的主机配置。在此示例中，您将为每个配置文件创建单独的存储配置文件。您可以使用以下任一方法创建存储配置文件：

- CreateStorageProfile API
- AWS::Deadline::StorageProfile CloudFormation 资源
- AWS 控制台

存储配置文件由文件系统位置列表组成，每个位置都告诉 Deadline Cloud 与从主机提交或在主机上运行的作业相关的文件系统位置的位置和类型。存储配置文件应仅对与作业相关的位置进行建模。例如，共享FSCommon位置位于工作站WS1上S:\，因此相应的文件系统位置为：

```
{
    "name": "FSCommon",
    "path": "S:\\",
    "type": "SHARED"
}
```

使用以下命令为工作站配置WS1、和工作器配置创建存储配置文件WS2，WS3并WorkerConfig使用AWS CLI中的命令创建工作器配置 AWS CloudShell：

```
# Change the value of FARM_ID to your farm's identifier
```

```
FARM_ID=farm-00112233445566778899aabbccddeeff

aws deadline create-storage-profile --farm-id $FARM_ID \
  --display-name WSAll \
  --os-family LINUX \
  --file-system-locations \
  '[
      {"name": "FSCommon", "type":"SHARED", "path":"/shared/common"},
      {"name": "FS1", "type":"SHARED", "path":"/shared/projects/project1"},
      {"name": "FS2", "type":"SHARED", "path":"/shared/projects/project2"}
  ]'

aws deadline create-storage-profile --farm-id $FARM_ID \
  --display-name WS1 \
  --os-family WINDOWS \
  --file-system-locations \
  '[
      {"name": "FSCommon", "type":"SHARED", "path":"S:\\"},
      {"name": "FS1", "type":"SHARED", "path":"Z:\\"}
   ]'

aws deadline create-storage-profile --farm-id $FARM_ID \
  --display-name WS2 \
  --os-family MACOS \
  --file-system-locations \
  '[
      {"name": "FSCommon", "type":"SHARED", "path":"/Volumes/common"},
      {"name": "FS2", "type":"SHARED", "path":"/Volumes/projects/project2"}
  ]'

aws deadline create-storage-profile --farm-id $FARM_ID \
  --display-name WorkerCfg \
  --os-family LINUX \
  --file-system-locations \
  '[
      {"name": "FSCommon", "type":"SHARED", "path":"/mnt/common"},
      {"name": "FS1", "type":"SHARED", "path":"/mnt/projects/project1"},
      {"name": "FS2", "type":"SHARED", "path":"/mnt/projects/project2"}
  ]'
```

> **Note**
>
> 在服务器场中所有存储配置文件中，必须使用相同的name属性值来引用存储配置文件中的文件系统位置。Deadline Cloud 比较名称以确定在生成路径映射规则时，来自不同存储配置文件的文件系统位置指的是相同的位置。

为队列配置存储配置文件

您可以将队列配置为包含存储配置文件，该配置文件对队列中所有工作人员的文件系统位置进行建模。队列中所有工作人员的主机文件系统配置必须与其队列的存储配置文件相匹配。具有不同文件系统配置的工作人员必须位于不同的队列中。

要将队列的配置设置为使用WorkerConfig存储配置文件，请使用AWS CLI中的 AWS CloudShell：

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of FLEET_ID to your fleet's identifier
FLEET_ID=fleet-00112233445566778899aabbccddeeff
# Change the value of WORKER_CFG_ID to your storage profile named WorkerConfig
WORKER_CFG_ID=sp-00112233445566778899aabbccddeeff

FLEET_WORKER_MODE=$( \
  aws deadline get-fleet --farm-id $FARM_ID --fleet-id $FLEET_ID \
   --query '.configuration.customerManaged.mode' \
)
FLEET_WORKER_CAPABILITIES=$( \
  aws deadline get-fleet --farm-id $FARM_ID --fleet-id $FLEET_ID \
   --query '.configuration.customerManaged.workerCapabilities' \
)

aws deadline update-fleet --farm-id $FARM_ID --fleet-id $FLEET_ID \
  --configuration \
  "{
    \"customerManaged\": {
      \"storageProfileId\": \"$WORKER_CFG_ID\",
      \"mode\": $FLEET_WORKER_MODE,
      \"workerCapabilities\": $FLEET_WORKER_CAPABILITIES
    }
  }"
```

为队列配置存储配置文件

队列的配置包括一份区分大小写的共享文件系统位置名称列表，提交到队列的作业需要访问这些位置。例如，提交到队列的作业Q1需要文件系统位置FSCommon和。FS1提交到队列的作业Q2需要文件系统位置FSCommon和FS2。

要将队列的配置设置为需要这些文件系统位置，请使用以下脚本：

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of QUEUE2_ID to queue Q2's identifier
QUEUE2_ID=queue-00112233445566778899aabbccddeeff

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --required-file-system-location-names-to-add FSComm FS1

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE2_ID \
  --required-file-system-location-names-to-add FSComm FS2
```

队列的配置还包括允许的存储配置文件列表，这些配置文件适用于提交给该队列的作业以及与该队列关联的队列。队列允许的存储配置文件列表中只允许为队列的所有必需文件系统位置定义文件系统位置的存储配置文件。

如果您提交任务时使用的存储配置文件不在队列允许的存储配置文件列表中，则作业将失败。您可以随时向队列提交没有存储配置文件的作业。标有标签WSAll的工作站配置WS1都有队列所需的文件系统位置（FSCommon和FS1）Q1。需要允许他们向队列提交作业。同样，工作站WSAll的配置也WS2符合队列的要求Q2。需要允许他们向该队列提交作业。使用以下脚本更新两个队列配置，以允许使用这些存储配置文件提交作业：

```
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff
# Change the value of WS1 to the identifier of the WS1 storage profile
WS1_ID=sp-00112233445566778899aabbccddeeff
# Change the value of WS2 to the identifier of the WS2 storage profile
WS2_ID=sp-00112233445566778899aabbccddeeff

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --allowed-storage-profile-ids-to-add $WSALL_ID $WS1_ID
```

```
aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE2_ID \
  --allowed-storage-profile-ids-to-add $WSALL_ID $WS2_ID
```

如果将WS2存储配置文件添加到队列允许的存储配置文件列表中，Q1则会失败：

```
$ aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --allowed-storage-profile-ids-to-add $WS2_ID

An error occurred (ValidationException) when calling the UpdateQueue operation: Storage
 profile id: sp-00112233445566778899aabbccddeeff does not have required file system
 location: FS1
```

这是因为WS2存储配置文件不包含该队列Q1所需的文件系统位置FS1的定义。

将配置的队列与不在队列允许的存储配置文件列表中的存储配置文件关联也会失败。例如：

```
$ aws deadline create-queue-fleet-association --farm-id $FARM_ID \
   --fleet-id $FLEET_ID \
   --queue-id $QUEUE1_ID

An error occurred (ValidationException) when calling the CreateQueueFleetAssociation
 operation: Mismatch between storage profile ids.
```

要修复错误，请将名为WorkerConfig的存储配置文件添加到队列Q1和队列允许的存储配置文件列表中Q2。然后，将队列与这些队列相关联，以便队列中的工作人员可以运行两个队列中的作业。

```
# Change the value of FLEET_ID to your fleet's identifier
FLEET_ID=fleet-00112233445566778899aabbccddeeff
# Change the value of WORKER_CFG_ID to your storage profile named WorkerCfg
WORKER_CFG_ID=sp-00112233445566778899aabbccddeeff

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --allowed-storage-profile-ids-to-add $WORKER_CFG_ID

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE2_ID \
  --allowed-storage-profile-ids-to-add $WORKER_CFG_ID

aws deadline create-queue-fleet-association --farm-id $FARM_ID \
  --fleet-id $FLEET_ID \
  --queue-id $QUEUE1_ID
```

```
aws deadline create-queue-fleet-association --farm-id $FARM_ID \
  --fleet-id $FLEET_ID \
  --queue-id $QUEUE2_ID
```

根据存储配置文件派生路径映射规则

路径映射规则描述了如何将路径从作业重新映射到工作主机上路径的实际位置。当任务在工作程序上运行时，会将该作业的存储配置文件与工作人员队列的存储配置文件进行比较，以得出该任务的路径映射规则。

Deadline Cloud 为队列配置中每个必需的文件系统位置创建映射规则。例如，使用WSAll存储配置文件提交到队列的作业Q1具有路径映射规则：

- FSComm: `/shared/common -> /mnt/common`
- FS1: `/shared/projects/project1 -> /mnt/projects/project1`

Deadline Cloud 会为FSComm和FS1文件系统位置创建规则，但不会为FS2文件系统位置创建规则，即使WSAll和WorkerConfig存储配置文件都定义了也是如此FS2。这是因为队列Q1的所需文件系统位置列表是["FSComm", "FS1"]。

您可以通过提交打印出 Ope n Job Description 路径映射规则文件的作业，然后在作业完成后读取会话日志，来确认使用特定存储配置文件提交的作业可用的路径映射规则：

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of WSALL_ID to the identifier of the WSALL storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

aws deadline create-job --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --priority 50 \
  --storage-profile-id $WSALL_ID \
  --template-type JSON --template \
  '{
    "specificationVersion": "jobtemplate-2023-09",
    "name": "DemoPathMapping",
    "steps": [
      {
```

```
        "name": "ShowPathMappingRules",
        "script": {
          "actions": {
            "onRun": {
              "command": "/bin/cat",
              "args": [ "{{Session.PathMappingRulesFile}}" ]
            }
          }
        }
      }
    ]
  }'
```

如果您使用 De adline Cloud CLI 提交作业，则其配置settings.storage_profile_id设置将设置通过 CLI 提交的作业将具有的存储配置文件。要使用WSAll存储配置文件提交作业，请设置：

```
deadline config set settings.storage_profile_id $WSALL_ID
```

要像在示例基础架构中运行一样运行客户管理的工作器，请按照《De adline Cloud 用户指南》中的 “运行工作器代理” 中的步骤来运行工作器。 AWS CloudShell如果您之前按照这些说明进行操作，请先删除~/demoenv-logs和~/demoenv-persist目录。此外，在执行此操作之前，请按如下方式设置方向所引用的DEV_FARM_ID和DEV_CMF_ID环境变量的值：

```
DEV_FARM_ID=$FARM_ID
DEV_CMF_ID=$FLEET_ID
```

作业运行后，您可以在作业的日志文件中看到路径映射规则：

```
cat demoenv-logs/${QUEUE1_ID}/*.log
...
JJSON log results (see below)
...
```

该日志包含FS1和FSComm文件系统的映射。为了便于阅读，重新格式化了日志条目，如下所示：

```
{
    "version": "pathmapping-1.0",
    "path_mapping_rules": [
        {
            "source_path_format": "POSIX",
```

```
            "source_path": "/shared/projects/project1",
            "destination_path": "/mnt/projects/project1"
        },
        {
            "source_path_format": "POSIX",
            "source_path": "/shared/common",
            "destination_path": "/mnt/common"
        }
    ]
```

您可以提交具有不同存储配置文件的作业，以查看路径映射规则是如何变化的。

使用作业附件共享文件

使用作业附件使不在共享目录中的文件可用于您的作业，如果输出文件未写入共享目录，则可以捕获这些文件。Job 附件使用 Amazon S3 在主机之间传输文件。文件存储在 S3 存储桶中，如果文件内容未更改，则无需上传文件。

在服务管理的队列上运行作业时，必须使用作业附件，因为主机不共享文件系统位置。当作业的输入或输出文件存储在共享网络文件系统上时，例如当你的任务包中包含 shell 或 Python 脚本时，作业附件对客户管理的队列也很有用。

当您使用 Deadline Cloud CLI 或 Dead line Cloud 提交者提交任务捆绑包时，作业附件会使用作业的存储配置文件和队列所需的文件系统位置来识别不在工作主机上且应作为作业提交的一部分上传到 Amazon S3 的输入文件。这些存储配置文件还有助于 Deadline Cloud 识别工作服务器主机位置中的输出文件，这些文件必须上传到 Amazon S3 才能供您的工作站使用。

作业附件示例使用和中的服务器场、队列、队列和存储配置文件配置存储配置文件和路径映射。示例项目基础架构在此之前，您应该仔细阅读这些部分。

在以下示例中，您使用示例作业捆绑包作为起点，然后对其进行修改以探索作业附件的功能。Job bundle 是您的工作使用作业附件的最佳方式。它们将目录中的 O pen Job Description 作业模板与列出使用任务捆绑包的作业所需的文件和目录的其他文件组合在一起。有关任务捆绑包的更多信息，请参阅Deadline Cloud 的打开职位描述 (OpenJD) 模板。

使用作业提交文件

借助 Deadline Cloud，您可以启用作业工作流来访问工作主机的共享文件系统位置中不可用的输入文件。Job 附件允许渲染任务访问仅位于本地工作站驱动器或服务管理的队列环境中的文件。提交任务包

时，可以包括作业所需的输入文件和目录的列表。Deadline Cloud 识别这些非共享文件，将其从本地计算机上传到 Amazon S3，然后将其下载到工作主机。它简化了将输入资源传输到渲染节点的过程，确保分布式作业执行所需的所有文件均可访问。

您可以直接在作业捆绑包中为作业指定文件，使用使用环境变量或脚本提供的作业模板中的参数，以及使用作业的`assets_references`文件。您可以使用其中一种方法或三种方法的组合。您可以为任务的捆绑包指定存储配置文件，使其仅上传本地工作站上已更改的文件。

本节使用中的示例任务捆绑包 GitHub 来演示 Deadline Cloud 如何识别任务中要上传的文件、这些文件在 Amazon S3 中的组织方式，以及如何将它们提供给处理您任务的工作主机。

主题

Deadline Cloud 如何将文件上传到亚马逊 S3

此示例显示 Deadline Cloud 如何将文件从您的工作站或工作主机上传到 Amazon S3，以便共享这些文件。它使用来自的示例任务包 GitHub 和 Deadline Cloud CLI 来提交作业。

首先将 De adline Cloud 示例 GitHub存储库克隆到您的AWS CloudShell环境中，然后将`job_attachments_devguide`任务包复制到您的主目录中：

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
cp -r deadline-cloud-samples/job_bundles/job_attachments_devguide ~/
```

安装 De adline Cloud CLI 以提交工作捆绑包：

```
pip install deadline --upgrade
```

`job_attachments_devguide`任务包只有一个步骤，任务运行一个 bash shell 脚本，该脚本的文件系统位置作为作业参数传递。作业参数的定义是：

```
...
- name: ScriptFile
  type: PATH
```

```
  default: script.sh
  dataFlow: IN
  objectType: FILE
...
```

该dataFlow属性的IN值告诉作业附件，该ScriptFile参数的值是作业的输入。该default属性的值是作业包目录的相对位置，但也可以是绝对路径。此参数定义将作业包目录中的script.sh文件声明为作业运行所需的输入文件。

接下来，请确保 Deadline Cloud CLI 没有配置存储配置文件，然后将任务提交到队列Q1：

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id ''

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID
 job_attachments_devguide/
```

运行此命令后 Deadline Cloud CLI 的输出如下所示：

```
Submitting to Queue: Q1
...
Hashing Attachments  [####################################]  100%
Hashing Summary:
    Processed 1 file totaling 39.0 B.
    Skipped re-processing 0 files totaling 0.0 B.
    Total processing time of 0.0327 seconds at 1.19 KB/s.

Uploading Attachments  [####################################]  100%
Upload Summary:
    Processed 1 file totaling 39.0 B.
    Skipped re-processing 0 files totaling 0.0 B.
    Total processing time of 0.25639 seconds at 152.0 B/s.

Waiting for Job to be created...
Submitted job bundle:
   job_attachments_devguide/
Job creation completed successfully
job-74148c13342e4514b63c7a7518657005
```

当您提交任务时，Deadline Cloud 会先对`script.sh`文件进行哈希处理，然后将其上传到 Amazon S3。

Deadline Cloud 将 S3 存储桶视为内容可寻址的存储。文件上传到 S3 对象。对象名称源自文件内容的哈希值。如果两个文件的内容相同，则无论文件位于何处或名称如何，它们都具有相同的哈希值。这种内容寻址存储使得 Deadline Cloud 能够避免上传已经可用的文件。

您可以使用 AWS CLI 来查看上传到 Amazon S3 的对象：

```
# The name of queue `Q1`'s job attachments S3 bucket
Q1_S3_BUCKET=$(
  aws deadline get-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
    --query 'jobAttachmentSettings.s3BucketName' | tr -d '"'
)

aws s3 ls s3://$Q1_S3_BUCKET --recursive
```

两个对象已上传到 S3：

- `DeadlineCloud/Data/87cb19095dd5d78fcaf56384ef0e6241.xxh128`— 的内容`script.sh`。对象键`87cb19095dd5d78fcaf56384ef0e6241`中的值是文件内容的哈希值，扩展名`xxh128`表示哈希值是以 128 位 xx hash 计算得出的。
- `DeadlineCloud/Manifests/<farm-id>/<queue-id>/Inputs/<guid>/a1d221c7fd97b08175b3872a37428e8c_input`— 作业提交的清单对象。`<farm-id>`、`<queue-id>`和`<guid>`的值是您的服务器场标识符、队列标识符和随机十六进制值。此示例`a1d221c7fd97b08175b3872a37428e8c`中的值是根据字符串`/home/cloudshell-user/job_attachments_devguide`（所在`script.sh`目录）计算得出的哈希值。

清单对象包含作为任务提交的一部分上传到 S3 的特定根路径上的输入文件的信息。下载此清单文件（`aws s3 cp s3://$Q1_S3_BUCKET/<objectname>`）。其内容类似于：

```
{
    "hashAlg": "xxh128",
    "manifestVersion": "2023-03-03",
    "paths": [
        {
            "hash": "87cb19095dd5d78fcaf56384ef0e6241",
            "mtime": 1721147454416085,
            "path": "script.sh",
```

```
            "size": 39
        }
    ],
    "totalSize": 39
}
```

这表示文件script.sh已上传，该文件内容的哈希值为87cb19095dd5d78fcaf56384ef0e6241。此哈希值与对象名称中的值相匹配DeadlineCloud/Data/87cb19095dd5d78fcaf56384ef0e6241.xxh128。Deadline Cloud 使用它来知道要为该文件的内容下载哪个对象。

此文件的完整架构可在中找到 GitHub。

使用该CreateJob 操作时，您可以设置清单对象的位置。您可以使用该GetJob操作来查看位置：

```
{
    "attachments": {
        "file system": "COPIED",
        "manifests": [
            {
                "inputManifestHash": "5b0db3d311805ea8de7787b64cbbe8b3",
                "inputManifestPath": "<farm-id>/<queue-id>/Inputs/<guid>/
a1d221c7fd97b08175b3872a37428e8c_input",
                "rootPath": "/home/cloudshell-user/job_attachments_devguide",
                "rootPathFormat": "posix"
            }
        ]
    },
    ...
}
```

Deadline Cloud 如何选择要上传的文件

任务附件考虑上传到 Amazon S3 作为任务输入的文件和目录是：

- 在作业捆绑包的作业模板中定义的所有 PATH-type 作业参数的值，其dataFlow值为IN或INOUT。
- 在作业捆绑包的资产引用文件中作为输入列出的文件和目录。

如果您提交的工作没有存储配置文件，则会上传所有考虑上传的文件。如果您提交带有存储配置文件的任务，则如果文件位于存储配置文件的 SHARED-type 文件系统位置，而这些位置也是队列所需的文件

系统位置，则不会将其上传到 Amazon S3。这些位置预计将在运行任务的工作服务器主机上可用，因此无需将其上传到 S3。

在此示例中，您在 AWS CloudShell 环境WSAll中创建SHARED文件系统位置，然后将文件添加到这些文件系统位置。使用以下命令：

```
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

sudo mkdir -p /shared/common /shared/projects/project1 /shared/projects/project2
sudo chown -R cloudshell-user:cloudshell-user /shared

for d in /shared/common /shared/projects/project1 /shared/projects/project2; do
  echo "File contents for $d" > ${d}/file.txt
done
```

接下来，将资产引用文件添加到作业捆绑包中，该文件包含您作为作业输入创建的所有文件。使用以下命令：

```
cat > ${HOME}/job_attachments_devguide/asset_references.yaml << EOF
assetReferences:
  inputs:
    filenames:
    - /shared/common/file.txt
    directories:
    - /shared/projects/project1
    - /shared/projects/project2
EOF
```

接下来，将 Deadline Cloud CLI 配置为使用WSAll存储配置文件提交作业，然后提交任务捆绑包：

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id $WSALL_ID
```

```
deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID
 job_attachments_devguide/
```

当您提交任务时，Deadline Cloud 会将两个文件上传到 Amazon S3。您可以从 S3 下载任务的清单对象以查看上传的文件：

```
for manifest in $( \
  aws deadline get-job --farm-id $FARM_ID --queue-id $QUEUE1_ID --job-id $JOB_ID \
    --query 'attachments.manifests[].inputManifestPath' \
    | jq -r '.[]'
); do
  echo "Manifest object: $manifest"
  aws s3 cp --quiet s3://$Q1_S3_BUCKET/DeadlineCloud/Manifests/$manifest /dev/stdout |
 jq .
done
```

在此示例中，有一个包含以下内容的清单文件：

```
{
    "hashAlg": "xxh128",
    "manifestVersion": "2023-03-03",
    "paths": [
        {
            "hash": "87cb19095dd5d78fcaf56384ef0e6241",
            "mtime": 1721147454416085,
            "path": "home/cloudshell-user/job_attachments_devguide/script.sh",
            "size": 39
        },
        {
            "hash": "af5a605a3a4e86ce7be7ac5237b51b79",
            "mtime": 1721163773582362,
            "path": "shared/projects/project2/file.txt",
            "size": 44
        }
    ],
    "totalSize": 83
}
```

使用清单的GetJob 操作可以查看是否rootPath为 “/”。

```
aws deadline get-job --farm-id $FARM_ID --queue-id $QUEUE1_ID --job-id $JOB_ID --query
 'attachments.manifests[*]'
```

一组输入文件的根路径始终是这些文件中最长的公共子路径。如果您的作业是从中提交的，并且由于输入文件位于不同的驱动器上Windows而没有公共子路径，则您会在每个驱动器上看到一个单独的根路径。清单中的路径始终相对于清单的根路径，因此上传的输入文件为：

- /home/cloudshell-user/job_attachments_devguide/script.sh— 作业包中的脚本文件。
- /shared/projects/project2/file.txt— WSAll 存储配置SHARED文件中文件系统位置中不在队列所需文件系统位置列表中的文件Q1。

文件系统位置 FSCommon (/shared/common/file.txt) 和 FS1 (/shared/projects/project1/file.txt) 中的文件不在列表中。这是因为这些文件系统位置位于WSAll存储配置文件SHARED中，并且都在队列中所需的文件系统位置列表中Q1。

您可以看到在操作中使用特定存储配置文件提交的GetStorageProfileForQueue 作业所考虑SHARED的文件系统位置。要查询队列WSAll的存储配置文件，Q1请使用以下命令：

```
aws deadline get-storage-profile --farm-id $FARM_ID --storage-profile-id $WSALL_ID

aws deadline get-storage-profile-for-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID --
storage-profile-id $WSALL_ID
```

作业如何查找工作附件输入文件

要使任务使用 Deadline Cloud 通过任务附件上传到 Amazon S3 的文件，您的任务需要这些文件可通过工作主机上的文件系统获得。当作业的会话在工作服务器主机上运行时，Deadline Cloud 会将作业的输入文件下载到工作服务器主机本地驱动器上的临时目录中，并将每个作业根路径的路径映射规则添加到其在本地驱动器上的文件系统位置。

在此示例中，在 AWS CloudShell 选项卡中启动 Deadline Cloud 工作者代理。让之前提交的所有作业完成运行，然后从日志目录中删除作业日志：

```
rm -rf ~/devdemo-logs/queue-*
```

以下脚本修改作业捆绑包以显示会话临时工作目录中的所有文件和路径映射规则文件的内容，然后使用修改后的捆绑包提交作业：

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
```

```
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id $WSALL_ID

cat > ~/job_attachments_devguide/script.sh << EOF
#!/bin/bash

echo "Session working directory is: \$(pwd)"
echo
echo "Contents:"
find . -type f
echo
echo "Path mapping rules file: \$1"
jq . \$1
EOF

cat > ~/job_attachments_devguide/template.yaml << EOF
specificationVersion: jobtemplate-2023-09
name: "Job Attachments Explorer"
parameterDefinitions:
- name: ScriptFile
  type: PATH
  default: script.sh
  dataFlow: IN
  objectType: FILE
steps:
- name: Step
  script:
    actions:
      onRun:
        command: /bin/bash
        args:
        - "{{Param.ScriptFile}}"
        - "{{Session.PathMappingRulesFile}}"
EOF

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID
 job_attachments_devguide/
```

在工作程序在您的 AWS CloudShell 环境中运行作业后，您可以查看作业的运行日志：

```
cat demoenv-logs/queue-*/session*.log
```

日志显示，会话中发生的第一件事是将作业的两个输入文件下载到工作器上：

```
2024-07-17 01:26:37,824 INFO ================================================
2024-07-17 01:26:37,825 INFO --------- Job Attachments Download for Job
2024-07-17 01:26:37,825 INFO ================================================
2024-07-17 01:26:37,825 INFO Syncing inputs using Job Attachments
2024-07-17 01:26:38,116 INFO Downloaded 142.0 B / 186.0 B of 2 files (Transfer rate:
 0.0 B/s)
2024-07-17 01:26:38,174 INFO Downloaded 186.0 B / 186.0 B of 2 files (Transfer rate:
 733.0 B/s)
2024-07-17 01:26:38,176 INFO Summary Statistics for file downloads:
Processed 2 files totaling 186.0 B.
Skipped re-processing 0 files totaling 0.0 B.
Total processing time of 0.09752 seconds at 1.91 KB/s.
```

接下来是作业`script.sh`运行的输出：

- 提交作业时上传的输入文件位于会话临时目录中名称以 “assetroot” 开头的目录下。
- 输入文件的路径已相对于 “assetroot” 目录重新定位，而不是相对于作业输入清单 () 的根路径。`"/"`
- 路径映射规则文件包含一个额外的规则，该规则会重新映射`"/"`到 “assetroot” 目录的绝对路径。

例如：

```
2024-07-17 01:26:38,264 INFO Output:
2024-07-17 01:26:38,267 INFO Session working directory is: /sessions/session-5b33f
2024-07-17 01:26:38,267 INFO
2024-07-17 01:26:38,267 INFO Contents:
2024-07-17 01:26:38,269 INFO ./tmp_xdhbsdo.sh
2024-07-17 01:26:38,269 INFO ./tmpdi00052b.json
2024-07-17 01:26:38,269 INFO ./assetroot-assetroot-3751a/shared/projects/project2/
file.txt
2024-07-17 01:26:38,269 INFO ./assetroot-assetroot-3751a/home/cloudshell-user/
job_attachments_devguide/script.sh
2024-07-17 01:26:38,269 INFO
2024-07-17 01:26:38,270 INFO Path mapping rules file: /sessions/session-5b33f/
tmpdi00052b.json
2024-07-17 01:26:38,282 INFO {
2024-07-17 01:26:38,282 INFO   "version": "pathmapping-1.0",
```

```
2024-07-17 01:26:38,282 INFO   "path_mapping_rules": [
2024-07-17 01:26:38,282 INFO     {
2024-07-17 01:26:38,282 INFO       "source_path_format": "POSIX",
2024-07-17 01:26:38,282 INFO       "source_path": "/shared/projects/project1",
2024-07-17 01:26:38,283 INFO       "destination_path": "/mnt/projects/project1"
2024-07-17 01:26:38,283 INFO     },
2024-07-17 01:26:38,283 INFO     {
2024-07-17 01:26:38,283 INFO       "source_path_format": "POSIX",
2024-07-17 01:26:38,283 INFO       "source_path": "/shared/common",
2024-07-17 01:26:38,283 INFO       "destination_path": "/mnt/common"
2024-07-17 01:26:38,283 INFO     },
2024-07-17 01:26:38,283 INFO     {
2024-07-17 01:26:38,283 INFO       "source_path_format": "POSIX",
2024-07-17 01:26:38,283 INFO       "source_path": "/",
2024-07-17 01:26:38,283 INFO       "destination_path": "/sessions/session-5b33f/
assetroot-assetroot-3751a"
2024-07-17 01:26:38,283 INFO     }
2024-07-17 01:26:38,283 INFO   ]
2024-07-17 01:26:38,283 INFO }
```

> **Note**
>
> 如果您提交的作业有多个具有不同根路径的清单，则每个根路径都有一个不同的 “assetroot” 命名目录。

如果您需要引用某个输入文件、目录或文件系统位置的重定位文件系统位置，则可以处理作业中的路径映射规则文件并自己执行重新映射，也可以将PATH类型作业参数添加到作业包中的作业模板中，然后将需要重新映射的值作为该参数的值传递。例如，以下示例修改任务捆绑包使其具有以下作业参数之一，然后提交以文件系统位置/shared/projects/project2为其值的作业：

```
cat > ~/job_attachments_devguide/template.yaml << EOF
specificationVersion: jobtemplate-2023-09
name: "Job Attachments Explorer"
parameterDefinitions:
- name: LocationToRemap
  type: PATH
steps:
- name: Step
  script:
    actions:
      onRun:
```

```
        command: /bin/echo
        args:
        - "The location of {{RawParam.LocationToRemap}} in the session is
 {{Param.LocationToRemap}}"
EOF

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID
 job_attachments_devguide/ \
  -p LocationToRemap=/shared/projects/project2
```

此作业运行的日志文件包含其输出：

```
2024-07-17 01:40:35,283 INFO Output:
2024-07-17 01:40:35,284 INFO The location of /shared/projects/project2 in the session
 is /sessions/session-5b33f/assetroot-assetroot-3751a
```

从作业中获取输出文件

此示例显示 Deadline Cloud 如何识别您的任务生成的输出文件，决定是否将这些文件上传到 Amazon S3，以及如何在工作站上获取这些输出文件。

在本示例中，使用job_attachments_devguide_output任务捆绑包而不是job_attachments_devguide任务捆绑包。首先，从克隆的 Deadline Cloud 示例 GitHub存储库中复制 AWS CloudShell 环境中的捆绑包：

```
cp -r deadline-cloud-samples/job_bundles/job_attachments_devguide_output ~/
```

此任务捆绑包和任务捆绑包之间的重要区别是在作业模板中添加了一个新的作业参数：job_attachments_devguide

```
...
parameterDefinitions:
...
- name: OutputDir
  type: PATH
  objectType: DIRECTORY
  dataFlow: OUT
  default: ./output_dir
  description: This directory contains the output for all steps.
...
```

参数的dataFlow属性具有值OUT。Deadline Cloud 使用值为OUT或的dataFlow作业参数的值INOUT作为作业的输出。如果将作为值传递给这类任务参数的文件系统位置重新映射到运行该作业的工作程序上的本地文件系统位置，则 Deadline Cloud 将在该位置查找新文件并将这些文件作为任务输出上传到 Amazon S3。

要了解其工作原理，请先在 AWS CloudShell 选项卡中启动 Deadline Cloud 工作器代理。让之前提交的所有作业完成运行。然后从日志目录中删除作业日志：

```
rm -rf ~/devdemo-logs/queue-*
```

接下来，使用此工作捆绑包提交作业。在你 CloudShell运行的工作线程之后，查看日志：

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id $WSALL_ID

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID ./
job_attachments_devguide_output
```

日志显示检测到一个文件作为输出并上传到 Amazon S3：

```
2024-07-17 02:13:10,873 INFO ------------------------------------------------
2024-07-17 02:13:10,873 INFO Uploading output files to Job Attachments
2024-07-17 02:13:10,873 INFO ------------------------------------------------
2024-07-17 02:13:10,873 INFO Started syncing outputs using Job Attachments
2024-07-17 02:13:10,955 INFO Found 1 file totaling 117.0 B in output directory: /
sessions/session-7efa/assetroot-assetroot-3751a/output_dir
2024-07-17 02:13:10,956 INFO Uploading output manifest to
 DeadlineCloud/Manifests/farm-0011/queue-2233/job-4455/step-6677/
task-6677-0/2024-07-17T02:13:10.835545Z_sessionaction-8899-1/
c6808439dfc59f86763aff5b07b9a76c_output
2024-07-17 02:13:10,988 INFO Uploading 1 output file to S3: s3BucketName/DeadlineCloud/
Data
2024-07-17 02:13:11,011 INFO Uploaded 117.0 B / 117.0 B of 1 file (Transfer rate: 0.0
 B/s)
2024-07-17 02:13:11,011 INFO Summary Statistics for file uploads:
Processed 1 file totaling 117.0 B.
```

```
Skipped re-processing 0 files totaling 0.0 B.
Total processing time of 0.02281 seconds at 5.13 KB/s.
```

日志还显示，Deadline Cloud 在 Amazon S3 存储桶中创建了一个新的清单对象，该存储桶配置为供队列中的任务附件使用Q1。清单对象的名称源自生成输出的任务的场、队列、作业、步骤、任务、时间戳和sessionaction标识符。下载此清单文件，查看 Deadline Cloud 将此任务的输出文件放在哪里：

```
# The name of queue `Q1`'s job attachments S3 bucket
Q1_S3_BUCKET=$(
  aws deadline get-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
    --query 'jobAttachmentSettings.s3BucketName' | tr -d '"'
)

# Fill this in with the object name from your log
OBJECT_KEY="DeadlineCloud/Manifests/..."

aws s3 cp --quiet s3://$Q1_S3_BUCKET/$OBJECT_KEY /dev/stdout | jq .
```

清单如下所示：

```
{
  "hashAlg": "xxh128",
  "manifestVersion": "2023-03-03",
  "paths": [
    {
      "hash": "34178940e1ef9956db8ea7f7c97ed842",
      "mtime": 1721182390859777,
      "path": "output_dir/output.txt",
      "size": 117
    }
  ],
  "totalSize": 117
}
```

这表明输出文件内容保存到 Amazon S3 的方法与保存任务输入文件的方式相同。与输入文件类似，输出文件存储在 S3 中，其对象名包含文件哈希值和前缀DeadlineCloud/Data。

```
$ aws s3 ls --recursive s3://$Q1_S3_BUCKET | grep 34178940e1ef9956db8ea7f7c97ed842
2024-07-17 02:13:11        117 DeadlineCloud/
Data/34178940e1ef9956db8ea7f7c97ed842.xxh128
```

你可以使用 Deadline Cloud 监控器或 Deadline Cloud CLI 将任务的输出下载到你的工作站：

```
deadline job download-output --farm-id $FARM_ID --queue-id $QUEUE1_ID --job-id $JOB_ID
```

提交的OutputDir作业中作业参数的值为./output_dir，因此输出将下载到作业捆绑包目录output_dir中名为的目录中。如果您将绝对路径或不同的相对位置指定为的值OutputDir，则输出文件将改为下载到该位置。

```
$ deadline job download-output --farm-id $FARM_ID --queue-id $QUEUE1_ID --job-id
 $JOB_ID
Downloading output from Job 'Job Attachments Explorer: Output'

Summary of files to download:
    /home/cloudshell-user/job_attachments_devguide_output/output_dir/output.txt (1
 file)

You are about to download files which may come from multiple root directories. Here are
 a list of the current root directories:
[0] /home/cloudshell-user/job_attachments_devguide_output
> Please enter the index of root directory to edit, y to proceed without changes, or n
 to cancel the download (0, y, n) [y]:

Downloading Outputs  [####################################]  100%
Download Summary:
    Downloaded 1 files totaling 117.0 B.
    Total download time of 0.14189 seconds at 824.0 B/s.
    Download locations (total file counts):
        /home/cloudshell-user/job_attachments_devguide_output (1 file)
```

使用依赖步骤中某个步骤中的文件

此示例说明了作业中的一个步骤如何访问同一作业中它所依赖的步骤的输出。

为了使一个步骤的输出可供另一个步骤使用，Deadline Cloud 向会话添加了其他操作，以便在会话中运行任务之前下载这些输出。你可以通过将这些步骤声明为需要使用输出的步骤的依赖关系来告诉它从哪些步骤下载输出。

在此示例中使用job_attachments_devguide_output任务捆绑包。首先，在您的 AWS CloudShell 环境中从克隆的 Deadline Cloud 示例 GitHub存储库中制作一份副本。对其进行修改以添加一个依赖步骤，该步骤仅在现有步骤之后运行并使用该步骤的输出：

```
cp -r deadline-cloud-samples/job_bundles/job_attachments_devguide_output ~/

cat >> job_attachments_devguide_output/template.yaml << EOF
- name: DependentStep
  dependencies:
  - dependsOn: Step
  script:
    actions:
      onRun:
        command: /bin/cat
        args:
        - "{{Param.OutputDir}}/output.txt"
EOF
```

使用此修改后的作业捆绑包创建的作业作为两个单独的会话运行，一个用于步骤 “Step” 中的任务，另一个用于步骤 “DependentStep” 中的任务。

首先在 CloudShell 选项卡中启动 Deadline Cloud 工作器代理。让之前提交的所有作业完成运行，然后从日志目录中删除作业日志：

```
rm -rf ~/devdemo-logs/queue-*
```

接下来，使用修改后的任务捆绑包提交job_attachments_devguide_output作业。等待它在您 CloudShell 环境中的工作器上完成运行。查看两个会话的日志：

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id $WSALL_ID

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID ./
job_attachments_devguide_output

# Wait for the job to finish running, and then:

cat demoenv-logs/queue-*/session-*
```

在名为的步骤中任务的会话日志中DependentStep，有两个单独的下载操作正在运行：

```
2024-07-17 02:52:05,666 INFO ================================================
2024-07-17 02:52:05,666 INFO --------- Job Attachments Download for Job
2024-07-17 02:52:05,667 INFO ================================================
2024-07-17 02:52:05,667 INFO Syncing inputs using Job Attachments
2024-07-17 02:52:05,928 INFO Downloaded 207.0 B / 207.0 B of 1 file (Transfer rate: 0.0
 B/s)
2024-07-17 02:52:05,929 INFO Summary Statistics for file downloads:
Processed 1 file totaling 207.0 B.
Skipped re-processing 0 files totaling 0.0 B.
Total processing time of 0.03954 seconds at 5.23 KB/s.

2024-07-17 02:52:05,979 INFO
2024-07-17 02:52:05,979 INFO ================================================
2024-07-17 02:52:05,979 INFO --------- Job Attachments Download for Step
2024-07-17 02:52:05,979 INFO ================================================
2024-07-17 02:52:05,980 INFO Syncing inputs using Job Attachments
2024-07-17 02:52:06,133 INFO Downloaded 117.0 B / 117.0 B of 1 file (Transfer rate: 0.0
 B/s)
2024-07-17 02:52:06,134 INFO Summary Statistics for file downloads:
Processed 1 file totaling 117.0 B.
Skipped re-processing 0 files totaling 0.0 B.
Total processing time of 0.03227 seconds at 3.62 KB/s.
```

第一个操作下载名为 “Step” 的步骤所使用的`script.sh`文件。第二个操作下载该步骤的输出。Deadline Cloud 使用该步骤生成的输出清单作为输入清单来确定要下载哪些文件。

在同一篇日志的后面，你可以看到名为 DependentStep “” 的步骤的输出：

```
2024-07-17 02:52:06,213 INFO Output:
2024-07-17 02:52:06,216 INFO Script location: /sessions/session-5b33f/
assetroot-assetroot-3751a/script.sh
```

为作业设置资源限制

提交到 Deadline Cloud 的作业可能取决于多个作业之间共享的资源。例如，对于特定资源，农场的工人人数可能多于浮动许可证。或者，共享文件服务器可能只能同时向有限数量的工作人员提供数据。在某些情况下，一个或多个作业可能会占用所有这些资源，从而在新员工开始工作时由于资源不可用而导致错误。

为了帮助解决这个问题，你可以对这些受限的资源使用限制。Deadline Cloud 考虑了受限资源的可用性，并使用这些信息来确保在新员工启动时资源可用，从而降低由于资源不可用而导致工作失败的可能性。

为整个服务器场创建了限制。提交到队列的作业只能获得与队列关联的限制。如果您为与队列无关的作业指定限制，则该作业不兼容且无法运行。

要使用限制，您

- 创建限制
- 关联限制和队列
- 提交需要限制的职位

> **Note**
>
> 如果您在与限制无关的队列中运行一个资源受限的作业，则该作业可能会消耗所有资源。如果您的资源受限，请确保使用该资源的队列中作业中的所有步骤都与限制相关联。

对于在服务器场中定义的限制、与队列关联的限制以及在作业中指定的限制，可能会发生以下四种情况之一：

- 如果您创建了限制，将其与队列关联并在作业的模板中指定了限制，则该作业将运行并仅使用限制中定义的资源。
- 如果您创建了限制，在作业模板中指定该限制，但不要将限制与队列相关联，则该作业将被标记为不兼容且无法运行。
- 如果您创建了限制，请勿将其与队列关联，也未在作业模板中指定限制，则作业会运行，但不会使用该限制。
- 如果您根本不使用限制，则作业就会运行。

如果您将限制关联到多个队列，则这些队列将共享受该限制约束的资源。例如，如果您创建的限制为 100，而一个队列使用 60 个资源，则其他队列只能使用 40 个资源。资源被释放后，任务可以从任何队列中获取该资源。

Deadline Cloud 提供了两个AWS CloudFormation指标来帮助您监控限额提供的资源。您可以监控当前使用的资源数量以及限制中可用的最大资源数量。有关更多信息，请参阅 De adline Cloud 开发者指南中的资源限制指标。

您可以对作业模板中的作业步骤应用限制。当您在步骤的amounts部分中指定限制的金额要求名称，并且与任务队列关联的限制与该限制关联时，为该步骤安排的任务将受到资源限制的限制。hostRequirements amountRequirementName

如果某个步骤需要的资源受到达到限制的限制，则该步骤中的任务将不会由其他工作人员接管。

您可以对一个任务步骤应用多个限制。例如，如果该步骤使用两个不同的软件许可证，则可以为每个许可证应用单独的限制。如果一个步骤需要两个限制，并且已达到其中一个资源的限制，则在资源可用之前，该步骤中的任务不会被其他工作人员接管。

停止和删除限制

当您停止或删除队列与限制之间的关联时，使用该限制的作业会停止从需要此限制的步骤中调度任务，并阻止为步骤创建新会话。

处于 READY 状态的任务仍处于就绪状态，任务会自动恢复，队列和限制之间的关联再次变为活动状态。您无需重新排队任何作业。

停止或删除队列与限制之间的关联时，在如何停止运行任务方面有两种选择：

- 停止和取消任务 — 会话达到限制的工作人员会取消所有任务。
- 停止并完成正在运行的任务 — 会话达到限制的工作人员完成任务。

使用控制台删除限制时，工作人员首先会立即停止运行任务，或者最终在任务完成后停止运行。删除关联后，会发生以下情况：

- 需要限制的步骤标记为 “不兼容”。
- 包含这些步骤的整个任务都将被取消，包括不需要限制的步骤。
- 该作业被标记为不兼容。

如果与限制关联的队列的关联队列具有与限制的数量要求名称相匹配的队列，则该队列将继续处理具有指定限制的任务。

创建限制

您可以使用 Deadline Cloud 控制台或 Deadline C loud API 中的CreateLimit 操作来创建限制。限制是为服务器场定义的，但与队列相关联。创建限制后，您可以将其与一个或多个队列关联。

要创建限制

1. 从 Deadline Clou d 控制台（Deadline Cloud 控制台）控制台中，选择要为其创建队列的场。
2. 选择要添加限制的场，选择 “限制” 选项卡，然后选择 “创建限制”。
3. 提供限制的详细信息。金额要求名称是作业模板中用来标识限额的名称。它必须以前缀开头，**amount.**后跟金额名称。金额要求名称在与限额关联的队列中必须是唯一的。
4. 如果您选择 “设置最大数量”，则即该限制允许的资源总数。如果选择 “无最大数量”，则资源使用量不受限制。即使资源使用不受限制，也会发布 `CurrentCount` Amazon CloudWatch 指标，以便您可以跟踪使用情况。有关更多信息，请参阅 De adline Cloud 开发者指南中的CloudWatch指标。
5. 如果您已经知道应该使用限制的队列，则可以立即选择它们。您无需关联队列即可创建限制。
6. 选择 “创建限制”。

关联限制和队列

创建限制后，您可以将一个或多个队列与限制相关联。只有与限制关联的队列才使用限制中指定的值。

您可以使用 Deadline Cloud 控制台或 Deadline Cloud API 中的CreateQueueLimitAssociation 操作创建与队列的关联。

将队列与限制相关联

1. 从 Deadline Cl oud 控制台（Deadline Cloud 控制台）控制台中，选择要将限制与队列关联的场。
2. 选择 “限制” 选项卡，选择要与队列关联的限制，然后选择 “编辑限制”。
3. 在关联队列部分中，选择要与限制关联的队列。
4. 选择保存更改。

提交需要限制的职位

您可以通过将其指定为作业或作业步骤的主机要求来应用限制。如果您未在步骤中指定限制，并且该步骤使用关联的资源，则该步骤的使用量不会计入计划作业时的限制。

某些 Deadline Cloud 提交者允许您设置主机要求。您可以在提交者中指定限额的金额要求名称以应用限额。

如果您的提交者不支持添加主持人要求，您也可以通过编辑职位的作业模板来应用限制。

对任务捆绑包中的任务步骤应用限制

1. 使用文本编辑器打开作业模板。作业模板位于作业的作业捆绑包目录中。有关更多信息，请参阅 De adline Cloud 开发者指南中的 Job 捆绑包。
2. 找到要应用限制的步骤的步骤定义。
3. 将以下内容添加到步骤定义中。*amount.name*替换为限额的金额要求名称。对于典型用法，应将该`min`值设置为 1。

   YAML

   ```
     hostRequirements:
       amounts:
       - name: amount.name
         min: 1
   ```

   JSON

   ```
   "hostRequirements": {
       "amounts": [
           {
               "name": "amount.name",
               "min": "1"
           }
       ]
   }
   ```

   您可以按如下方式向任务步骤添加多个限制。*amount.name_2*用限额的金额要求名称替换*amount.name_1*和。

   YAML

   ```
     hostRequirements:
       amounts:
       - name: amount.name_1
         min: 1
       - name: amount.name_2
         min: 1
   ```

JSON

```
"hostRequirements": {
    "amounts": [
        {
            "name": "amount.name_1",
            "min": "1"
        },
        {
            "name": "amount.name_2",
            "min": "1"
        }
    ]
}
```

4. 保存对作业模板的更改。

如何向 Deadline Cloud 提交工作

向 De AWS adline Cloud 提交作业的方式有很多。本节介绍使用Deadline Cloud提供的工具或为工作负载创建自己的自定义工具来提交作业的一些方法。

- 从终端——当你第一次开发任务捆绑包时，或者当用户可以轻松地使用命令行提交作业时
- 来自脚本 — 用于自定义和自动化工作负载
- 来自应用程序 — 当用户的工作在应用程序中时，或者当应用程序的上下文很重要时。

以下示例使用 `deadline` Python 库和`deadline`命令行工具。两者均可从中获得 PyPi，并托管在上 GitHub。

主题

- 从终端向 Deadline Cloud 提交作业
- 使用脚本向 Deadline Cloud 提交作业
- 在申请中提交工作

从终端向 Deadline Cloud 提交作业

仅使用任务包和 Deadline Cloud CLI，您或您的技术含量更高的用户就可以快速迭代编写作业捆绑包来测试提交作业。使用以下命令提交任务捆绑包：

```
deadline bundle submit <path-to-job-bundle>
```

如果您提交的任务捆绑包的参数在捆绑包中没有默认值，则可以使用-p/--parameter选项指定它们。

```
deadline bundle submit <path-to-job-bundle> -p <parameter-name>=<parameter-value> -
p ...
```

要查看可用选项的完整列表，请运行 help 命令：

```
deadline bundle submit --help
```

使用 GUI 向 Deadline Cloud 提交作业

Deadline Cloud CLI 还带有图形用户界面，使用户能够在提交作业之前查看他们必须提供的参数。如果您的用户不想与命令行交互，则可以编写一个桌面快捷方式，打开一个对话框来提交特定的任务包：

```
deadline bundle gui-submit <path-to-job-bundle>
```

使用 c --browse an 选项，这样用户就可以选择任务捆绑包：

```
deadline bundle gui-submit --browse
```

要查看可用选项的完整列表，请运行 help 命令：

```
deadline bundle gui-submit --help
```

使用脚本向 Deadline Cloud 提交作业

要自动将作业提交到 Deadline Cloud，你可以使用 bash、Powershell 和批处理文件等工具编写作业脚本。

您可以添加诸如从环境变量或其他应用程序填充作业参数之类的功能。您也可以连续提交多个作业，或者编写要提交的任务捆绑包的创建脚本。

使用 Python 提交作业

Deadline Cloud 还有一个用于与该服务进行交互的开源 Python 库。源代码可在上找到 GitHub。

该库可通过 pip () `pip install deadline` 在 pypi 上使用。它与 Deadline Cloud CLI 工具使用的库相同：

```
from deadline.client import api

job_bundle_path = "/path/to/job/bundle"
job_parameters = [
    {
        "name": "parameter_name",
        "value": "parameter_value"
    },
]

job_id = api.create_job_from_job_bundle(
    job_bundle_path,
    job_parameters
)
print(job_id)
```

要创建类似`deadline bundle gui-submit`命令的对话框，您可以使用中的`show_job_bundle_submitter`函数`deadline.client.ui.job_bundle_submitter。`

以下示例启动 Qt 应用程序并显示任务包提交者：

```
# The GUI components must be installed with pip install "deadline[gui]"
import sys
from qtpy.QtWidgets import QApplication
from deadline.client.ui.job_bundle_submitter import show_job_bundle_submitter

app = QApplication(sys.argv)
submitter = show_job_bundle_submitter(browse=True)
submitter.show()
app.exec()
print(submitter.create_job_response)
```

要创建自己的对话框，你可以使用中的SubmitJobToDeadlineDialog类deadline.client.ui.dialogs.submit_job_to_deadline_di您可以传入值，嵌入自己的任务特定选项卡，并确定如何创建（或传入）任务包。

在申请中提交工作

为了便于用户提交作业，您可以使用应用程序提供的脚本运行时或插件系统。用户拥有熟悉的界面，您可以创建强大的工具来帮助用户提交工作负载。

在应用程序中嵌入作业捆绑包

此示例演示如何提交您在应用程序中提供的任务捆绑包。

要允许用户访问这些任务包，请创建一个嵌入在菜单项中的脚本来启动 Deadline Cloud CLI。

以下脚本允许用户选择作业捆绑包：

```
deadline bundle gui-submit --install-gui
```

要改用菜单项中的特定任务包，请使用以下命令：

```
deadline bundle gui-submit </path/to/job/bundle> --install-gui
```

这将打开一个对话框，用户可以在其中修改作业参数、输入和输出，然后提交作业。您可以为不同的工作捆绑包设置不同的菜单项，供用户在应用程序中提交。

如果您使用作业捆绑包提交的作业在提交时包含相似的参数和资产引用，则可以在底层作业捆绑包中填写默认值。

从应用程序获取信息

要从应用程序中提取信息，这样用户就不必手动将其添加到提交中，您可以将 Deadline Cloud 与应用程序集成，这样您的用户就可以使用熟悉的界面提交作业，而无需退出应用程序或使用命令行工具。

如果您的应用程序具有支持 Python 和 pyside/pyqt 的脚本运行时，则可以使用 De adline Cloud 客户端库中的 GUI 组件来创建用户界面。有关示例，请参阅上的 Maya 截止日期云集成 GitHub。

Deadline Cloud 客户端库提供的操作可执行以下操作，以帮助您提供强大的集成用户体验：

- 通过调用应用程序 SDK 从环境变量中提取队列环境参数、作业参数和资产引用。

- 在作业捆绑包中设置参数。为避免修改原始捆绑包，您应该制作捆绑包的副本并提交副本。

如果您使用`deadline bundle gui-submit`命令提交任务捆绑包，则必须以编程方式使用`parameter_values.yaml`和`asset_references.yaml`文件来传递来自应用程序的信息。有关这些文件的更多信息，请参阅Deadline Cloud 的打开职位描述 (OpenJD) 模板。

如果您需要比 OpenJD 提供的控件更复杂的控件，需要将作业从用户手中抽象出来，或者想要使集成与应用程序的视觉风格相匹配，则可以编写自己的对话框，调用 Deadline Cloud 客户端库来提交作业。

在截止日期云中安排作业

创建任务后，De AWS adline Cloud 会安排在与队列关联的一个或多个队列上对其进行处理。处理特定任务的队列是根据调度配置、为队列配置的功能以及特定步骤的主机要求来选择的。

以下各节详细介绍了安排作业的过程。

调度配置

您可以通过在队列上设置调度配置来配置 Deadline Cloud 如何调度队列中的作业。调度配置控制工作人员在作业中的分布方式。

您可以使用 Deadline Cloud 控制台或通过调用CreateQueue或来设置计划配置UpdateQueue APIs。

有三种可用的调度配置：

- 优先级， first-in-first-out(`priorityFifo`)-首先安排优先级最高、最早提交的作业（默认）。
- 优先级，平衡 (`priorityBalanced`)-在优先级最高的工作中平均分配员工。
- 加权，平衡 (`weightedBalanced`)-使用加权公式来确定员工在不同工作中的分布情况。

在所有调度配置中，正在进行的任务在做出新的调度决定之前都会运行到完成。如果您在任务运行时更改调度配置，则更改仅在接下来分配工作人员时适用。正在运行的任务不会中断或重新分配。

优先级， first-in-first-out

优先级， first-in-first-out(`priorityFifo`) 是新队列的默认调度配置。Deadline Cloud 会先将员工分配给优先级最高的工作。当多个作业具有相同的优先级时，最早（最早提交）的作业会首先接收所有可用工作人员。

当需要严格排序任务时，请使用优先级 FIFO。当作业应按提交顺序逐一完成时，此配置是合适的，例如连续的管道阶段或批处理，其中每个作业都必须在下一个作业开始之前完成。

此配置没有其他参数。

优先，平衡

Priority，balanced (`priorityBalanced`) 将员工平均分配给所有优先级最高的工作岗位。当只有一个优先级最高的工作存在时，Deadline Cloud 会将所有工作人员分配给该工作。当多个工作具有最高优先级时，员工将在其中平均分配。如果无法平均分配工人，则额外的工作人员将分配到最优先的工作中。

当多个艺术家或用户以相同的优先级提交工作并且每个用户都需要即时反馈时，请使用优先级平衡。此配置可确保没有一个作业垄断所有可用工作人员，因此在提交后不久就会为所有用户分配工作人员。

如果某项工作的剩余任务少于其工人所占比例，则剩余的工人将被重新分配到具有相同优先级别的其他工作。如果所有最高优先级的工作都被全部分配，剩余的工人就会流向下一个最高优先级别的工作。

此配置具有以下参数：

`renderingTaskBuffer`

控制工作人员的粘性。只有当渲染任务的差异超过该`renderingTaskBuffer`值时，工作人员才会从其当前作业切换到具有相同优先级的另一个作业。更高的值可以延长员工在当前工作的时间，从而减少上下文切换。默认值为 1。

加权、平衡

加权，平衡 (`weightedBalanced`) 使用公式计算每项作业的权重。Deadline Cloud 会先为员工分配权重最高的工作。如果多个工作具有相同的权重，则工人将在其中分配。

当您需要精细控制工作人员在优先级、错误率和提交时间各不相同的作业中的分布方式时，请使用加权平衡。此配置适用于复杂的渲染农场环境，在这些环境中，您需要调整作业优先级、作业年限、错误处理和工作人员粘性之间的平衡。

每项任务的权重计算方法如下：

```
weight = (job.Priority * priorityWeight) +
         (job.Errors * errorWeight) +
         ((currentTimeInSeconds - job.SubmissionTime) * submissionTimeWeight) +
```

```
        ((job.RenderingTasks - renderingTaskBuffer) * renderingTaskWeight)
```

只有当工作人员当前正在处理工作时，才会应用该renderingTaskBuffer组件。通常将其设置为负值，这样分配了工作人员的任务的权重就会降低，从而使其他任务排在队列的最前面。renderingTaskWeight通常errorWeight也是负数，因此出现错误的作业会被取消优先级。您可以对优先级最低和最高优先级的作业使用计划替代。

此配置具有以下参数：

priorityWeight

应用于作业优先级的权重。正值表示优先级较高的作业首先被安排。默认值为 100.0。射程：0到10000。

errorWeight

应用于作业错误计数的权重。负值表示首先计划没有错误的作业。默认值为 -10.0。射程：-10000到10000。

submissionTimeWeight

应用于作业提交时间的权重（以秒为单位）。正值表示先前提交的作业被安排在最前面。默认值为 3.0。射程：0到10000。

renderingTaskWeight

应用于当前为作业渲染的任务数量的权重。负值表示接下来会安排工人较少的工作。默认值为 -100.0。射程：-10000到10000。

renderingTaskBuffer

渲染任务权重生效之前的渲染任务数。正值可以让员工继续从事当前的工作。默认值为 1。射程：0到1000。

maxPriorityOverride

可选。如果设置为alwaysScheduleFirst，则无论加权公式如何，最高优先级 (100) 的任务始终排在其他作业之前。当多个任务具有最高优先级时，将使用标准加权公式打破平局。如果不存在改写，则最高优先级任务使用标准加权公式，不进行特殊处理。

minPriorityOverride

可选。如果设置为alwaysScheduleLast，则无论加权公式如何，最低优先级 (0) 的作业始终排在其他作业之后。当多个任务具有最低优先级时，将使用标准加权公式打破平局。如果不存在改写，则最低优先级任务使用标准加权公式，不进行特殊处理。

确定机队兼容性

创建任务后，Deadline Cloud 会根据与提交任务的队列关联的队列的能力来检查任务中每个步骤的主机要求。如果舰队符合主机要求，则该任务将进入该READY状态。

如果任务中的任何步骤具有与队列关联的队列无法满足的要求，则该步骤的状态将设置为NOT_COMPATIBLE。此外，作业中的其余步骤也将被取消。

舰队的能力是在舰队级别设置的。即使车队中的工人符合工作要求，如果其车队不符合工作要求，也不会从工作中为其分配任务。

以下作业模板的步骤指定了该步骤的主机要求：

```
name: Sample Job With Host Requirements
specificationVersion: jobtemplate-2023-09
steps:
- name: Step 1
  script:
    actions:
      onRun:
        args:
        - '1'
        command: /usr/bin/sleep
  hostRequirements:
    amounts:
    # Capabilities starting with "amount." are amount capabilities. If they start with
 "amount.worker.",
    # they are defined by the OpenJD specification. Other names are free for custom
 usage.
    - name: amount.worker.vcpu
      min: 4
      max: 8
    attributes:
    - name: attr.worker.os.family
      anyOf:
      - linux
```

可以将此任务安排给具有以下功能的舰队：

```
{
    "vCpuCount": {"min": 4, "max": 8},
    "memoryMiB": {"min": 1024},
```

```
    "osFamily": "linux",
    "cpuArchitectureType": "x86_64"
}
```

无法将此任务安排给具有以下任何功能的舰队：

```
{
    "vCpuCount": {"min": 4},
    "memoryMiB": {"min": 1024},
    "osFamily": "linux",
    "cpuArchitectureType": "x86_64"
}
    The vCpuCount has no maximum, so it exceeds the maximum vCPU host requirement.

{
    "vCpuCount": {"max": 8},
    "memoryMiB": {"min": 1024},
    "osFamily": "linux",
    "cpuArchitectureType": "x86_64"
}
    The vCpuCount has no minimum, so it doesn't satisfy the minimum vCPU host
 requirement.

{
    "vCpuCount": {"min": 4, "max": 8},
    "memoryMiB": {"min": 1024},
    "osFamily": "windows",
    "cpuArchitectureType": "x86_64"
}
    The osFamily doesn't match.
```

实例集扩展

将任务分配给兼容的服务托管队列时，队列会自动缩放。车队中的工作人员数量会根据可供车队运行的任务数量而变化。

将任务分配给客户管理的队列时，工作人员可能已经存在，或者可以使用基于事件的 auto Scaling 创建工作人员。有关更多信息，请参阅 Amazon EC2 Auto Scaling 用户指南中的用于 EventBridge处理自动扩展事件。

会话

作业中的任务分为一个或多个会话。工作人员运行会话来设置环境，运行任务，然后拆除环境。每个会话都由工作人员必须采取的一项或多项操作组成。

工作人员完成分区操作后，可以向该工作人员发送其他会话操作。工作人员在会话中重复使用现有环境和作业附件，以更高效地完成任务。

在服务管理的车队工作人员上，会话目录将在会话结束后删除，但其他目录将在会话之间保留。此行为允许您为可在多个会话中重复使用的数据实施缓存策略。要在会话之间缓存数据，请将其存储在运行作业的用户的主目录下。例如，conda 包缓存在作业用户的主目录下，位于Windows工作人员和Linux工作人员`C:\Users\job-user\.conda-pkgs`/`home/job-user/.conda-pkgs`上。在工作人员关闭之前，这些数据一直可用。

作业附件由提交者创建，您将其用作 Deadline Cloud CLI 任务捆绑包的一部分。您也可以使用`create-job` AWS CLI 命令的`--attachments`选项创建作业附件。环境在两个位置定义：附加到特定队列的队列环境以及作业模板中定义的作业和步骤环境。

有四种会话操作类型：

- `syncInputJobAttachments`— 将输入的作业附件下载给工作人员。
- `envEnter`— 对环境执行`onEnter`操作。
- `taskRun`— 执行任务的`onRun`操作。
- `envExit`— 对环境执行`onExit`操作。

以下作业模板具有步骤环境。它有一个`onEnter`用于设置步骤环境的`onRun`定义、一个定义要运行的任务的定义以及一个用于拆除步骤环境的`onExit`定义。为此作业创建的会话将包括一个`envEnter`操作、一个或多个`taskRun`操作，然后是一个`envExit`操作。

```
name: Sample Job with Maya Environment
specificationVersion: jobtemplate-2023-09
steps:
- name: Maya Step
  stepEnvironments:
  - name: Maya
    description: Runs Maya in the background.
    script:
      embeddedFiles:
      - name: initData
        filename: init-data.yaml
```

```
        type: TEXT
        data: |
          scene_file: MyAwesomeSceneFile
          renderer: arnold
          camera: persp
      actions:
        onEnter:
          command: MayaAdaptor
          args:
          - daemon
          - start
          - --init-data
          - file://{{Env.File.initData}}
        onExit:
          command: MayaAdaptor
          args:
          - daemon
          - stop
  parameterSpace:
    taskParameterDefinitions:
    - name: Frame
      range: 1-5
      type: INT
  script:
    embeddedFiles:
    - name: runData
      filename: run-data.yaml
      type: TEXT
      data: |
        frame: {{Task.Param.Frame}}
    actions:
      onRun:
        command: MayaAdaptor
        args:
        - daemon
        - run
        - --run-data
        - file://{{ Task.File.runData }}
```

会话操作流水线

会话操作流水线允许调度器将多个会话操作预先分配给工作人员。然后，工作人员可以按顺序运行这些操作，从而减少或消除任务之间的空闲时间。

要创建初始分配，调度器会创建一个包含一个任务的会话，工作人员完成任务，然后调度器分析任务持续时间以确定未来的分配。

为了使调度程序生效，有任务持续时间规则。对于一分钟以下的任务，调度程序使用 2 的乘方增长模式。例如，对于 1 秒钟的任务，调度器会分配 2 个新任务，然后分配 4 个新任务，然后分配 8 个新任务。对于超过一分钟的任务，调度程序仅分配一个新任务，管道传输仍处于禁用状态。

要计算管道大小，调度器会执行以下操作：

- 使用已完成任务的平均任务持续时间
- 旨在让员工忙一分钟
- 仅考虑同一会话中的任务
- 不在工作人员之间共享工期数据

通过会话操作流水线，工作人员可以立即开始新任务，并且在调度器请求之间没有等待时间。它还为长时间运行的流程提供了更高的员工效率和更好的任务分配。

此外，如果有新的更高优先级的作业可用，则工作人员将在当前会话结束之前完成先前分配的所有工作，并分配来自更高优先级作业的新会话。

步骤依赖关系

Deadline Cloud 支持定义步骤之间的依赖关系，以便一个步骤等到另一个步骤完成后再开始。您可以为一个步骤定义多个依赖关系。只有在所有依赖项都完成之后，才会安排具有依赖关系的步骤。

如果作业模板定义了循环依赖关系，则该作业将被拒绝，作业状态将设置为CREATE_FAILED。

以下作业模板创建了一个包含两个步骤的作业。 StepB取决于StepA。 StepB仅在成功StepA完成后运行。

作业创建后，StepA处于READY状态并StepB处于PENDING状态。StepA完成后，StepB移动到READY状态。如果StepA失败或已取消，则StepAStepB移至CANCELED状态。

您可以为多个步骤设置依赖关系。例如，如果同时StepC依赖StepA和StepB，StepC则要等到其他两个步骤完成后才会启动。

步骤依赖关系具有以下限制：

- 每个步骤的依赖关系-一个步骤最多可以依赖于 128 个其他步骤。
- 每个步骤的使用者 — 单个步骤最多可以有 32 个其他步骤。

```
name: Step-Step Dependency Test
specificationVersion: 'jobtemplate-2023-09'
steps:
- name: A
  script:
    actions:
      onRun:
        command: bash
        args: ['{{ Task.File.run }}']
    embeddedFiles:
      - name: run
        type: TEXT
        data: |
          #!/bin/env bash

          set -euo pipefail

          sleep 1
          echo Task A Done!
- name: B
  dependencies:
  - dependsOn: A # This means Step B depends on Step A
  script:
    actions:
      onRun:
        command: bash
        args: ['{{ Task.File.run }}']
    embeddedFiles:
      - name: run
        type: TEXT
        data: |
          #!/bin/env bash

          set -euo pipefail

          sleep 1
          echo Task B Done!
```

在截止日期云中修改作业

您可以使用以下 AWS Command Line Interface (AWS CLI) `update` 命令修改作业的配置，或者设置作业、步骤或任务的目标状态：

- aws deadline update-job
- aws deadline update-step
- aws deadline update-task

在以下命令示例中，将每个update命令替换*user input placeholder*为您自己的信息。

Example— 重新排队作业

除非存在步骤依赖关系，否则作业中的所有任务都会切换到READY状态。具有依赖关系的步骤在恢复时切换到任一READY或PENDING。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--target-task-run-status PENDING
```

Example— 取消作业

作业中所有没有状态SUCCEEDED或已标记FAILED的任务CANCELED。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--target-task-run-status CANCELED
```

Example— 将任务标记为失败

作业中所有处于该状态的任务SUCCEEDED都保持不变。所有其他任务都已标记FAILED。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--target-task-run-status FAILED
```

Example— 将工作标记为成功

作业中的所有任务都将变为SUCCEEDED状态。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--target-task-run-status SUCCEEDED
```

Example— 暂停作业

作业中处于SUCCEEDEDCANCELED、或FAILED状态的任务不会改变。所有其他任务都已标记SUSPENDED。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--target-task-run-status SUSPENDED
```

Example— 更改作业的优先级

更新队列中任务的优先级以更改其调度顺序。优先级较高的作业通常先安排。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--priority 100
```

Example— 更改允许的失败任务数

更新在取消剩余任务之前该任务可以执行的最大失败任务数。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--max-failed-tasks-count 200
```

Example— 更改允许的任务重试次数

更新任务失败前任务的最大重试次数。已达到最大重试次数的任务在增加该值之前无法重新排队。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--max-retries-per-task 10
```

Example— 存档作业

将作业的生命周期状态更新为ARCHIVED。无法安排或修改已存档的作业。您只能存档处于FAILED、CANCELEDSUCCEEDED、或SUSPENDED状态的作业。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--lifecycle-status ARCHIVED
```

Example— 更改作业的名称

更新作业的显示名称。任务名称的长度最多为 128 个字符。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--name "New Job Name"
```

Example— 更改职位描述

更新任务描述。描述的长度最多可为 2048 个字符。要删除现有描述，请传递一个空字符串。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--description "New Job Description"
```

Example— 重新排队步骤

除非存在步骤依赖关系，否则步骤中的所有任务都会切换到READY状态。具有依赖关系的步骤中的任务会切换到READY或PENDING，任务将恢复。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--target-task-run-status PENDING
```

Example— 取消步骤

步骤中所有没有状态SUCCEEDED或已标记FAILED的任务CANCELED。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--target-task-run-status CANCELED
```

Example— 将步骤标记为失败

步骤中所有状态为的任务SUCCEEDED均保持不变。所有其他任务都已标记FAILED。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--target-task-run-status FAILED
```

Example— 将步骤标记为成功

该步骤中的所有任务都已标记SUCCEEDED。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--target-task-run-status SUCCEEDED
```

Example— 暂停步骤

处于SUCCEEDEDCANCELED、或FAILED状态的步骤中的任务不会更改。所有其他任务都已标记SUSPENDED。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--target-task-run-status SUSPENDED
```

Example— 更改任务的状态

当您使用 De update-task adline Cloud CLI 命令时，任务会切换到指定的状态。

```
aws deadline update-task \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--task-id taskID \
--target-task-run-status SUCCEEDED | SUSPENDED | CANCELED | FAILED | PENDING
```

创建和使用 Deadline Cloud 客户管理的车队

创建客户管理的队列 (CMF) 时，您可以完全控制自己的处理管道。您可以为每位工作人员定义网络和软件环境。Deadline Cloud 充当作业的存储库和调度器。

工作人员可以是亚马逊弹性计算云 (Amazon EC2) 实例、托管设施中的工作人员或本地工作人员。每个工作人员都必须运行 Deadline Cloud 工作器代理。所有工作人员都必须有权访问 De adline Cloud 服务端点。

以下主题向您展示了如何使用 Amazon EC2 实例创建基本 CMF。

主题

创建由客户管理的车队

要创建客户管理的队列 (CMF)，请完成以下步骤。

Deadline Cloud console

使用 Deadline Cloud 控制台创建客户管理的舰队

1. 打开截止日期云控制台。
2. 选择 “农场”。将显示可用场列表。
3. 选择要在其中工作的农场的名称。
4. 选择 “舰队” 选项卡，然后选择 “创建舰队”。

5. 输入您的舰队的名称。
6. （可选）为您的舰队输入描述。
7. 为 “舰队类型” 选择 “客户管理”。
8. 选择您的车队的服务访问权限。
   a. 我们建议为每个队列使用 “创建并使用新的服务角色” 选项，以实现更精细的权限控制。已默认选定此选项。
   b. 您也可以通过选择 “选择服务角色” 来使用现有的服务角色。
9. 查看您的选择，然后选择 “下一步”。
10. 为您的舰队选择操作系统。车队的所有工作人员都必须使用通用的操作系统。
11. 选择主机 CPU 架构。
12. 选择最小和最大 vCPU 和内存硬件容量，以满足队列的工作负载需求。
13. 选择 Auto Scaling 类型。有关更多信息，请参阅用于 EventBridge 处理 Auto Scaling 事件。
    - 不扩展：你正在创建本地队列，想选择退出 Deadline Cloud Auto Scaling。
    - 扩展建议：您正在创建亚马逊弹性计算云 (Amazon EC2) 队列。
14. （可选）选择箭头以展开添加功能部分。
15. （可选）选中 “添加 GPU 功能-可选” 复选框，然后输入最小值 GPUs 和最大值以及内存。
16. 查看您的选择，然后选择 “下一步”。
17. （可选）定义自定义工作人员权能，然后选择下一步。
18. 使用下拉列表选择一个或多个要与队列关联的队列。

> **Note**
>
> 我们建议仅将队列与处于相同信任边界的队列相关联。此建议可确保在同一工作器上运行作业之间保持牢固的安全边界。

19. 查看队列关联，然后选择下一步。
20. （可选）对于默认 Conda 队列环境，我们将为您的队列创建一个环境，该环境将安装作业请求的 conda 软件包。

> **Note**
>
> conda 队列环境用于安装作业请求的 conda 软件包。通常，您应该取消选中与之关联的队列上的 conda 队列环境， CMFs 因为默认情况下 CMFs 不会安装所需的 conda 命令。

21. （可选）向 CMF 添加标签。有关更多信息，请参阅为AWS 资源添加标签。
22. 查看您的舰队配置并进行任何更改，然后选择创建舰队。
23. 选择 “舰队” 选项卡，然后记下舰队 ID。

AWS CLI

使用创建客户管理的车队 AWS CLI

1. 打开终端。
2. 在新编辑器`fleet-trust-policy.json`中创建。

   a. 添加以下 IAM 政策，将*ITALICIZED*文本替换为您的 AWS 账户 ID 和 Deadline Cloud Farm ID。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "credentials.deadline.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn":
 "arn:aws:deadline:*:111122223333:farm/FARM_ID"
                }
```

```
            }
        }
    ]
}
```

b. 保存更改。

3. 创建`fleet-policy.json`。

a. 添加以下 IAM 策略。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "deadline:AssumeFleetRoleForWorker",
                "deadline:UpdateWorker",
                "deadline:DeleteWorker",
                "deadline:UpdateWorkerSchedule",
                "deadline:BatchGetJobEntity",
                "deadline:AssumeQueueRoleForWorker"
            ],
            "Resource": "arn:aws:deadline:*:111122223333:*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalAccount": "${aws:ResourceAccount}"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*://deadline/*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalAccount": "${aws:ResourceAccount}"
                }
```

```
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:*:/aws/deadline/*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalAccount": "${aws:ResourceAccount}"
                }
            }
        }
    ]
}
```

b. 保存更改。

4. 添加 IAM 角色供队伍中的工作人员使用。

```
aws iam create-role --role-name FleetWorkerRoleName --assume-role-policy-
document file://fleet-trust-policy.json
aws iam put-role-policy --role-name FleetWorkerRoleName --policy-name
 FleetWorkerPolicy --policy-document file://fleet-policy.json
```

5. 创建create-fleet-request.json。

a. 添加以下 IAM 策略，用您的 CMF 值替换斜体文本。

> **Note**
>
> 你可以在*ROLE_ARN*里面找到create-cmf-fleet.json。
> 对于*OS_FAMILY*，您必须从macos或中选择一个windows。linux

```
{
    "farmId": "FARM_ID",
    "displayName": "FLEET_NAME",
    "description": "FLEET_DESCRIPTION",
    "roleArn": "ROLE_ARN",
    "minWorkerCount": 0,
```

```
    "maxWorkerCount": 10,
    "configuration": {
        "customerManaged": {
            "mode": "NO_SCALING",
            "workerCapabilities": {
                "vCpuCount": {
                    "min": 1,
                    "max": 4
                },
                "memoryMiB": {
                    "min": 1024,
                    "max": 4096
                },
                "osFamily": "OS_FAMILY",
                "cpuArchitectureType": "x86_64",
            },
        },
    }
}
```

b. 保存更改。

6. 创建您的舰队。

```
aws deadline create-fleet --cli-input-json file://create-fleet-request.json
```

工作主机设置和配置

工作主机是指运行 Deadline Cloud 工作程序的主机。本节介绍如何设置工作主机并根据您的特定需求对其进行配置。每台工作器主机都运行一个名为工作器代理的程序。工作人员代理负责：

- 管理工作人员的生命周期。
- 同步分配的工作、其进度和结果。
- 监控正在运行的工作。
- 将日志转发到已配置的目的地。

我们建议您使用提供的 Deadline Cloud 工作者代理。worker 代理是开源的，我们鼓励您提出功能请求，但您也可以根据自己的需求进行开发和定制。

要完成以下各节中的任务，您需要具备以下条件：

Linux

- Linux基于亚马逊弹性计算云 (Amazon EC2) 的实例。我们推荐亚马逊 Linux 2023。
- sudo特权
- Python 3.9 或更高版本

Windows

- Windows基于亚马逊弹性计算云 (Amazon EC2) 的实例。我们建议Windows Server 2022。
- 管理员对工作主机的访问权限
- 为所有用户安装了 Python 3.9 或更高版本

创建和配置 Python 虚拟环境

Linux如果您已经安装了 Python 3.9 或更高版本并将其放置在您的 Python 虚拟环境中，则可以在上创建 Python 虚拟环境PATH。

> **Note**
>
> 启用Windows，必须将代理文件安装到 Python 的全局站点包目录中。目前不支持 Python 虚拟环境。

创建和激活 Python 虚拟环境

1. 以root用户身份打开终端（或使用sudo/su）。
2. 创建并激活 Python 虚拟环境。

```
python3 -m venv /opt/deadline/worker
source /opt/deadline/worker/bin/activate
pip install --upgrade pip
```

安装 Deadline Cloud

设置 Python 并在上创建虚拟环境后，安装 Deadline Cloud 工作器代理 Python 软件包。Linux

安装工作器代理 Python 软件包

Linux

1. 以root用户身份打开终端（或使用sudo/su）。
2. 从 PyPI 下载并安装 Deadline Cloud 工作器代理包：

```
/opt/deadline/worker/bin/python -m pip install deadline-cloud-worker-agent
```

Windows

1. 打开管理员命令提示符或 PowerShell终端。
2. 从 PyPI 下载并安装 Deadline Cloud 工作器代理包：

```
python -m pip install deadline-cloud-worker-agent
```

当您的Windows工作主机需要长路径名（超过 250 个字符）时，必须按如下方式启用长路径名：

为Windows工作主机启用长路径

1. 确保已启用长路径注册表项。有关更多信息，请参阅在 Microsoft 网站上启用日志路径的注册表设置。
2. 安装适用于桌面 C++ x86 应用程序的Windows软件开发工具包。有关更多信息，请参阅Windows开发人员中心的 WindowsSDK。
3. 在安装工作器代理的环境中打开 Python 安装位置。默认值为 C:\Program Files\Python311。有一个名为的可执行文件pythonservice.exe。
4. 在同一位置创建一个pythonservice.exe.manifest名为的新文件。添加以下内容：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity type="win32" name="pythonservice" processorArchitecture="x86"
 version="1.0.0.0"/>
  <application xmlns="urn:schemas-microsoft-com:asm.v3">
```

```
    <windowsSettings>
      <longPathAware xmlns="http://schemas.microsoft.com/SMI/2016/
WindowsSettings">true</longPathAware>
    </windowsSettings>
  </application>
</assembly>
```

5. 打开命令提示符并在您创建的清单文件所在的位置运行以下命令：

```
"C:\Program Files (x86)\Windows Kits\10\bin\10.0.26100.0\x86\mt.exe" -manifest
 pythonservice.exe.manifest -outputresource:pythonservice.exe;#1
```

您应该可以看到类似于如下所示的输出内容：

```
Microsoft (R) Manifest Tool
Copyright (c) Microsoft Corporation.
All rights reserved.
```

现在，工作人员可以访问长路径了。要进行清理，请删除`pythonservice.exe.manifest`文件并卸载 SDK。

配置 Deadline 云端工作器代理

您可以通过三种方式配置 Deadline Cloud 工作器代理设置。我们建议您通过运行该`install-deadline-worker`工具来使用操作系统设置。

工作器代理不支持在 Windows 上以域用户身份运行。要以域用户身份运行作业，可以在为运行作业配置队列用户时指定域用户帐户。有关更多信息，请参阅 Deadline Cloud 用户指南中 Deadline Cloud 队列中的步骤 7。AWS

命令行参数-从命令行运行 Deadline Cloud 工作器代理时，可以指定参数。某些配置设置无法通过命令行参数获得。要查看所有可用的命令行参数，请输入`deadline-worker-agent --help`。

环境变量 — 您可以通过设置以开头的环境变量来配置 Deadline Cloud 工作器代理`DEADLINE_WORKER_`。例如，要查看所有可用的命令行参数，可以`export DEADLINE_WORKER_VERBOSE=true`用来将工作代理的输出设置为详细。有关更多示例和信息，请参阅 `/etc/amazon/deadline/worker.toml.example` on Linux 或 `C:\ProgramData\Amazon\Deadline\Config\worker.toml.example` on Windows。

配置文件-安装工作器代理时，它会创建一个位于 `/etc/amazon/deadline/worker.toml` on Linux 或 `C:\ProgramData\Amazon\Deadline\Config\worker.toml` on 的配置文件 Windows。工作器代理在启动时加载此配置文件。您可以使用示例配置文件（`/etc/amazon/deadline/worker.toml.example`开启Linux或开`C:\ProgramData\Amazon\Deadline\Config\worker.toml.example`启Windows）根据您的特定需求定制默认工作器代理配置文件。

最后，我们建议您在部署软件并按预期运行后为工作器代理启用 auto shutdown。这使工作人员队伍能够在需要时扩大规模，并在作业完成时关闭。自动缩放有助于确保您只使用所需的资源。要使由 auto Scaling 组启动的实例能够关闭，您必须将其`shutdown_on_stop=true`添加到`worker.toml`配置文件中。

启用自动关机

作为**`root`**用户：

- 安装带有参数的工作器代理**`--allow-shutdown`**。

  Linux

  输入：

  ```
  /opt/deadline/worker/bin/install-deadline-worker \
    --farm-id FARM_ID \
    --fleet-id FLEET_ID \
    --region REGION \
    --allow-shutdown
  ```

  Windows

  输入：

  ```
  install-deadline-worker ^
    --farm-id FARM_ID ^
    --fleet-id FLEET_ID ^
    --region REGION ^
    --allow-shutdown
  ```

创建作业用户和群组

本节介绍代理用户与队列中`jobRunAsUser`定义的用户之间所需的用户和组关系。

Deadline Cloud 工作服务器代理应在主机上以代理专用用户身份运行。您应配置 Deadline Cloud 队列的jobRunAsUser属性，以便工作人员以特定的操作系统用户和组的身份运行队列作业。此配置意味着您可以控制作业拥有的共享文件系统权限。它还提供了作业和工作代理用户之间的重要安全边界。

Linux工作用户和群组

要设置本地工作人员代理用户和jobRunAsUser，请确保满足以下要求。如果您使用的是 Linux 可插拔身份验证模块 (PAM)，例如 Active Directory 或 LDAP，则过程可能会有所不同。

工作器代理用户和共享jobRunAsUser组是在安装工作器代理时设置的。默认值为deadline-worker-agent和deadline-job-users，但可以在安装工作器代理时对其进行更改。

```
install-deadline-worker \
    --user AGENT_USER_NAME \
    --group JOB_USERS_GROUP
```

命令应以 root 用户身份运行。

- 每个组都jobRunAsUser应该有一个匹配的主组。使用adduser命令创建用户通常会创建匹配的主组。

  ```
  adduser -r -m jobRunAsUser
  ```

- 的主组jobRunAsUser是工作代理用户的辅助组。共享组允许工作器代理在作业运行时向其提供文件。

  ```
  usermod -a -G jobRunAsUser deadline-worker-agent
  ```

- jobRunAsUser必须是共享工作组的成员。

  ```
  usermod -a -G deadline-job-users jobRunAsUser
  ```

- jobRunAsUser不得属于工作代理用户的主组。工作器代理写入的敏感文件归代理的主组所有。如果 a jobRunAsUser 属于该组，则工作器上运行的作业可以访问工作器代理文件。
- 默认值 AWS 区域 必须与工作人员所属服务器场的区域相匹配。这应适用于工作人员的所有jobRunAsUser账户。

  ```
  sudo -u jobRunAsUser aws configure set default.region aws-region
  ```

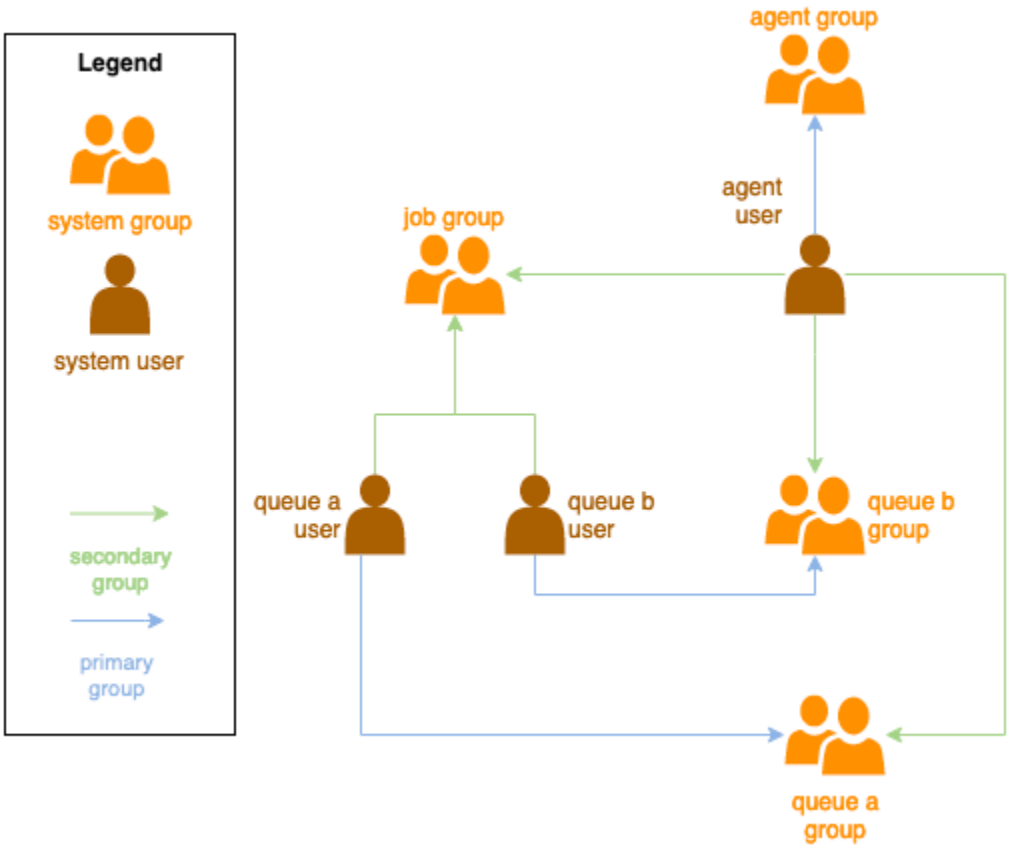
- 工作代理用户必须能够以. 的身份运行sudo命令jobRunAsUser。运行以下命令打开编辑器以创建新的 sudoers 规则：

```
visudo -f /etc/sudoers.d/deadline-worker-job-user
```

将以下内容添加到文件中：

```
# Allows the Deadline Cloud worker agent OS user to run commands
 # as the queue OS user without requiring a password.
deadline-worker-agent ALL=(jobRunAsUser) NOPASSWD:ALL
```

下图说明了代理用户与队列关联的jobRunAsUser用户和群组之间的关系。



Windows 用户

要使用Windows用户作为jobRunAsUser，它必须满足以下要求：

- 所有队列jobRunAsUser用户都必须存在。

- 他们的密码必须与队列JobRunAsUser字段中指定的密钥值相匹配。有关说明，请参阅 Deadline Cloud 用户指南中 Deadline Cloud 队列中的第 7 步。AWS
- 代理用户必须能够以这些用户的身份登录。

保护您的工作人员主机

设置工作主机时，请遵循安全最佳实践来保护敏感信息并保持适当的访问控制。

配置日志文件夹权限

工作器代理写入的日志文件可能包含来自主机配置脚本和作业执行的敏感信息。该install-deadline-worker命令使用安全权限创建日志目录。如果您需要在安装前手动创建目录，请使用以下过程来匹配服务管理队列使用的权限：

Linux

要在上配置日志目录权限 Linux

1. 创建日志目录：

```
sudo mkdir -p /var/log/amazon/deadline
```

2. 将所有者和群组设置为工作代理用户：

```
sudo chown -R deadline-worker:deadline-worker /var/log/amazon/deadline
```

3. 将权限设置为 750：

```
sudo chmod -R 750 /var/log/amazon/deadline
```

这些权限可确保只有工作代理用户和组才能访问日志文件，从而防止作业用户和其他未经授权的用户读取潜在的敏感信息。

Windows

要在上配置日志目录权限 Windows

1. 打开管理员 PowerShell 终端。
2. 创建日志目录：

```
New-Item -ItemType Directory -Force -Path "$env:PROGRAMDATA\Amazon\Deadline
\Logs"
```

3. 将限制配置 ACLs 为仅允许工作代理用户和管理员：

```
$acl = Get-Acl "$env:PROGRAMDATA\Amazon\Deadline\Logs"
$acl.SetAccessRuleProtection($true, $false)
$acl.Access | ForEach-Object { $acl.RemoveAccessRule($_) }
$agentRule = New-Object
 System.Security.AccessControl.FileSystemAccessRule("deadline-worker",
 "FullControl", "ContainerInherit,ObjectInherit", "None", "Allow")
$adminRule = New-Object
 System.Security.AccessControl.FileSystemAccessRule("Administrators",
 "FullControl", "ContainerInherit,ObjectInherit", "None", "Allow")
$acl.AddAccessRule($agentRule)
$acl.AddAccessRule($adminRule)
Set-Acl "$env:PROGRAMDATA\Amazon\Deadline\Logs" $acl
```

这些命令限制只有工作代理用户和管理员组才能访问日志目录，从而防止作业用户和其他未经授权的用户读取潜在的敏感信息。

管理对Windows工作用户密钥的访问权限

使用配置队列时 Windows`jobRunAsUser`，必须指定 Secrets Manager AWS 密钥。这个秘密的值应该是 JSON 编码的对象，其形式为：

```
{
    "password": "JOB_USER_PASSWORD"
}
```

要使工作人员按照队列的配置运行作业`jobRunAsUser`，队列的 IAM 角色必须具有获取密钥值的权限。如果使用客户管理的 KMS 密钥对密钥进行加密，则队列的 IAM 角色还必须具有使用 KMS 密钥进行解密的权限。

强烈建议对这些机密遵循最低权限原则。这意味着获取队列 `jobRunAsUser` `windows` → 的秘密值的访问权限`passwordArn`应为：

- 在舰队和队列之间创建队列队列关联时授予舰队角色
- 删除队列和队列之间的队列队列关联后，已从舰队角色中撤销

此外，当不再使用包含jobRunAsUser密码的 Secr AWS ets Manager 密钥时，应将其删除。

授予对密码密钥的访问权限

当队列和队列关联时，Dead jobRunAsUser line Cloud 舰队需要访问存储在队列密码密钥中的密码。我们建议使用 S AWS ecrets Manager 资源策略来授予对舰队角色的访问权限。如果您严格遵守此准则，则可以更轻松地确定哪些舰队角色可以访问该机密。

授予访问密钥的权限

1. 打开 AWS 密钥管理器控制台查看密钥。
2. 在资源权限部分，添加以下形式的政策声明：

   JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "arn:aws:iam::111122223333:role/FleetRole"
      },
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource" : "arn:aws:secretsmanager:us-
east-1:111122223333:secret:YourSecretName-ABC123"
    }
  ]
}
```

撤消对密码密钥的访问权限

当队列不再需要访问队列时，请移除对队列密码密钥的访问权限jobRunAsUser。我们建议使用 S AWS ecrets Manager 资源策略来授予对舰队角色的访问权限。如果您严格遵守此准则，则可以更轻松地确定哪些舰队角色可以访问该机密。

撤消对密钥的访问权限

1. 打开 AWS 密钥管理器控制台查看密钥。
2. 在资源权限部分中，删除以下形式的政策声明：

   JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "arn:aws:iam::111122223333:role/FleetRole"
      },
      "Action" : [
        "secretsmanager:GetSecretValue"
      ],
      "Resource" : "arn:aws:secretsmanager:us-
east-1:111122223333:secret:YourSecretName-ABC123"
    }
  ]
}
```

安装和配置作业所需的软件

设置 Deadline Cloud 工作器代理后，您可以为工作器主机准备运行作业所需的任何软件。

当您向关联的队列提交作业时`jobRunAsUser`，该作业将以该用户的身份运行。当提交作业时使用的命令不是绝对路径时，该命令必须在该用户的`PATH`中找到。

在 Linux 上，您可以在以下任一选项中`PATH`为用户指定：

- 他们的`~/.bashrc`或 `~/.bash_profile`
- 系统配置文件，例如`/etc/profile.d/*`和 `/etc/profile`
- shell 启动脚本：`/etc/bashrc`。

在 Windows 上，你可以在以下任一选项中`PATH`为用户指定：

- 他们特定于用户的环境变量

- 系统范围的环境变量

安装数字内容创作工具适配器

Deadline Cloud 为使用流行的数字内容创作 (DCC) 应用程序提供了 OpenJobDescription 适配器。要在客户管理的车队中使用这些适配器，必须安装 DCC 软件和应用程序适配器。然后，确保软件的可执行程序在系统搜索路径中可用（例如，在PATH环境变量中）。

在客户管理的车队上安装 DCC 适配器

1. 打开 a 终端。
   a. 在 Linux 上，以root用户身份打开终端（或使用sudo/su）
   b. 打开Windows，打开管理员命令提示符或 PowerShell 终端。
2. 安装 Deadline Cloud 适配器包。

```
pip install deadline deadline-cloud-for-maya deadline-cloud-for-nuke deadline-
cloud-for-blender deadline-cloud-for-3ds-max
```

配置 AWS 凭证

工作人员生命周期的初始阶段是自力更生。在此阶段，工作人员代理软件会在您的车队中创建一个工作人员，并从您的车队的角色中获取 AWS 凭证以进行进一步的操作。

AWS credentials for Amazon EC2

为具有 Deadline Cloud 工作人员主机权限的 Amazon EC2 创建 IAM 角色

1. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择导航窗格中的角色，然后选择创建角色。
3. 选择AWS 服务。
4. 选择 EC2 作为服务或用例，然后选择下一步。
5. 要授予必要的权限，请附加AWSDeadlineCloud-WorkerHost AWS 托管策略。

On-premises AWS credentials

您的本地员工使用凭据访问 Deadline Cloud。为了获得最安全的访问权限，我们建议使用 IAM Anywhere 角色对工作人员进行身份验证。有关更多信息，请参阅任何地方的 IAM 角色。

为了进行测试，您可以使用 IAM 用户访问密钥作为 AWS 证书。我们建议您通过包含限制性内联策略来为 IAM 用户设置过期时间。

> **Important**
>
> 请注意以下警告：
>
> - 请勿使用您账户的根凭证访问 AWS 资源。这些凭证可提供不受限的账户访问且难以撤销。
> - 不得在应用程序文件中按字面输入访问密钥或凭证信息。如果您这样做，则在将项目上传到公共存储库或在其他情况下，会有意外暴露凭证的风险。
> - 不得在项目区域中放入包含凭证的文件。
> - 保护您的访问密钥。请不要向未经授权方提供访问密钥，即便是为了帮助找到您的账户标识符也不行。通过这样做，您可以授予他人永久访问您的帐户的权限。
> - 请注意，存储在共享凭据文件中的所有 AWS 凭据都以纯文本形式存储。

有关更多详细信息，请参阅《AWS 一般参考》中的管理 AWS 访问密钥的最佳实践。

创建 IAM 用户

1. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择用户，然后选择创建用户。
3. 为用户命名。清除 “为用户提供访问权限” 复选框 AWS 管理控制台，然后选择 “下一步”。
4. 选择直接附加策略。
5. 从权限策略列表中选择策略，然后选择下一步。AWSDeadlineCloud-WorkerHost
6. 查看用户详细信息，然后选择创建用户。

将用户访问权限限制在有限的时间段内

您创建的任何 IAM 用户访问密钥都属于长期凭证。为了确保这些凭证在处理不当的情况下会过期，您可以创建内联策略来指定密钥失效的日期，从而限制这些凭证的使用时间。

1. 打开刚创建的 IAM 用户。在 “权限” 选项卡中，选择 “添加权限”，然后选择 “创建内联策略”。
2. 在 JSON 编辑器中，指定以下权限。要使用此策略，请将示例策略中的`aws:CurrentTime`时间戳值替换为您自己的时间和日期。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "DateGreaterThan": {
          "aws:CurrentTime": "2024-01-01T00:00:00Z"
        }
      }
    }
  ]
}
```

创建访问密钥

1. 在用户详细信息页面上，选择安全凭证选项卡。在访问密钥部分，选择创建访问密钥。
2. 指明您要将密钥用于 “其他”，然后选择 “下一步”，然后选择 “创建访问密钥”。
3. 在 “检索访问密钥” 页面上，选择 “显示” 以显示用户的私有访问密钥的值。您可以复制凭证或下载 .csv 文件。

存储用户访问密钥

- 将用户访问密钥存储在工作主机系统的代理用户 AWS 凭证文件中：
  - 开启Linux，文件位于 `~/.aws/credentials`
  - 开启Windows，文件位于 `%USERPROFILE\.aws\credentials`

  替换以下密钥：

```
[default]
aws_access_key_id=ACCESS_KEY_ID
aws_secret_access_key=SECRET_ACCESS_KEY
```

> **Important**
>
> 当您不再需要此 IAM 用户时，我们建议您将其删除，以符合AWS 安全最佳实践。我们建议您要求人类用户在访问AWS IAM Identity Center时使用临时证书 AWS。

工作人员为客户管理的车队托管数据流

本主题介绍了 De AWS adline Cloud（Deadline Cloud）工作服务器在操作期间建立的网络连接，包括联系的端点、使用的协议和传输的数据。此信息适用于客户管理的队列 (CMF) 工作人员，包括亚马逊弹性计算云 (Amazon EC2) 实例和本地工作人员。使用此信息为您的工作主机配置防火墙规则、创建 VPC 终端节点、执行安全审计或规划网络策略。有关服务托管舰队网络的信息，请参阅Inter-network 交通隐私。

所有工作人员通信仅限出站。工作主机启动所有连接，您无需允许任何入站连接。所有连接都使用端口 443 上的 HTTPS（TLS 1.2 或更高版本）。

本主题包括以下部分：

- the section called “端点和协议”
- the section called “工作人员使用的 API 操作”
- the section called “传输的其他数据”
- the section called “私有连接选项”

端点和协议

下表列出了工作主机在操作期间连接到的 AWS 服务端点。有关每项服务的区域终端节点的完整列表，请参阅AWS 一般参考中的服务终端节点和配额。

工作主机端点参考

AWS 服务	端点	端口/协议	用途	必需
截止日期云（日程安排）	scheduling.deadline. *[Region]*.amazonaws.com	443/HTTPS	工作人员注册、任务轮询、状态更新、证书交换、工作实体检索。请参阅the section called “工作人员使用的 API 操作”。	总是
Amazon CloudWatch 日志（CloudWatch 日志）	logs.*[Region]*.amazonaws.com	443/HTTPS	工作器代理和会话日志传输。	总是
Amazon Simple Storage Service（Amazon S3）	s3.*[Region]*.amazonaws.com	443/HTTPS	上传和下载 Job 附件。	如果使用作业附件

如果您的任务使用其他 AWS 服务，则可能还需要允许与这些服务终端节点的出站连接。

工作人员使用的 API 操作

以下所有 API 操作都使用scheduling.deadline.*[Region]*.amazonaws.com终端节点。有关每个操作的完整请求和响应架构，请参阅 De adline Cloud API 参考。

引导阶段

当工作器主机启动时，工作器代理会在队列中注册。引导凭证需要AWSDeadlineCloud-WorkerHost AWS 托管策略中的权限或等效的自定义权限。引导阶段使用以下 API 操作：

- CreateWorker— 在车队中注册工作人员。发送主机名和 IP 地址。收到工作人员 ID。
- AssumeFleetRoleForWorker— 获取舰队角色证书。接收工作器代理用于后续操作的临时 AWS 证书。

运营阶段

启动后，工作器代理会轮询工作和流程会话。舰队角色需要AWSDeadlineCloud-FleetWorker AWS 托管策略中的权限或等效的自定义权限，并使用以下 API 操作：

- UpdateWorker— 将工作器状态更新为关机STOPPED期间。
- UpdateWorkerSchedule— 对工作任务进行民意调查。发送会话操作状态更新，包括完成状态、进度百分比、进度消息和输出清单哈希值。接收分配的会话（作业 ID、队列 ID、会话操作、日志配置）、取消请求、所需的工作器状态和更新间隔。
- BatchGetJobEntity— 获取已分配工作的作业详细信息。发送作业实体标识符。接收作业详细信息、环境详细信息和作业附件详细信息。
- AssumeFleetRoleForWorker— 定期刷新舰队角色凭证。
- AssumeQueueRoleForWorker— 获取限定于特定队列的队列角色凭证。工作人员使用这些证书访问 Amazon S3 中的作业附件。

传输的其他数据

除了 Deadline Cloud 调度 API 操作外，工作主机还会将以下数据传输到其他 AWS 服务：

日志数据

工作器代理使用 API 操作将工作器代理日志和会话日志（作业进程中的 stdout 和 stderr）发送到 CloudWatch 日志。PutLogEvents

Job 附件

工作人员使用和 PutObject API 操作通过 Amazon S3 传输输入GetObject和输出文件。工作人员使用通过获取的队列角色凭证AssumeQueueRoleForWorker进行此访问。

遥测（可选）

工作代理发送操作指标，例如崩溃报告。您可以选择退出遥测采集。有关更多信息，请参阅 选择退出。

私有连接选项

您可以使用 AWS PrivateLink 在您的 VPC 内保持 CMF 工作线程主机和 Deadline Cloud 之间的流量，无需通过公共互联网。对于本地工作人员，您可以 AWS PrivateLink 与 AWS Direct Connect (Direct

Connect) 或 VPN 连接结合使用。有关更多信息，请参阅 访问 AWS Deadline Cloud 使用接口端点 (AWS PrivateLink)。

测试您的工作主机的配置

在安装了工作器代理、安装了处理任务所需的软件并配置了工作器代理的 AWS 凭据之后，在创建工作器代理之前，应测试安装是否可以处理您的作业 AMI 为了你的舰队。您应该测试以下内容：

- Deadline Cloud 工作代理已正确配置为作为系统服务运行。
- 工作人员对关联的工作队列进行轮询。
- 工作人员成功处理发送到与队列关联的队列的作业。

在测试配置并能够成功处理代表性作业之后，您可以使用已配置的工作器创建 AMI 适用于 Amazon EC2 员工，或者作为本地员工的榜样。

> **Note**
>
> 如果您正在测试 auto Scaling 队列的工作服务器主机配置，则在以下情况下可能难以测试您的工作服务器：
>
> - 如果队列中没有工作，Deadline Cloud 会在工作器启动后不久停止工作器代理。
> - 如果将工作器代理配置为在停止时关闭主机，则当队列中没有工作时，代理会关闭计算机。
>
> 为避免这些问题，请使用不自动缩放的暂存队列来配置和测试您的工作人员。测试工作服务器主机后，请务必在烘焙之前设置正确的队列 ID AMI.

测试您的工作器主机配置

1. 通过启动操作系统服务来运行工作器代理。

   Linux

   从根 shell 运行以下命令：

   ```
   systemctl start deadline-worker
   ```

Windows

通过管理员命令提示符或 PowerShell 终端，输入以下命令：

```
sc.exe start DeadlineWorker
```

2. 监控工作人员以确保其启动并轮询是否有工作。

Linux

从根 shell 运行以下命令：

```
systemctl status deadline-worker
```

该命令应返回如下响应：

```
Active: active (running) since Wed 2023-06-14 14:44:27 UTC; 7min ago
```

如果响应不是这样，请使用以下命令检查日志文件：

```
tail -n 25 /var/log/amazon/deadline/worker-agent.log
```

Windows

通过管理员命令提示符或 PowerShell 终端，输入以下命令：

```
sc.exe query DeadlineWorker
```

该命令应返回如下响应：

```
STATE   : 4 RUNNING
```

如果响应不包含RUNNING，请检查工作器日志文件。打开并管理员 PowerShell 提示并运行以下命令：

```
Get-Content -Tail 25 -Path $env:PROGRAMDATA\Amazon\Deadline\Logs\worker-
agent.log
```

3. 将任务提交到与您的队列关联的队列。这些工作应该代表车队处理的任务。

4. 使用 Deadline Cloud 监控器或 CLI 监控任务的进度。如果作业失败，请检查会话和工作器日志。
5. 根据需要更新工作主机的配置，直到作业成功完成。
6. 当测试作业成功后，您可以停止该工作人员：

   Linux

   从根 shell 运行以下命令：

   ```
   systemctl stop deadline-worker
   ```

   Windows

   通过管理员命令提示符或 PowerShell 终端，输入以下命令：

   ```
   sc.exe stop DeadlineWorker
   ```

创建一个 Amazon Machine Image

要创建 Amazon Machine Image (AMI) 要在亚马逊弹性计算云 (Amazon EC2) 客户管理的队列 (CMF) 中使用，请完成本节中的任务。在继续操作之前，您必须创建一个 Amazon EC2 实例。有关更多信息，请参阅《Amazon Linux 实例 EC2 用户指南》中的启动实例。

> **Important**
>
> 创建一个 AMI 创建 Amazon EC2 实例所连接卷的快照。实例上安装的所有软件都将保留，因此当您从实例启动实例时，这些实例会被重复使用 AMI。 我们建议采用修补策略，并定期更新任何新内容 AMI 在应用到您的车队之前，请使用更新的软件。

准备 Amazon EC2 实例

在你建一个之前 AMI，则必须删除工作器状态。在工作器代理启动之间，工作器状态保持不变。如果这种状态持续到 AMI，那么从它启动的所有实例都将共享相同的状态。

我们还建议您删除所有现有的日志文件。准备 AMI 时，日志文件可以保留在 Amazon EC2 实例上。在诊断使用 AMI 的工作队列中可能存在的问题时，删除这些文件可以最大限度地减少混乱。

您还应该启用工作代理系统服务，这样 Deadline Cloud 工作器代理在亚马逊启动 EC2 时启动。

最后，我们建议您启用工作器代理自动关机。这允许工作人员队列在需要时扩大规模，并在渲染作业完成时关闭。这种 auto Scaling 有助于确保您仅根据需要使用资源。

准备 Amazon EC2 实例

1. 打开 Amazon EC2 控制台。
2. 启动 Amazon EC2 实例。有关更多信息，请参阅启动您的实例。
3. 将主机设置为连接到您的身份提供商 (IdP)，然后挂载它需要的任何共享文件系统。
4. 然后，按照教程进行操作安装 Deadline Cloud，然后配置工作器代理，和创建作业用户和群组。
5. 如果你正在准备 AMI 基于 Amazon Linux 2023，要运行与视觉特效参考平台兼容的软件，您需要更新多项要求。有关信息，请参阅《De AWS adline Cloud 用户指南》中的 VFX 参考平台兼容性。
6. 打开终端。
   a. 在 Linux 上，以`root`用户身份打开终端（或使用`sudo`/`su`）
   b. On Windows，打开管理员命令提示符或 PowerShell 终端。
7. 确保 worker 服务未运行且配置为在启动时启动：
   a. 在 Linux 上，运行

```
systemctl stop deadline-worker
systemctl enable deadline-worker
```

   b. On Windows，运行

```
sc.exe stop DeadlineWorker
sc.exe config DeadlineWorker start= auto
```

8. 删除工作器状态。
   a. 在 Linux 上，运行

```
rm -rf /var/lib/deadline/*
```

   b. On Windows，运行

```
del /Q /S %PROGRAMDATA%\Amazon\Deadline\Cache\*
```

9. 删除日志文件。

a. 在 Linux 上，运行

```
rm -rf /var/log/amazon/deadline/*
```

b. On Windows，运行

```
del /Q /S %PROGRAMDATA%\Amazon\Deadline\Logs\*
```

10. On Windows，建议运行 “开始” 菜单中的 Amazon EC2 Launch Settings 应用程序，以完成实例的最终主机准备和关闭。

> **Note**
>
> 你必须选择 “不使用 Sysprep 关闭”，切勿选择 “使用 Sysprep 关闭”。使用 Sysprep 关闭将导致所有本地用户都无法使用。有关更多信息，请参阅《Windows 实例用户指南》的 “创建自定义 AMI” 主题的 “开始之前” 部分。

构建 AMI

要构建 AMI

1. 打开 Amazon EC2 控制台。
2. 在导航窗格中选择实例，然后选择您的实例。
3. 选择实例状态，然后选择停止实例。
4. 实例停止后，选择操作。
5. 选择图像和模板，然后选择创建图像。
6. 输入图像名称。
7. （可选）输入图片的描述。
8. 选择创建映像。

使用 Amazon EC2 Auto Scaling 组创建队列基础设施

本节介绍如何创建 Amazon EC2 Auto Scaling 队列。

使用下面的 CloudFormation YAML 模板创建一个 Amazon EC2 Auto Scaling (Auto Scaling) 组、一个包含两个子网、一个实例配置文件和一个实例访问角色的亚马逊虚拟私有云 (Amazon VPC)。这些是在子网中使用 Auto Scaling 启动实例所必需的。

您应该查看并更新实例类型列表以满足您的渲染需求。

有关 CloudFormation YAML 模板中使用的资源和参数的完整说明，请参阅《AWS CloudFormation 用户指南》中的 Deadl ine Cloud 资源类型参考。

创建 Amazon EC2 Auto Scaling 队列

1. 使用以下示例创建定义`FarmID`、`FleetID`、和`AMIId`参数的 CloudFormation 模板。将模板保存到本地计算机上的`.YAML`文件中。

```
AWSTemplateFormatVersion: 2010-09-09
Description: Amazon Deadline Cloud customer-managed fleet
Parameters:
  FarmId:
    Type: String
    Description: Farm ID
  FleetId:
    Type: String
    Description: Fleet ID
  AMIId:
    Type: String
    Description: AMI ID for launching workers
Resources:
  deadlineVPC:
    Type: 'AWS::EC2::VPC'
    Properties:
      CidrBlock: 100.100.0.0/16
  deadlineWorkerSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: !Join
        - ' '
        - - Security group created for Deadline Cloud workers in the fleet
          - !Ref FleetId
      GroupName: !Join
        - ''
        - - deadlineWorkerSecurityGroup-
          - !Ref FleetId
      SecurityGroupEgress:
```

```
      - CidrIp: 0.0.0.0/0
        IpProtocol: '-1'
    SecurityGroupIngress: []
    VpcId: !Ref deadlineVPC
deadlineIGW:
  Type: 'AWS::EC2::InternetGateway'
  Properties: {}
deadlineVPCGatewayAttachment:
  Type: 'AWS::EC2::VPCGatewayAttachment'
  Properties:
    VpcId: !Ref deadlineVPC
    InternetGatewayId: !Ref deadlineIGW
deadlinePublicRouteTable:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref deadlineVPC
deadlinePublicRoute:
  Type: 'AWS::EC2::Route'
  Properties:
    RouteTableId: !Ref deadlinePublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref deadlineIGW
  DependsOn:
    - deadlineIGW
    - deadlineVPCGatewayAttachment
deadlinePublicSubnet0:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref deadlineVPC
    CidrBlock: 100.100.16.0/22
    AvailabilityZone: !Join
      - ''
      - - !Ref 'AWS::Region'
        - a
deadlineSubnetRouteTableAssociation0:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    RouteTableId: !Ref deadlinePublicRouteTable
    SubnetId: !Ref deadlinePublicSubnet0
deadlinePublicSubnet1:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref deadlineVPC
    CidrBlock: 100.100.20.0/22
```

```
      AvailabilityZone: !Join
        - ''
        - - !Ref 'AWS::Region'
          - c
  deadlineSubnetRouteTableAssociation1:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
      RouteTableId: !Ref deadlinePublicRouteTable
      SubnetId: !Ref deadlinePublicSubnet1
  deadlineInstanceAccessAccessRole:
    Type: 'AWS::IAM::Role'
    Properties:
      RoleName: !Join
        - '-'
        - - deadline
          - InstanceAccess
          - !Ref FleetId
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action:
              - 'sts:AssumeRole'
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy'
        - 'arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore'
        - 'arn:aws:iam::aws:policy/AWSDeadlineCloud-WorkerHost'
  deadlineInstanceProfile:
    Type: 'AWS::IAM::InstanceProfile'
    Properties:
      Path: /
      Roles:
        - !Ref deadlineInstanceAccessAccessRole
  deadlineLaunchTemplate:
    Type: 'AWS::EC2::LaunchTemplate'
    Properties:
      LaunchTemplateName: !Join
        - ''
        - - deadline-LT-
          - !Ref FleetId
      LaunchTemplateData:
        NetworkInterfaces:
```

```
          - DeviceIndex: 0
            AssociatePublicIpAddress: true
            Groups:
              - !Ref deadlineWorkerSecurityGroup
            DeleteOnTermination: true
        ImageId: !Ref AMIId
        InstanceInitiatedShutdownBehavior: terminate
        IamInstanceProfile:
          Arn: !GetAtt
            - deadlineInstanceProfile
            - Arn
        MetadataOptions:
          HttpTokens: required
          HttpEndpoint: enabled

  deadlineAutoScalingGroup:
    Type: 'AWS::AutoScaling::AutoScalingGroup'
    Properties:
      AutoScalingGroupName: !Join
        - ''
        - - deadline-ASG-autoscalable-
          - !Ref FleetId
      MinSize: 0
      MaxSize: 10
      VPCZoneIdentifier:
        - !Ref deadlinePublicSubnet0
        - !Ref deadlinePublicSubnet1
      NewInstancesProtectedFromScaleIn: true
      MixedInstancesPolicy:
        InstancesDistribution:
          OnDemandBaseCapacity: 0
          OnDemandPercentageAboveBaseCapacity: 0
          SpotAllocationStrategy: capacity-optimized
          OnDemandAllocationStrategy: lowest-price
        LaunchTemplate:
          LaunchTemplateSpecification:
            LaunchTemplateId: !Ref deadlineLaunchTemplate
            Version: !GetAtt
              - deadlineLaunchTemplate
              - LatestVersionNumber
          Overrides:
            - InstanceType: m5.large
            - InstanceType: m5d.large
            - InstanceType: m5a.large
```

```
            - InstanceType: m5ad.large
            - InstanceType: m5n.large
            - InstanceType: m5dn.large
            - InstanceType: m4.large
            - InstanceType: m3.large
            - InstanceType: r5.large
            - InstanceType: r5d.large
            - InstanceType: r5a.large
            - InstanceType: r5ad.large
            - InstanceType: r5n.large
            - InstanceType: r5dn.large
            - InstanceType: r4.large
      MetricsCollection:
        - Granularity: 1Minute
          Metrics:
            - GroupMinSize
            - GroupMaxSize
            - GroupDesiredCapacity
            - GroupInServiceInstances
            - GroupTotalInstances
            - GroupInServiceCapacity
            - GroupTotalCapacity
```

2. 在 https://console.aws.amazon.com/cloudformation 上打开 CloudFormation 控制台。

   使用 CloudFormation 控制台按照上传您创建的模板文件的说明创建堆栈。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的在 CloudFormation 控制台上创建堆栈。

> **Note**
>
> - 附加到您的工作人员的 Amazon EC2 实例的 IAM 角色的证书可用于在该工作程序上运行的所有进程，包括作业。工作人员应拥有最少的操作权限：`deadline:CreateWorker`和`deadline:AssumeFleetRoleForWorker.`
> - 工作器代理获取队列角色的凭证，并对其进行配置以供运行作业使用。Amazon EC2 实例配置文件角色不应包含您的任务所需的权限。

使用 Deadline Cloud 规模推荐功能自动扩展您的 Amazon EC2 机群

Deadline Cloud 利用亚马逊 EC2 Auto Scaling（Auto Scaling）组自动扩展亚马逊 EC2 客户管理的队列 (CMF)。您需要配置舰队模式并在您的账户中部署所需的基础架构，以实现队列自动扩展。您部署的基础架构将适用于所有舰队，因此您只需要设置一次即可。

基本工作流程是：将舰队模式配置为 auto scale，然后，每当建议的舰队规模 EventBridge 发生变化时，Deadline Cloud 就会为该舰队发送一个事件（一个事件包含舰队 ID、推荐的舰队规模和其他元数据）。您将有一 EventBridge 条规则来筛选相关事件，并使用 Lambda 来使用它们。Lambda 将与亚马逊 EC2 Auto Scaling 集成`AutoScalingGroup`，以自动扩展亚马逊 EC2 队列。

将舰队模式设置为 **EVENT_BASED_AUTO_SCALING**

将您的舰队模式配置为`EVENT_BASED_AUTO_SCALING`. 您可以使用控制台来执行此操作，也可以使用直接调 AWS CLI 用`CreateFleet`或 `UpdateFleet` API。配置模式后，每当建议的队列规模发生变化时，Deadline Cloud 就会开始发送 EventBridge事件。

- `UpdateFleet`命令示例：

```
aws deadline update-fleet \
  --farm-id FARM_ID \
  --fleet-id FLEET_ID \
  --configuration file://configuration.json
```

- `CreateFleet`命令示例：

```
aws deadline create-fleet \
  --farm-id FARM_ID \
  --display-name "Fleet name" \
  --max-worker-count 10 \
  --configuration file://configuration.json
```

以下是在上述 CLI 命令中`configuration.json`使用的示例（`--configuration file://configuration.json`）。

- 要在队列上启用 Auto Scaling，应将模式设置为`EVENT_BASED_AUTO_SCALING`。
- `workerCapabilities`这些是您创建 CMF 时分配给它的默认值。如果您需要增加 CMF 的可用资源，可以更改这些值。

配置队列模式后，Deadline Cloud 开始为该队列发出舰队规模建议事件。

```
{
    "customerManaged": {
        "mode": "EVENT_BASED_AUTO_SCALING",
        "workerCapabilities": {
            "vCpuCount": {
                "min": 1,
                "max": 4
            },
            "memoryMiB": {
                "min": 1024,
                "max": 4096
            },
            "osFamily": "linux",
            "cpuArchitectureType": "x86_64"
        }
    }
}
```

使用 CloudFormation 模板部署 Auto Scaling 堆栈

您可以设置 EventBridge 规则来筛选事件，设置用于使用事件和控制 Auto Scaling 的 Lambda，以及用于存储未处理事件的 SQS 队列。使用以下 CloudFormation 模板部署堆栈中的所有内容。成功部署资源后，您可以提交任务，队列将自动扩展。

```
Resources:
  AutoScalingLambda:
    Type: 'AWS::Lambda::Function'
    Properties:
      Code:
        ZipFile: |-
          """
          This lambda is configured to handle "Fleet Size Recommendation Change"
          messages. It will handle all such events, and requires
          that the ASG is named based on the fleet id. It will scale up/down the fleet
          based on the recommended fleet size in the message.

          Example EventBridge message:
          {
              "version": "0",
              "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
              "detail-type": "Fleet Size Recommendation Change",
```

```
            "source": "aws.deadline",
            "account": "111122223333",
            "time": "2017-12-22T18:43:48Z",
            "region": "us-west-1",
            "resources": [],
            "detail": {
                "farmId": "farm-12345678900000000000000000000000",
                "fleetId": "fleet-12345678900000000000000000000000",
                "oldFleetSize": 1,
                "newFleetSize": 5,
            }
        }
        """

        import json
        import boto3
        import logging

        logger = logging.getLogger()
        logger.setLevel(logging.INFO)

        auto_scaling_client = boto3.client("autoscaling")

        def lambda_handler(event, context):
            logger.info(event)
            event_detail = event["detail"]
            fleet_id = event_detail["fleetId"]
            desired_capacity = event_detail["newFleetSize"]

            asg_name = f"deadline-ASG-autoscalable-{fleet_id}"
            auto_scaling_client.set_desired_capacity(
                AutoScalingGroupName=asg_name,
                DesiredCapacity=desired_capacity,
                HonorCooldown=False,
            )

            return {
                'statusCode': 200,
                'body': json.dumps(f'Successfully set desired_capacity for {asg_name}
to {desired_capacity}')
            }
    Handler: index.lambda_handler
    Role: !GetAtt
      - AutoScalingLambdaServiceRole
```

```
        - Arn
      Runtime: python3.11
    DependsOn:
      - AutoScalingLambdaServiceRoleDefaultPolicy
      - AutoScalingLambdaServiceRole
  AutoScalingEventRule:
    Type: 'AWS::Events::Rule'
    Properties:
      EventPattern:
        source:
          - aws.deadline
        detail-type:
          - Fleet Size Recommendation Change
      State: ENABLED
      Targets:
        - Arn: !GetAtt
            - AutoScalingLambda
            - Arn
          DeadLetterConfig:
            Arn: !GetAtt
              - UnprocessedAutoScalingEventQueue
              - Arn
          Id: Target0
          RetryPolicy:
            MaximumRetryAttempts: 15
  AutoScalingEventRuleTargetPermission:
    Type: 'AWS::Lambda::Permission'
    Properties:
      Action: 'lambda:InvokeFunction'
      FunctionName: !GetAtt
        - AutoScalingLambda
        - Arn
      Principal: events.amazonaws.com
      SourceArn: !GetAtt
        - AutoScalingEventRule
        - Arn
  AutoScalingLambdaServiceRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: 'sts:AssumeRole'
            Effect: Allow
            Principal:
```

```
            Service: lambda.amazonaws.com
        Version: 2012-10-17
      ManagedPolicyArns:
        - !Join
          - ''
          - - 'arn:'
            - !Ref 'AWS::Partition'
            - ':iam::aws:policy/service-role/AWSLambdaBasicExecutionRole'
  AutoScalingLambdaServiceRoleDefaultPolicy:
    Type: 'AWS::IAM::Policy'
    Properties:
      PolicyDocument:
        Statement:
          - Action: 'autoscaling:SetDesiredCapacity'
            Effect: Allow
            Resource: '*'
        Version: 2012-10-17
      PolicyName: AutoScalingLambdaServiceRoleDefaultPolicy
      Roles:
        - !Ref AutoScalingLambdaServiceRole
  UnprocessedAutoScalingEventQueue:
    Type: 'AWS::SQS::Queue'
    Properties:
      QueueName: deadline-unprocessed-autoscaling-events
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  UnprocessedAutoScalingEventQueuePolicy:
    Type: 'AWS::SQS::QueuePolicy'
    Properties:
      PolicyDocument:
        Statement:
          - Action: 'sqs:SendMessage'
            Condition:
              ArnEquals:
                'aws:SourceArn': !GetAtt
                  - AutoScalingEventRule
                  - Arn
            Effect: Allow
            Principal:
              Service: events.amazonaws.com
            Resource: !GetAtt
              - UnprocessedAutoScalingEventQueue
              - Arn
        Version: 2012-10-17
```

```
      Queues:
        - !Ref UnprocessedAutoScalingEventQueue
```

执行舰队运行状况检查

创建队列后，您应构建自定义运行状况检查，以确保您的队列保持健康且没有停滞的实例，从而帮助避免不必要的成本。请参阅部署 Deadline Cloud 舰队运行状况检查 GitHub。运行状况检查可以降低在未被发现的情况下意外更改您的Amazon Machine Image启动模板或网络配置的风险。

配置和使用 Deadline Cloud 服务管理的舰队

服务管理车队 (SMF) 是由 Deadline Cloud 管理的员工集合。SMF 无需为处理需求而管理队列扩展，也无需在任务完成后缩小队伍规模。

当使用默认 conda 队列环境将 SMF 与队列关联时，Deadline Cloud 会使用相应的软件包为队列中的工作人员配置。有关支持的合作伙伴应用程序，请参阅 De AWS adline Cloud 用户指南中的默认 conda 队列环境。

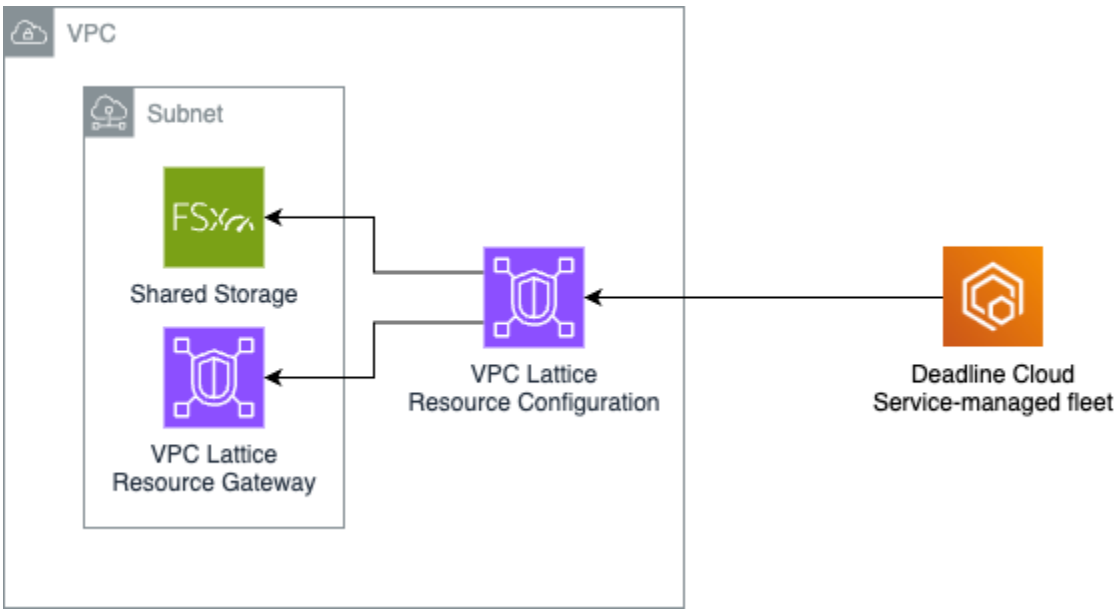
在大多数情况下，您无需更改 SMF 即可处理您的工作负载。但是，在某些情况下，可能需要您对车队进行更改。

> **Note**
>
> 要使用主机配置脚本在 Worker 上安装自定义软件，请参见使用管理员权限运行主机配置脚本。

主题

使用 VPC 资源终端节点将 VPC 资源连接到您的 SMF

使用适用于 Deadline Cloud 服务托管舰队 (SMF) 的 Amazon VPC 资源终端节点，您可以将 VPC 资源（例如网络文件系统 (NFS)、许可证服务器和数据库）与 Deadline Cloud 工作人员连接起来。此功能允许您利用 Deadline Cloud 的完全托管平台，同时与 VPC 内的现有基础设施集成。



> **Tip**
>
> 有关设置亚马逊 FSx 集群并将其连接到服务托管队列的参考 CloudFormation 模板，请参阅上的 Deadline Cloud 示例存储库中的 s mf_vpc_fsx。 GitHub

VPC 资源终端节点的工作原理

VPC 资源终端节点使用 VPC Lattice 在您的 Deadline Cloud SMF 工作线程和 VPC 中的资源之间创建安全连接。连接是单向的，这意味着工作人员可以与您的 VPC 中的资源建立连接并来回传输数据，但是您的 VPC 中的资源无法与工作人员建立连接。

当您将一个 VPC 资源连接到 Deadline Cloud 服务管理的队列时，您的工作人员可以使用私有域名访问您的 VPC 中的资源。此外，流量通过VPC Lattice从工作人员流向您的VPC资源，而您的VPC中的资源则看到来自VPC莱迪思资源网关的流量。

要了解更多信息，请参阅 VPC Lattice用户指南。

先决条件

在将 VPC 资源连接到 Deadline Cloud 服务管理的队列之前，请确保您具备以下条件：

- 一个拥有包含您要连接的资源的 VPC 的 AWS 账户。
- 创建和管理 VPC 莱迪思资源的 IAM 权限。
- 具有至少一个服务托管队列的 Deadline 云场。

- 您想要访问的 VPC 资源（FSxNFS、许可证服务器等）。

设置 VPC 资源终端节点

要设置 VPC 资源终端节点，您需要在 VP C Lattice 中创建资源 AWS RAM，然后在 Deadline Cloud 中将这些资源连接到您的队列。要为您的 SMF 设置 VPC 资源终端节点，请完成以下步骤。

1. 要在 VPC Lattice 中创建资源网关，请参阅 VPC 莱迪思用户指南中的创建资源网关。
2. 要在 VPC Lattice 中创建资源配置，请参阅 VPC 莱迪思用户指南中的创建资源配置。
3. 要与您的 Deadline Cloud 队列共享资源，请在中创建资源共享 AWS RAM。有关说明，请参阅创建资源共享。

   在创建资源共享时，对于委托人，请从下拉列表中选择服务主体，然后输入**`fleets.deadline.amazonaws.com`**。
4. 要将资源配置与您的 Deadline Cloud 队列连接起来，请完成以下步骤。

   a. 如果您还没有，请打开 De adline Cloud 控制台。
   b. 在导航窗格中，选择农场，然后选择您的农场。
   c. 选择 “舰队” 选项卡，然后选择您的舰队。
   d. 选择配置选项卡。
   e. 在 VPC 资源终端节点下，选择编辑。
   f. 选择您创建的资源配置，然后选择保存更改。

访问您的 VPC 资源

将您的 VPC 资源连接到队列后，工作人员可以使用以下格式的私有域名对其进行访问：`<resource_config_id>.resource-endpoints.deadline.<region>.amazonaws.com`

该域名是私有的，只有工作人员才能访问（不能通过互联网或您的工作站）。

要在您的工作人员上安装或配置对 VPC 资源的访问权限，请使用主机配置脚本。当工作人员启动时，主机配置脚本以管理员权限运行，允许您装载文件系统、配置网络设置或执行其他设置任务。

身份验证和安全

对于需要身份验证的资源，请将凭据安全地存储在 S AWS ecrets Manager 中，访问主机配置脚本或作业脚本中的密钥，并实施适当的文件系统权限来控制访问权限。在多个舰队之间共享资源时，请考虑安

全影响。例如，如果两个队列连接到同一个共享存储，则在一个队列上运行的作业可能能够访问从另一个队列创建的资产。

技术注意事项

使用 VPC 资源终端节点时，请考虑以下几点：

- 只能启动从工作人员到 VPC 资源的连接，不能从 VPC 资源启动到工作线程的连接。
- 建立连接后，即使资源配置已断开连接，连接也会一直持续到重置为止。
- VPC Lattice 连接可自动处理可用区之间的连接，无需支付额外费用。您的资源网关必须与您的 VPC 资源共享一个可用区，因此我们建议将资源网关配置为跨越所有可用区。
- 通过 VPC 资源终端节点的流量使用网络地址转换 (NAT)，这并不与所有用例兼容。例如，微软 Active Directory (AD) 无法通过 NAT 进行连接。

有关 VPC 莱迪思配额的更多信息，请参阅 VPC 莱迪思配额。

问题排查

如果您遇到与 VPC 资源终端节点有关的问题，请检查以下内容。

- 如果您收到诸如 “mount.nfs：装载时服务器拒绝访问” 之类的错误消息，则可能需要更新 NFS 卷的客户端配置。
- 通过在 Amazon EC2 实例或 VPC AWS CloudShell 中进行测试来验证您的资源配置设置。
- 使用简单的 CLI 作业测试你的 Deadline Cloud 连接。有关更多信息，请参阅上的 Deadline Cloud 示例 GitHub。
- 如果您遇到连接故障，请检查资源网关安全组的设置。
- 启用 VPC 访问日志以监控连接。

在服务托管队列中使用作业附件

作业附件使用亚马逊简单存储服务 (Amazon S3) Service 在您的工作站和 Deadline Cloud 工作人员之间传输文件。您可以单独使用作业附件，也可以与共享存储一起使用，将辅助数据附加到未与其他作业共享的作业，例如作业脚本、配置文件或存储在本地的项目资产。

有关作业附件的工作原理的信息，请参阅 De adline Cloud 用户指南中的作业附件。有关在作业捆绑包中指定输入和输出文件的详细信息，请参阅使用作业附件共享文件。

选择文件系统模式

提交带附件的作业时，您可以通过设置`fileSystem`属性来选择工作人员如何从 Amazon S3 加载文件：

- 已复制（默认）-在任务开始之前将所有文件下载到本地磁盘。最好在每项任务都需要大多数输入文件时使用。
- 虚拟-装载按需下载文件的虚拟文件系统。当任务只需要输入文件的子集时，效果最好。仅在 Linux SMF 工作程序上可用。

> **Important**
>
> 虚拟模式缓存会增加内存消耗，并且并未针对所有工作负载进行优化。我们建议您在运行生产作业之前测试工作负载。

有关配置文件系统模式的详细信息，请参阅 De adline Cloud 用户指南中的虚拟文件系统。

优化传输性能

将文件从 Amazon S3 同步到 SMF 工作程序的速度取决于您的队列的亚马逊弹性区块存储 (Amazon EBS) Block Store 卷配置。默认情况下，SMF 工作程序使用具有基准性能设置的 gp3 Amazon EBS 卷。对于具有大型输入文件或许多小文件的工作负载，您可以通过提高 Amazon EBS 吞吐量和 IOPS 设置来提高传输速度。您可以使用 AWS Command Line Interface (AWS CLI) 更新这些设置。

吞吐量（MiB/s）

从卷读取或写入数据的速率。gp3 卷MiB/s, maximum is 1,000 MiB/s的默认值为 125。对于大型顺序文件传输，请增加。

IOPS

每秒输入/输出操作数。默认值为 3,000 IOPS，gp3 卷的最大值为 16,000 IOPS。传输许多小文件时增加。

> **Note**
>
> 增加 Amazon EBS 吞吐量和 IOPS 会增加机队成本。有关价格信息，请参阅 De adline Cloud 定价。

要在现有队列上更新 Amazon EBS 设置，请使用 AWS CLI

- 运行如下命令：

```
aws deadline update-fleet \
  --farm-id farm-0123456789abcdef0 \
  --fleet-id fleet-0123456789abcdef0 \
  --configuration '{
    "serviceManagedEc2": {
      "instanceCapabilities": {
        "vCpuCount": {"min": 4},
        "memoryMiB": {"min": 8192},
        "osFamily": "linux",
        "cpuArchitectureType": "x86_64",
        "rootEbsVolume": {
          "sizeGiB": 250,
          "iops": 6000,
          "throughputMiB": 500
        }
      },
      "instanceMarketOptions": {"type": "spot"}
    }
  }'
```

下载作业输出

任务完成后，使用 Deadline Cloud CLI 或 AWS Deadline Cloud 监控器（Deadline Cloud 监控器）下载输出文件：

```
deadline job download-output --job-id job-0123456789abcdef0
```

要在作业完成后自动下载输出，请参阅 De adline Cloud 用户指南中的自动下载。

在工作人员上部署和配置自定义软件

AWS Deadline Cloud 提供了多种在工作人员上部署和配置自定义软件、插件和工具的方法。您选择的方法取决于您的要求，例如您是否需要管理员权限、软件更改频率，以及该软件是适用于所有作业还是仅适用于特定作业。

选择部署方法

使用下表为您的用例选择正确的部署方法。

标准	队列环境	主机配置脚本	定制 conda 套餐
需要管理员权限	否	是	否
它何时运行	会话开始	工人创业	会话开始
Scope	每个队列或作业	车队中的所有员工	每个队列或作业
可以通过提交作业来控制	是	否	是
设置复杂性	低	中	高
适用于	简单的插件、脚本、环境变量	系统驱动程序、Docker、存储支架	具有依赖关系的复杂应用程序

快速决策指南：

- 需要管理员权限或 root 权限吗？ 使用主机配置脚本。
- 没有管理员权限的简单插件或脚本？ 使用队列环境。
- 需要版本控制的复杂应用程序？ 创建自定义 conda 软件包。

使用队列环境配置作业

AWS Deadline Cloud 使用队列环境在工作人员上配置软件。环境使您能够对会话中的所有任务执行一次耗时的任务，例如设置和拆卸。它定义了启动或停止会话时要在工作器上运行的操作。您可以为队列、队列中运行的作业以及作业的各个步骤配置环境。

您可以将环境定义为队列环境或作业环境。使用 Deadline Cloud 控制台或截止日期：CreateQueueEnvironment操作创建队列环境，并在您提交的作业的作业模板中定义作业环境。它们遵循环境的 Open Job Description (OpenJD) 规范。有关详细信息，请参阅<Environment>上https://github.com/OpenJobDescription/openjd-specifications/wiki/2023-09-Template-Schemas#4-environment的 OpenJD 规范。 GitHub

除了name和之外description，每个环境还包含两个用于定义主机环境的字段。它们是：

- script— 在工作器上运行此环境时采取的操作。
- variables— 进入环境时设置的一组环境变量 name/value 对。

必须至少设置script或中的一个variables。

您可以在作业模板中定义多个环境。每个环境都是按照它们在模板中列出的顺序应用的。您可以使用它来帮助管理环境的复杂性。

Deadline Cloud 的默认队列环境使用 conda 包管理器将软件加载到环境中，但你可以使用其他包管理器。默认环境定义了两个参数来指定应加载的软件。这些变量由 Deadline Cloud 提供的提交者设置，但您可以在自己的脚本和使用默认环境的应用程序中进行设置。它们是：

- CondaPackages— 以空格分隔的 conda 软件包列表与要为该任务安装的规格相匹配。例如，Blender 提交者将在 Blender 3.6 中添加blender=3.6渲染帧。
- CondaChannels— 以空格分隔的 conda 频道列表，用于安装软件包。对于服务管理的舰队，软件包是通过渠道安装的。deadline-cloud您可以添加其他频道。

使用 OpenJD 队列环境控制作业环境

您可以使用队列环境为渲染作业定义自定义环境。队列环境是一种模板，用于控制在特定队列中运行的作业的环境变量、文件映射和其他设置。它使您能够根据工作负载的要求为提交到队列的作业定制执行环境。 AWS Deadline Cloud 提供了三个嵌套级别，您可以在其中应用开放式职位描述 (OpenJD) 环境：队列、作业和步骤。通过定义队列环境，您可以确保不同类型的作业具有一致且经过优化的性能，简化资源分配并简化队列管理。

队列环境是一个模板，您可以通过 AWS 管理控制台或使用将其附加到 AWS 账户中的队列 AWS CLI。您可以为队列创建一个环境，也可以创建用于创建执行环境的多个队列环境。这种方法使您能够分步创建和测试环境，以帮助确保它能正常运行于您的作业。

作业和步骤环境是在您用来在队列中创建作业的作业模板中定义的。在这些不同形式的环境中，OpenJD 语法是相同的。在本节中，我们将在作业模板中展示它们。

主题

在队列环境中设置环境变量

许多应用程序和框架使用环境变量来控制功能设置、日志级别和显示配置。你可以使用 Ope n Job Description (OpenJD) 环境来设置其范围内的每个任务命令都继承的环境变量。

环境变量作用域

AWS Deadline Cloud 会应用您附加到队列的队列环境中的环境变量。在作业模板中，您还可以使用 OpenJD 环境在作业和步骤级别定义环境变量。在较窄的作用域中定义的变量会覆盖范围更广的同名变量。

- 队列环境 — 附加到 Deadline Cloud 中队列的模板。变量适用于提交到队列的所有作业。您可以使用固定值的`variables`映射来设置变量，也可以使用脚本来设置动态值。
- 作业环境-`jobEnvironments` 在作业模板中定义。变量适用于作业中的所有步骤和任务。作业级变量会覆盖同名的队列级变量。
- 步骤环境-`stepEnvironments` 在作业模板中定义。变量仅适用于该步骤中的任务。阶梯级变量会覆盖同名的作业级别或队列级变量。

在队列环境中设置变量

可以在队列环境中使用固定值的`variables`映射来设置环境变量，或者使用`script`带动`onEnter`作表示动态值的环境变量。

以下队列环境模板使用`variables`映射将`QT_QPA_PLATFORM`变量设置为`offscreen`，这允许使用 Qt Fram ework 的应用程序在没有交互式显示屏的情况下在工作主机上运行。

```
specificationVersion: 'environment-2023-09'
environment:
  name: QtOffscreen
  variables:
```

```
    QT_QPA_PLATFORM: offscreen
```

对于动态值，例如修改PATH或激活虚拟环境，请使用将格式openjd_env: *VAR*=*value*为 stdout 的行打印脚本。前openjd_env:缀为必填项。使用不带前缀的echoexport、或其他外壳机制不会将变量传播到作业和任务。

以下队列环境模板使用脚本设置QT_QPA_PLATFORM变量。

```
specificationVersion: 'environment-2023-09'
environment:
  name: QtOffscreen
  script:
    actions:
      onEnter:
        command: bash
        args:
        - "{{Env.File.Enter}}"
    embeddedFiles:
    - name: Enter
      type: TEXT
      data: |
        #!/bin/env bash
        set -euo pipefail
        echo "openjd_env: QT_QPA_PLATFORM=offscreen"
```

要将队列环境附加到您的队列，请使用 Deadline Cloud 控制台或 AWS CLI。有关更多信息，请参阅 De AWS adline Cloud 用户指南中的创建队列环境。以下 AWS CLI 命令根据模板文件创建队列环境。

```
aws deadline create-queue-environment \
    --farm-id FARM_ID \
    --queue-id QUEUE_ID \
    --priority 1 \
    --template-type YAML \
    --template file://my-queue-env.yaml
```

有关更复杂的示例，例如创建和激活 conda 虚拟环境，请参阅上 GitHub的 De adline Cloud 队列环境示例。

在作业模板中设置变量

在作业模板中，向jobEnvironments或stepEnvironments条目添加variables地图。每个条目都是一个键值对，其中键是变量名，值是变量值。

以下作业模板将QT_QPA_PLATFORM环境变量设置为offscreen，这允许使用 Qt Fram ework 的应用程序在没有交互式显示屏的情况下在工作主机上运行。

```
specificationVersion: 'jobtemplate-2023-09'
name: MyJob
jobEnvironments:
- name: JobEnv
  variables:
    QT_QPA_PLATFORM: offscreen
```

可以在单个环境定义中设置多个变量。

```
jobEnvironments:
- name: JobEnv
  variables:
    JOB_VERBOSITY: MEDIUM
    JOB_PROJECT_ID: my-project-id
    JOB_ENDPOINT_URL: https://my-host-name/my/path
    QT_QPA_PLATFORM: offscreen
```

您可以使用{{Param.*ParameterName*}}语法在变量值中引用作业参数。

```
jobEnvironments:
- name: JobEnv
  variables:
    JOB_EXAMPLE_PARAM: "{{Param.ExampleParam}}"
```

要覆盖特定步骤的作业级变量，请使用相同的变量名称定义一个stepEnvironments条目。以下示例JOB_PROJECT_ID在作业级别使用值进行定义project-12，然后使用覆盖步骤级别的step-project-12值。步骤中的任务使用步骤级别的值。

```
specificationVersion: 'jobtemplate-2023-09'
name: MyJob
jobEnvironments:
- name: JobEnv
  variables:
    JOB_PROJECT_ID: project-12
steps:
- name: MyStep
  stepEnvironments:
  - name: StepEnv
    variables:
```

```
      JOB_PROJECT_ID: step-project-12
```

试试看：运行环境变量示例

Deadline Cloud 示例存储库包含一个作业包，用于演示如何设置和查看环境变量。示例作业模板定义了作业和步骤级别的变量，然后运行打印合并结果的任务。使用以下步骤运行样本并检查结果。

先决条件

1. 如果您没有包含队列和关联的 Linux 队列的 Deadline Cloud 场，请按照 De adline Cloud 控制台中的指导性入门体验创建具有默认设置的群组。
2. 如果您的工作站上没有 Deadline Cloud CLI 和 De AWS adline Cloud 监控器，请按照设置 Deadline Cloud 提交者中的步骤进行操作。
3. 用于克隆 D `git` e adline Cloud 示例 GitHub存储库。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
cd deadline-cloud-samples/job_bundles
```

运行示例

1. 使用 Deadline Cloud CLI 提交`job_env_vars`示例。

```
deadline bundle submit job_env_vars
```

2. 在 Deadline Cloud 监视器中，选择新作业以监控其进度。与Linux队列关联的队列有工作人员可用后，任务将在几秒钟内完成。选择任务，然后在任务面板的右上角菜单中选择 “查看日志”。

将会话操作与其定义进行比较

日志视图显示三个会话操作。在文本编辑器中打开 job_env_vars/template.yaml 文件，将每个操作与其在作业模板中的定义进行比较。

1. 选择 “启动 JobEnv会话” 操作。日志输出显示正在设置的作业级环境变量。

```
Setting: JOB_VERBOSITY=MEDIUM
Setting: JOB_EXAMPLE_PARAM=An example parameter value
Setting: JOB_PROJECT_ID=project-12
Setting: JOB_ENDPOINT_URL=https://internal-host-name/some/path
Setting: QT_QPA_PLATFORM=offscreen
```

作业模板中的以下几行定义了此环境。

```
jobEnvironments:
- name: JobEnv
  variables:
    JOB_VERBOSITY: MEDIUM
    JOB_EXAMPLE_PARAM: "{{Param.ExampleParam}}"
    JOB_PROJECT_ID: project-12
    JOB_ENDPOINT_URL: https://internal-host-name/some/path
    QT_QPA_PLATFORM: offscreen
```

2. 选择 “启动 StepEnv会话” 操作。日志输出显示步骤级变量，包括被覆盖JOB_PROJECT_ID的变量。

```
Setting: STEP_VERBOSITY=HIGH
Setting: JOB_PROJECT_ID=step-project-12
```

作业模板中的以下几行定义了此环境。

```
stepEnvironments:
- name: StepEnv
  variables:
    STEP_VERBOSITY: HIGH
    JOB_PROJECT_ID: step-project-12
```

3. 选择任务运行会话操作。日志输出显示任务可用的合并环境变量。请注意，JOB_PROJECT_ID它使用的是阶梯级值step-project-12。

```
Environment variables starting with JOB_*:
JOB_ENDPOINT_URL=https://internal-host-name/some/path
JOB_EXAMPLE_PARAM='An example parameter value'
JOB_PROJECT_ID=step-project-12
JOB_VERBOSITY=MEDIUM

Environment variables starting with STEP_*:
STEP_VERBOSITY=HIGH
```

在队列环境中设置路径

使用 OpenJD 环境在环境中提供新命令。首先，创建一个包含脚本文件的目录，然后将该目录添加到PATH环境变量中，这样脚本中的可执行文件就可以运行它们，而无需每次都指定目录路径。环境定

义中的变量列表不提供修改变量的方法，因此您可以通过运行脚本来完成此操作。在脚本设置并修改之后PATH，它会使用命令将变量导出到 OpenJD 运行时。`echo "openjd_env: PATH=$PATH"`

先决条件

执行以下步骤，使用来自 Deadline Cloud 示例 github 存储库的环境变量运行示例作业包。

1. 如果您没有包含队列和关联的 Linux 队列的 Deadline Cloud 场，请按照 De adline Cloud 控制台中的指导性入门体验创建具有默认设置的群组。
2. 如果您的工作站上没有 Deadline Cloud CLI 和 Deadline Cloud 监控器，请按照用户指南中设置 Deadline Cloud 提交者中的步骤进行操作。
3. 用于克隆 D `git` e adline Cloud 示例 GitHub存储库。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
 Cloning into 'deadline-cloud-samples'...
 ...
cd deadline-cloud-samples/job_bundles
```

运行路径示例

1. 使用 Deadline Cloud CLI 提交`job_env_with_new_command`示例。

```
 $ deadline bundle submit job_env_with_new_command
 Submitting to Queue: MySampleQueue
 ...
```

2. 在 Deadline Cloud 监视器中，您将看到新作业并可以监控其进度。一旦与Linux队列关联的队列有工作人员可以运行作业的任务，该作业将在几秒钟内完成。选择任务，然后在任务面板的右上角菜单中选择 “查看日志” 选项。

   右边是两个会话操作：“启动” RandomSleepCommand 和 “任务运行”。窗口中央的日志查看器对应于右侧选定的会话操作。

将会话操作与其定义进行比较

在本节中，您将使用 Deadline Cloud 监视器将会话操作与作业模板中定义的会话操作进行比较。它延续了上一节。

在文本编辑器中打开 job_env_with_new_command/temp late.yaml 文件。将会话操作与作业模板中的定义位置进行比较。

1. 在 Deadline Cloud 监视器中选择 “启动 RandomSleepCommand会话” 操作。您将看到如下所示的日志输出。

```
 2024/07/16 17:25:32-07:00
 2024/07/16 17:25:32-07:00 ================================================
 2024/07/16 17:25:32-07:00 --------- Entering Environment: RandomSleepCommand
 2024/07/16 17:25:32-07:00 ================================================
 2024/07/16 17:25:32-07:00 ------------------------------------------------
 2024/07/16 17:25:32-07:00 Phase: Setup
 2024/07/16 17:25:32-07:00 ------------------------------------------------
 2024/07/16 17:25:32-07:00 Writing embedded files for Environment to disk.
 2024/07/16 17:25:32-07:00 Mapping: Env.File.Enter -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmpbt8j_c3f
 2024/07/16 17:25:32-07:00 Mapping: Env.File.SleepScript -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmperastlp4
 2024/07/16 17:25:32-07:00 Wrote: Enter -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmpbt8j_c3f
 2024/07/16 17:25:32-07:00 Wrote: SleepScript -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmperastlp4
 2024/07/16 17:25:32-07:00 ------------------------------------------------
 2024/07/16 17:25:32-07:00 Phase: Running action
 2024/07/16 17:25:32-07:00 ------------------------------------------------
 2024/07/16 17:25:32-07:00 Running command sudo -u job-user -i setsid -w /sessions/
session-ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/tmpbwrquq5u.sh
 2024/07/16 17:25:32-07:00 Command started as pid: 2205
 2024/07/16 17:25:32-07:00 Output:
 2024/07/16 17:25:33-07:00 openjd_env: PATH=/sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/bin:/opt/conda/condabin:/home/job-
user/.local/bin:/home/job-user/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/
bin:/sbin:/bin:/var/lib/snapd/snap/bin
 No newer logs at this moment.
```

作业模板中的以下几行指定了此操作。

```
 jobEnvironments:
 - name: RandomSleepCommand
   description: Adds a command 'random-sleep' to the environment.
   script:
     actions:
       onEnter:
         command: bash
         args:
```

```
        - "{{Env.File.Enter}}"
    embeddedFiles:
    - name: Enter
      type: TEXT
      data: |
        #!/bin/env bash
        set -euo pipefail

        # Make a bin directory inside the session's working directory for providing
 new commands
        mkdir -p '{{Session.WorkingDirectory}}/bin'

        # If this bin directory is not already in the PATH, then add it
        if ! [[ ":$PATH:" == *':{{Session.WorkingDirectory}}/bin:'* ]]; then
          export "PATH={{Session.WorkingDirectory}}/bin:$PATH"

          # This message to Open Job Description exports the new PATH value to the
 environment
          echo "openjd_env: PATH=$PATH"
        fi

        # Copy the SleepScript embedded file into the bin directory
        cp '{{Env.File.SleepScript}}' '{{Session.WorkingDirectory}}/bin/random-
sleep'
        chmod u+x '{{Session.WorkingDirectory}}/bin/random-sleep'
    - name: SleepScript
      type: TEXT
      runnable: true
      data: |
        ...
```

2. 在 Deadline Cloud 监视器中选择 “启动 StepEnv会话” 操作。您将看到如下所示的日志输出。

```
 2024/07/16 17:25:33-07:00
 2024/07/16 17:25:33-07:00 ================================================
 2024/07/16 17:25:33-07:00 --------- Running Task
 2024/07/16 17:25:33-07:00 ================================================
 2024/07/16 17:25:33-07:00 ------------------------------------------------
 2024/07/16 17:25:33-07:00 Phase: Setup
 2024/07/16 17:25:33-07:00 ------------------------------------------------
 2024/07/16 17:25:33-07:00 Writing embedded files for Task to disk.
 2024/07/16 17:25:33-07:00 Mapping: Task.File.Run -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmpdrwuehjf
```

```
 2024/07/16 17:25:33-07:00 Wrote: Run -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmpdrwuehjf
 2024/07/16 17:25:33-07:00 ------------------------------------------------
 2024/07/16 17:25:33-07:00 Phase: Running action
 2024/07/16 17:25:33-07:00 ------------------------------------------------
 2024/07/16 17:25:33-07:00 Running command sudo -u job-user -i setsid -w /sessions/
session-ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/tmpz81iaqfw.sh
 2024/07/16 17:25:33-07:00 Command started as pid: 2256
 2024/07/16 17:25:33-07:00 Output:
 2024/07/16 17:25:34-07:00 + random-sleep 12.5 27.5
 2024/07/16 17:26:00-07:00 Sleeping for duration 26.90
 2024/07/16 17:26:00-07:00 ------------------------------------------------
 2024/07/16 17:26:00-07:00 Uploading output files to Job Attachments
 2024/07/16 17:26:00-07:00 ------------------------------------------------
```

3. 作业模板中的以下几行指定了此操作。

```
steps:
- name: EnvWithCommand
  script:
    actions:
      onRun:
        command: bash
        args:
        - '{{Task.File.Run}}'
    embeddedFiles:
    - name: Run
      type: TEXT
      data: |
        set -xeuo pipefail

        # Run the script installed into PATH by the job environment
        random-sleep 12.5 27.5
  hostRequirements:
    attributes:
    - name: attr.worker.os.family
      anyOf:
      - linux
```

在队列环境中运行后台守护进程

在许多渲染用例中，加载应用程序和场景数据可能需要大量时间。如果作业每帧都重新加载它们，则会将大部分时间花在开销上。通常可以将应用程序作为后台守护进程加载一次，让它加载场景数据，然后通过进程间通信 (IPC) 向其发送命令以执行渲染。

许多开源 Deadline Cloud 集成都使用这种模式。Open Job Description 项目提供了一个适配器运行时库，该库在所有支持的操作系统上都具有强大的 IPC 模式。

为了演示这种模式，有一个独立的示例任务包，它使用 Python 和 bash 代码来实现后台守护程序，并使用 IPC 来实现任务与之通信。该守护程序是用 Python 实现的，它监听 POSIX SIGUSR1 信号以了解何时处理任务。任务详细信息以特定的 JSON 文件形式传递给守护程序，运行任务的结果将作为另一个 JSON 文件返回。

先决条件

执行以下步骤，使用来自 Deadline Cloud 示例 github 存储库的守护程序进程运行示例作业包。

1. 如果您没有包含队列和关联的 Linux 队列的 Deadline Cloud 场，请按照 De adline Cloud 控制台中的指导性入门体验创建具有默认设置的群组。
2. 如果您的工作站上没有 Deadline Cloud CLI 和 Deadline Cloud 监控器，请按照用户指南中设置 Deadline Cloud 提交者中的步骤进行操作。
3. 用于克隆 D `git` e adline Cloud 示例 GitHub存储库。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
 Cloning into 'deadline-cloud-samples'...
 ...
cd deadline-cloud-samples/job_bundles
```

运行守护程序示例

1. 使用 Deadline Cloud CLI 提交`job_env_daemon_process`示例。

```
 git clone https://github.com/aws-deadline/deadline-cloud-samples.git
Cloning into 'deadline-cloud-samples'...
 ...
cd deadline-cloud-samples/job_bundles
```

2. 在 Deadline Cloud 监控应用程序中，您将看到新作业并可以监控其进度。一旦与Linux队列关联的队列有工作人员可以运行作业的任务，该任务将在大约一分钟后完成。选择其中一项任务后，在任务面板的右上角菜单中选择 “查看日志” 选项。

   右侧有两个会话操作：“启动” DaemonProcess 和 “任务运行”。窗口中央的日志查看器对应于右侧选定的会话操作。

   选择 “查看所有任务的日志” 选项。时间轴显示了作为会话一部分运行的其余任务以及退出环境的`Shut down DaemonProcess`操作。

查看守护程序日志

1. 在本节中，您将使用 Deadline Cloud 监视器将会话操作与作业模板中定义的会话操作进行比较。它延续了上一节。

   在文本编辑器中打开 job_env_daemon_process/temp late.yaml 文件。将会话操作与作业模板中的定义位置进行比较。

2. 在 Deadline Cloud 监视器中选择`Launch DaemonProcess`会话操作。您将看到如下所示的日志输出。

```
 2024/07/17 16:27:20-07:00
 2024/07/17 16:27:20-07:00 ================================================
 2024/07/17 16:27:20-07:00 --------- Entering Environment: DaemonProcess
 2024/07/17 16:27:20-07:00 ================================================
 2024/07/17 16:27:20-07:00 ------------------------------------------------
 2024/07/17 16:27:20-07:00 Phase: Setup
 2024/07/17 16:27:20-07:00 ------------------------------------------------
 2024/07/17 16:27:20-07:00 Writing embedded files for Environment to disk.
 2024/07/17 16:27:20-07:00 Mapping: Env.File.Enter -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/enter-daemon-
process-env.sh
 2024/07/17 16:27:20-07:00 Mapping: Env.File.Exit -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/exit-daemon-
process-env.sh
 2024/07/17 16:27:20-07:00 Mapping: Env.File.DaemonScript -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
script.py
 2024/07/17 16:27:20-07:00 Mapping: Env.File.DaemonHelperFunctions -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
helper-functions.sh
```

```
 2024/07/17 16:27:20-07:00 Wrote: Enter -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/enter-daemon-
process-env.sh
 2024/07/17 16:27:20-07:00 Wrote: Exit -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/exit-daemon-
process-env.sh
 2024/07/17 16:27:20-07:00 Wrote: DaemonScript -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
script.py
 2024/07/17 16:27:20-07:00 Wrote: DaemonHelperFunctions -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
helper-functions.sh
 2024/07/17 16:27:20-07:00 ----------------------------------------------
 2024/07/17 16:27:20-07:00 Phase: Running action
 2024/07/17 16:27:20-07:00 ----------------------------------------------
 2024/07/17 16:27:20-07:00 Running command sudo -u job-user -i setsid -w /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/tmp_u8slys3.sh
 2024/07/17 16:27:20-07:00 Command started as pid: 2187
 2024/07/17 16:27:20-07:00 Output:
 2024/07/17 16:27:21-07:00 openjd_env: DAEMON_LOG=/sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/daemon.log
 2024/07/17 16:27:21-07:00 openjd_env: DAEMON_PID=2223
 2024/07/17 16:27:21-07:00 openjd_env: DAEMON_BASH_HELPER_SCRIPT=/sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
helper-functions.sh
```

作业模板中的以下几行指定了此操作。

```
  stepEnvironments:
  - name: DaemonProcess
    description: Runs a daemon process for the step's tasks to share.
    script:
      actions:
        onEnter:
          command: bash
          args:
          - "{{Env.File.Enter}}"
        onExit:
          command: bash
          args:
          - "{{Env.File.Exit}}"
      embeddedFiles:
      - name: Enter
```

```
          filename: enter-daemon-process-env.sh
          type: TEXT
          data: |
            #!/bin/env bash
            set -euo pipefail

            DAEMON_LOG='{{Session.WorkingDirectory}}/daemon.log'
            echo "openjd_env: DAEMON_LOG=$DAEMON_LOG"
            nohup python {{Env.File.DaemonScript}} > $DAEMON_LOG 2>&1 &
            echo "openjd_env: DAEMON_PID=$!"
            echo "openjd_env:
 DAEMON_BASH_HELPER_SCRIPT={{Env.File.DaemonHelperFunctions}}"

            echo 0 > 'daemon_log_cursor.txt'
      ...
```

3. 在 Deadline Cloud 监视器中选择 “任务运行：N 会话” 操作之一。您将看到如下所示的日志输出。

```
2024/07/17 16:27:22-07:00
 2024/07/17 16:27:22-07:00 ================================================
 2024/07/17 16:27:22-07:00 --------- Running Task
 2024/07/17 16:27:22-07:00 ================================================
 2024/07/17 16:27:22-07:00 Parameter values:
 2024/07/17 16:27:22-07:00 Frame(INT) = 2
 2024/07/17 16:27:22-07:00 ------------------------------------------------
 2024/07/17 16:27:22-07:00 Phase: Setup
 2024/07/17 16:27:22-07:00 ------------------------------------------------
 2024/07/17 16:27:22-07:00 Writing embedded files for Task to disk.
 2024/07/17 16:27:22-07:00 Mapping: Task.File.Run -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/run-task.sh
 2024/07/17 16:27:22-07:00 Wrote: Run -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/run-task.sh
 2024/07/17 16:27:22-07:00 ------------------------------------------------
 2024/07/17 16:27:22-07:00 Phase: Running action
 2024/07/17 16:27:22-07:00 ------------------------------------------------
 2024/07/17 16:27:22-07:00 Running command sudo -u job-user -i setsid -w /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/tmpv4obfkhn.sh
 2024/07/17 16:27:22-07:00 Command started as pid: 2301
 2024/07/17 16:27:22-07:00 Output:
 2024/07/17 16:27:23-07:00 Daemon PID is 2223
 2024/07/17 16:27:23-07:00 Daemon log file is /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/daemon.log
 2024/07/17 16:27:23-07:00
 2024/07/17 16:27:23-07:00 === Previous output from daemon
```

```
2024/07/17 16:27:23-07:00 ===
2024/07/17 16:27:23-07:00
2024/07/17 16:27:23-07:00 Sending command to daemon
2024/07/17 16:27:23-07:00 Received task result:
2024/07/17 16:27:23-07:00 {
2024/07/17 16:27:23-07:00   "result": "SUCCESS",
2024/07/17 16:27:23-07:00   "processedTaskCount": 1,
2024/07/17 16:27:23-07:00   "randomValue": 0.2578537967668988,
2024/07/17 16:27:23-07:00   "failureRate": 0.1
2024/07/17 16:27:23-07:00 }
2024/07/17 16:27:23-07:00
2024/07/17 16:27:23-07:00 === Daemon log from running the task
2024/07/17 16:27:23-07:00 Loading the task details file
2024/07/17 16:27:23-07:00 Received task details:
2024/07/17 16:27:23-07:00 {
2024/07/17 16:27:23-07:00  "pid": 2329,
2024/07/17 16:27:23-07:00  "frame": 2
2024/07/17 16:27:23-07:00 }
2024/07/17 16:27:23-07:00 Processing frame number 2
2024/07/17 16:27:23-07:00 Writing result
2024/07/17 16:27:23-07:00 Waiting until a USR1 signal is sent...
2024/07/17 16:27:23-07:00 ===
2024/07/17 16:27:23-07:00
2024/07/17 16:27:23-07:00 ------------------------------------------------
2024/07/17 16:27:23-07:00 Uploading output files to Job Attachments
2024/07/17 16:27:23-07:00 ------------------------------------------------
```

作业模板中的以下几行是指定此操作的内容。```步骤：

```
steps:
- name: EnvWithDaemonProcess
  parameterSpace:
    taskParameterDefinitions:
    - name: Frame
      type: INT
      range: "{{Param.Frames}}"

  stepEnvironments:
    ...

  script:
    actions:
      onRun:
```

```
        timeout: 60
        command: bash
        args:
        - '{{Task.File.Run}}'
    embeddedFiles:
    - name: Run
      filename: run-task.sh
      type: TEXT
      data: |
        # This bash script sends a task to the background daemon process,
        # then waits for it to respond with the output result.

        set -euo pipefail

        source "$DAEMON_BASH_HELPER_SCRIPT"

        echo "Daemon PID is $DAEMON_PID"
        echo "Daemon log file is $DAEMON_LOG"

        print_daemon_log "Previous output from daemon"

        send_task_to_daemon "{\"pid\": $$, \"frame\": {{Task.Param.Frame}} }"
        wait_for_daemon_task_result

        echo Received task result:
        echo "$TASK_RESULT" | jq .

        print_daemon_log "Daemon log from running the task"

  hostRequirements:
    attributes:
    - name: attr.worker.os.family
      anyOf:
      - linux
```

为您的工作提供申请

您可以使用队列环境加载应用程序来处理您的作业。使用 Deadline Cloud 控制台创建服务管理队列时，您可以选择创建使用 conda 包管理器加载应用程序的队列环境。

如果要使用其他包管理器，可以为该管理器创建队列环境。有关使用 Rez 的示例，请参见使用其他软件包管理器。

Deadline Cloud 提供了一个 conda 通道，用于将精选的渲染应用程序加载到您的环境中。他们支持 Deadline Cloud 为数字内容创作应用程序提供的提交者。

你也可以加载软件让 conda-forge 在你的工作中使用。以下示例显示了在运行作业之前使用 Deadline Cloud 提供的队列环境加载应用程序的作业模板。

主题

从 conda 频道获取应用程序

您可以为 Deadline Cloud 工作人员创建自定义队列环境，安装您选择的软件。此示例队列环境的行为与控制台用于服务管理队列的环境相同。它直接运行 conda 来创建环境。

该环境为在工作器上运行的每个 Deadline Cloud 会话创建一个新的 conda 虚拟环境，然后在完成后删除该环境。

Conda 会缓存下载的软件包，这样就不需要再次下载了，但是每个会话都必须将所有软件包链接到环境中。

该环境定义了三个脚本，这些脚本在 Deadline Cloud 在工作人员上启动会话时运行。第一个脚本在调用`onEnter`操作时运行。它调用另外两个来设置环境变量。脚本运行完毕后，conda 环境将可用，并设置了所有指定的环境变量。

有关该示例的最新版本，请参阅上存储库中的 conda_queue_env_console_equeuvalen t.yaml。deadline-cloud-samples GitHub

如果您想使用 conda 频道中没有的应用程序，则可以在 Amazon S3 中创建一个 conda 频道，然后为该应用程序构建自己的软件包。请参阅 使用 S3 创建 conda 频道，了解更多信息。

从 conda-forge 获取开源库

本节介绍如何使用`conda-forge`频道中的开源库。以下示例是使用 `polars` Python 包的作业模板。

该任务设置队列环境中定义的`CondaPackages`和`CondaChannels`参数，这些参数告诉 Deadline Cloud 在哪里获取软件包。

作业模板中设置参数的部分是：

```
- name: CondaPackages
  description: A list of conda packages to install. The job expects a Queue Environment
 to handle this.
  type: STRING
  default: polars
- name: CondaChannels
  description: A list of conda channels to get packages from. The job expects a Queue
 Environment to handle this.
  type: STRING
  default: conda-forge
```

有关完整示例作业模板的最新版本，请参阅 stage _1_self_contained_template/template.yaml。有关加载 conda 包的队列环境的最新版本，请参阅上存储库中的 conda_queue_env_console _equeuval ent.yaml。deadline-cloud-samples GitHub

Blender从 deadline-cloud 频道获取

以下示例显示了Blender从 `deadline-cloud` conda 频道获取的作业模板。该频道支持 Deadline Cloud 为数字内容创作软件提供的提交者，但您可以使用相同的渠道加载软件供自己使用。

有关该`deadline-cloud`频道提供的软件列表，请参阅 De AWS adline Cloud 用户指南中的默认队列环境。

此作业设置队列环境中定义的`CondaPackages`参数，告诉 Deadline Cloud 加载Blender到环境中。

作业模板中设置参数的部分是：

```
- name: CondaPackages
  type: STRING
  userInterface:
    control: LINE_EDIT
    label: Conda Packages
    groupLabel: Software Environment
  default: blender
  description: >
    Tells the queue environment to install Blender from the deadline-cloud conda
 channel.
```

有关完整示例作业模板的最新版本，请参阅 blender_render/template .yaml。有关加载 conda 包的队列环境的最新版本，请参阅上存储库中的 conda_queue_env_console _equeuval ent.yaml。deadline-cloud-samplesGitHub

使用其他软件包管理器

Deadline Cloud 的默认包管理器是 conda。如果您需要使用其他包管理器（例如）Rez，则可以创建一个自定义队列环境，其中包含使用您的包管理器的脚本。

此示例队列环境提供的行为与控制台用于服务管理队列的环境相同。它将 conda 包管理器替Rez换为。

该环境定义了三个脚本，这些脚本在 Deadline Cloud 在工作人员上启动会话时运行。第一个脚本在调用`onEnter`操作时运行。它调用另外两个来设置环境变量。脚本运行完毕后，设置了Rez所有指定的环境变量的环境即可使用。

该示例假设您有一个客户管理的队列，该队列使用共享文件系统来处理 Rez 软件包。

有关该示例的最新版本，请参阅上存储库中的 rez_queue_env .yaml。deadline-cloud-samplesGitHub

使用 S3 创建 conda 频道

如果你的作业需要运行`deadline-cloud`或`conda-forge`频道上没有的应用程序，你可以托管一个自定义 conda 频道来提供你自己的软件包。当你在 Deadline Cloud（De AWS adline Cloud）控制台中创建队列时，控制台默认会添加一个 conda 队列环境。要使您的包可用于作业，请将自定义频道添加到队列环境中。

conda 频道是静态托管的内容，您可以通过多种方式托管，包括在文件系统或亚马逊简单存储服务 (Amazon S3) 存储桶中。如果您的 Deadline Cloud 场使用共享文件系统存储资产，则可以使用其上的任何路径作为频道名称。您可以将频道托管在 Amazon S3 存储桶中，以便使用 AWS Identity and Access Management (IAM) 权限进行更广泛的访问。

你可以在本地构建和测试软件包，然后将其发布到频道。在本地构建软件包是一种无需设置基础架构即可开始迭代包构建配方的简便方法。您也可以使用 Deadline Cloud 包构建队列来构建软件包并将其发布到频道。软件包生成队列简化了针对多个操作系统和加速器配置的软件包的维护。您可以随时随地更新版本并提交全套软件包构建。

您可以通过多种方式为工作室和 Deadline Cloud 农场配置频道。您可以拥有一个 Amazon S3 通道，并将所有工作站和场主机配置为使用该通道。您也可以拥有多个频道，并使用 AWS DataSync (DataSync) 设置镜像。例如，您的 Deadline Cloud 包构建队列可以发布到 Amazon S3 频道，该频道会在本地镜像工作站和本地服务器场主机。

主题

在本地构建和测试软件包

在将软件包发布到 Amazon S3 或在 Deadline Cloud 服务器场上设置 CI/CD 自动化之前，您可以使用本地文件系统通道在工作站上构建和测试 conda 软件包。这种方法可以让您在本地快速迭代配方并验证软件包。

该`rattler-build publish`命令构建配方，将生成的包复制到频道，然后一步为该频道编制索引。当你以本地文件系统目录为目标时，如果该目录不存在，则会自动`rattler-build`创建和初始化该频道。

以下说明使用了 Deadline Cloud 示例存储库中的 Blender 4.5 示例配方GitHub。您可以替换样本存储库中的其他食谱，也可以使用自己的食谱。

先决条件

在开始之前，请在您的工作站上安装以下工具：

- pixi — 用于安装`rattler-build`和测试软件包的软件包管理器。从 pixi.sh 安装 pixi。
- rattler-build — Deadline Cloud conda 配方使用的软件包构建工具。安装 pixi 后，运行以下命令进行安装`rattler-build`。

```
pixi global install rattler-build
```

- git — 克隆示例存储库所必需的。在Windows，git f or Windows 还提供了一个`bash`外壳，这是一些Windows示例配方所需要的。

生成包并将其发布到本地频道

在此过程中，您将克隆 Deadline Cloud 示例存储库，然后使用`rattler-build publish`生成该包并将其发布到本地文件系统频道。

> **Note**
>
> 大型应用程序可能需要数十 GB 的可用磁盘空间来存放源存档、提取的文件和生成输出。确保使用具有足够可用空间的磁盘，用于软件包生成输出。

生成软件包并将其发布到本地频道

1. 克隆截止日期云示例存储库。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
```

2. 切换到 `conda_recipes` 目录。

```
cd deadline-cloud-samples/conda_recipes
```

3. 运行以下命令构建 Blender 4.5 配方并将软件包发布到本地频道目录。

   在 Linux and 上macOS，运行以下命令。

```
rattler-build publish blender-4.5/recipe/recipe.yaml \
    --to file://$HOME/my-conda-channel \
    --build-number=+1
```

   在 Windows (cmd) 上，运行以下命令。

```
rattler-build publish blender-4.5/recipe/recipe.yaml ^
    --to file://%USERPROFILE%/my-conda-channel ^
    --build-number=+1
```

   该`rattler-build publish`命令执行以下操作：

   - 根据配方构建软件包。
   - 如果频道目录不存在，则创建该目录。
   - 将软件包文件复制到频道。

- 为频道编制索引，以便包管理员可以找到该软件包。

  如果你的包配方依赖于来自特定渠道（例如 conda-forge）的软件包，请`-c conda-forge`添加到命令中。

> **关于内部版本号**
>
> 该`--build-number=+1`选项会根据目标频道中已有的版本号自动选择下一个内部版本号。最佳做法是永远不要覆盖频道中的软件包。如果软件包在其他情况下会有相同的文件名，请务必使用新的内部版本号进行构建。当你构建到制作频道或过渡频道时，使用可以`--build-number=+1`实现这一点。
> 如果要直接控制内部版本号，则可以使用特定值对其进行设置，例如`--build-number=7`。
> 如果省略该选项，则`rattler-build`使用`recipe.yaml`文件中定义的内部版本号。

有关更多信息`rattler-build publish`，请参阅 rattler- build 发布文档。

调试版本

如果构建失败，则`rattler-build`会保留构建目录，以便您可以进行调查。运行以下命令在构建环境中打开一个交互式 shell，其中所有环境变量都与生成期间一样设置。

```
rattler-build debug shell
```

在调试 shell 中，您可以修改文件、运行单独的构建命令以及添加依赖关系以隔离问题。有关更多信息，请参阅 rattler-build 文档中的调试构建。

测试软件包

生成并发布软件包后，创建一个临时 pixi 项目。使用该项目从本地渠道安装软件包并验证其是否正常运行。

测试软件包

1. 创建临时测试目录并使用本地频道初始化 pixi 项目。

   在 Linux an macOS d 上，运行以下命令。

   ```
   mkdir package-test-env
   ```

```
cd package-test-env
pixi init --channel file://$HOME/my-conda-channel
```

在 Windows (cmd) 上，运行以下命令。

```
mkdir package-test-env
cd package-test-env
pixi init --channel file://%USERPROFILE%/my-conda-channel
```

2. 将软件包添加到项目中。

```
pixi add blender=4.5
```

3. 验证软件包是否正常运行。

```
pixi run blender --version
```

该`pixi run`命令激活项目目录的 conda 环境，并在其中运行指定的命令。环境保留在项目目录中，因此您可以从其他终端使用相同的`pixi run`命令。

如果您对软件包感到满意，则可以将软件包发布到 Amazon S3 conda 频道，这样 Deadline Cloud 工作人员就可以安装该软件包。请参阅将软件包发布到 S3 conda 频道。

从频道中移除软件包

避免从用于生产的渠道中删除软件包，因为锁文件通过哈希引用特定的软件包。删除软件包会阻止从这些锁文件中重新创建环境。对于开发和测试频道，您可以从频道目录中删除特定软件包，然后重新索引该频道，从而移除该包。`.conda`首先，安装`rattler-index`。

```
pixi global install rattler-index
```

然后删除包文件并重新索引频道。

在 Linux an macOS d 上，运行以下命令。

```
rm $HOME/my-conda-channel/linux-64/blender-4.5.0-hb0f4dca_1.conda
rattler-index fs $HOME/my-conda-channel
```

在 Windows (cmd) 上，运行以下命令。

```
del %USERPROFILE%\my-conda-channel\win-64\blender-4.5.0-hb0f4dca_1.conda
rattler-index fs %USERPROFILE%\my-conda-channel
```

Package 文件存储在平台特定的子目录中linux-64，例如、win-64或。osx-arm64列出这些子目录的内容，找到要删除的软件包的确切文件名。

清理

测试完成后，您可以移除测试项目和本地频道。

清理测试资源

1. 移除测试项目目录。

   在 Linux and 上macOS，运行以下命令。

   ```
   rm -rf package-test-env
   ```

   在 Windows (cmd) 上，运行以下命令。

   ```
   rmdir /s /q package-test-env
   ```

2. 移除本地 conda 频道目录。

   在 Linux and 上macOS，运行以下命令。

   ```
   rm -rf $HOME/my-conda-channel
   ```

   在 Windows (cmd) 上，运行以下命令。

   ```
   rmdir /s /q %USERPROFILE%\my-conda-channel
   ```

3. （可选）移除包含已生成软件包文件的rattler-build输出目录。

   在 Linux and 上macOS，运行以下命令。

   ```
   rm -rf deadline-cloud-samples/conda_recipes/output
   ```

   在 Windows (cmd) 上，运行以下命令。

```
rmdir /s /q deadline-cloud-samples\conda_recipes\output
```

将包发布到 Amazon S3 conda 频道

您可以将 conda 包发布到亚马逊简单存储服务 (Amazon S3) 存储桶，这样 AWS Deadline Cloud（Deadline Cloud）工作人员就可以安装它们来运行作业。该`rattler-build publish`命令在 Amazon S3 上的使用方式与使用本地文件系统通道的方式相同。该命令可以生成配方并发布结果，也可以发布您已经构建的包文件。在这两种情况下，该命令都会将包上传到存储桶，并一步为频道编制索引。

该`rattler-build publish`命令 AWS 使用标准凭证链进行身份验证，因此它像使用任何 AWS 工具一样使用您的 AWS 配置。有关配置凭据的更多信息，请参阅 AWS Command Line Interface (AWS CLI) 用户指南中的配置和凭据文件设置。

先决条件

在将包发布到 Amazon S3 之前，请完成以下先决条件：

- pixi 和 rattler-build — 从 pixi.sh 安装 pixi，然后安装。`rattler-build`

```
pixi global install rattler-build
```

- git — 克隆示例存储库所必需的。在Windows，git f or Windows 还提供了一个`bash`外壳，这是一些Windows示例配方所需要的。
- 亚马逊 S3 存储桶 — 用作 conda 通道的亚马逊 S3 存储桶。您可以使用 Deadline Cloud 场中的任务附件存储桶，也可以创建单独的存储桶。
- AWS 凭证-使用`aws configure`命令或命令在您的工作站上配置凭据。`aws login`有关更多信息，请参阅 AWS Command Line Interface 用户指南中的设置 AWS CLI。
- IAM 权限 —（可选）要缩小您的证书所拥有的权限范围，您可以使用 AWS Identity and Access Management (IAM) 策略，该策略仅授予对 Amazon S3 存储桶和您使用的频道前缀的以下权限（例如，`/Conda/*`）：
  - `s3:GetObject`
  - `s3:PutObject`
  - `s3:DeleteObject`
  - `s3:ListBucket`

- s3:GetBucketLocation

将包裹发布到 Amazon S3 频道

与s3://目标rattler-build publish一起使用，将包发布到您的 Amazon S3 conda 频道。如果存储桶中不存在该频道，则会自动rattler-build初始化该频道。在开始之前，请确保您已完成先决条件。

以下示例发布了 Deadline Cloud 示例存储库中的 Blender 4.5 示例配方GitHub。您可以替换样本存储库中的其他食谱，也可以使用自己的食谱。

> **Note**
>
> 大型应用程序可能需要数十 GB 的可用磁盘空间来存放源存档、提取的文件和生成输出。请确保使用具有足够可用空间的磁盘，用于软件包生成输出。

将包发布到 Amazon S3 频道

1. 克隆截止日期云示例存储库。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
```

2. 切换到 conda_recipes 目录。

```
cd deadline-cloud-samples/conda_recipes
```

3. 运行如下命令。将 *amzn-s3-demo-bucket*替换为存储桶名称。

```
rattler-build publish blender-4.5/recipe/recipe.yaml --to s3://amzn-s3-demo-bucket/
Conda/Default --build-number=+1
```

前/Conda/Default缀组织存储桶内的频道。您可以使用不同的前缀，但该前缀在引用该频道的所有命令和队列配置中必须一致。

> **关于内部版本号**
>
> 该--build-number=+1选项会根据目标频道中已有的版本号自动选择下一个内部版本号。最佳做法是永远不要覆盖频道中的软件包。如果软件包在其他情况下会有相同的文件名，请务必使用新的内部版本号进行构建。当你构建到制作频道或过渡频道时，使用可以--build-number=+1实现这一点。
>
> 如果要直接控制内部版本号，则可以使用特定值对其进行设置，例如--build-number=7。
>
> 如果省略该选项，则rattler-build使用recipe.yaml文件中定义的内部版本号。

如果你的包配方依赖于来自特定渠道（例如 conda-forge）的软件包，请-c conda-forge添加到命令中。

您也可以发布已经生成的包文件，例如，来自本地版本的.conda文件。将 *amzn-s3-demo-bucket*替换为存储桶名称。

```
rattler-build publish output/linux-64/blender-4.5.0-hb0f4dca_0.conda \
    --to s3://amzn-s3-demo-bucket/Conda/Default
```

初始化频道或重新索引频道

当您使用rattler-build publish发布包时，如果频道尚不存在，则该命令会自动初始化该频道。在大多数情况下，您无需手动初始化频道或重新索引频道。

在以下情况下，您可能需要手动初始化频道或重新索引频道：

- 例如，您希望在发布任何包之前创建一个空频道，以验证您的 Deadline Cloud 队列环境是否可以连接到该频道。
- 您直接使用 Amazon S3 工具上传或删除.conda文件rattler-build publish，而不是使用，并且频道索引已过期。

初始化空频道

要初始化空频道，请创建一个repodata.json文件并将其上传到频道前缀的noarch子目录。将 *amzn-s3-demo-bucket*替换为存储桶名称。

```
echo '{"info":{"subdir":"noarch"},"packages":{},"packages.conda":{},"removed":
[],"repodata_version":1}' > empty_channel_repodata.json
```

```
aws s3api put-object --body empty_channel_repodata.json --key Conda/Default/noarch/
repodata.json --bucket amzn-s3-demo-bucket
```

前/Conda/Default缀必须与您的队列环境使用的频道前缀匹配。初始化频道后，您可以使用将包发布到频道rattler-build publish。

为频道重新编制索引

如果频道索引已过期，则rattler-index使用从频道中的包文件中重建索引。首先，安装rattler-index。

```
pixi global install rattler-index
```

然后重新索引频道。将 *amzn-s3-demo-bucket*替换为存储桶名称。

```
rattler-index s3 s3://amzn-s3-demo-bucket/Conda/Default
```

测试软件包

发布软件包后，创建一个临时 pixi 项目以验证软件包是否正常运行。该项目通过 Amazon S3 渠道安装软件包。

测试软件包

1. 创建临时测试目录并使用 Amazon S3 通道初始化一个 pixi 项目。将 *amzn-s3-demo-bucket*替换为存储桶名称。

   ```
   mkdir package-test-env
   cd package-test-env
   pixi init --channel s3://amzn-s3-demo-bucket/Conda/Default
   ```

2. 将软件包添加到项目中。

   ```
   pixi add blender=4.5
   ```

3. 验证软件包是否正常运行。

   ```
   pixi run blender --version
   ```

   该pixi run命令激活项目目录的 conda 环境，并在其中运行指定的命令。环境保留在项目目录中，因此您可以从其他终端使用相同的pixi run命令。

从频道中移除软件包

避免从用于生产的渠道中删除软件包，因为锁文件通过哈希引用特定的软件包。删除软件包会阻止从这些锁文件中重新创建环境。对于开发和测试频道，您可以移除特定软件包，方法是从存储桶中删除该.conda文件，然后重新索引该频道。

删除包文件，然后重新索引频道。将 *amzn-s3-demo-bucket*替换为存储桶名称。

```
aws s3 rm s3://amzn-s3-demo-bucket/Conda/Default/linux-64/blender-4.5.0-
hb0f4dca_1.conda
```

删除文件后，重新索引频道以更新频道元数据。有关说明，请参阅为频道重新编制索引。

Package 文件存储在平台特定的子目录中linux-64，例如、win-64或。osx-arm64要列出子目录中的软件包，请运行以下命令。

```
aws s3 ls s3://amzn-s3-demo-bucket/Conda/Default/linux-64/
```

清理

测试完成后，移除测试项目目录。

清理测试资源

- 移除测试项目目录。

  在 Linux and 上macOS，运行以下命令。

  ```
  rm -rf package-test-env
  ```

  在 Windows (cmd) 上，运行以下命令。

  ```
  rmdir /s /q package-test-env
  ```

调试版本

如果构建失败，则rattler-build保留构建目录以便您可以进行调查。运行以下命令在构建环境中打开一个交互式 shell，其中所有环境变量都与生成期间一样设置。

```
rattler-build debug shell
```

在调试 shell 中，您可以修改文件、运行单独的构建命令以及添加依赖关系以隔离问题。有关更多信息，请参阅 rattler-build 文档中的调试构建。

为其他平台构建软件包

该`rattler-build publish`命令为运行该命令的工作站的操作系统生成软件包。如果您的 Deadline Cloud 队列使用的操作系统与您的工作站不同，或者您的软件包有其他主机要求，则您可以选择以下选项：

- `rattler-build publish`在与目标操作系统匹配的主机上运行。例如，使用Linux正在运行的亚马逊弹性计算云 (Amazon EC2) 实例为Linux队列构建软件包。
- 使用 Deadline Cloud 软件包生成队列在目标平台上自动构建。请参见创建包生成队列。
- （高级）使用交叉编译为不同于工作站的不同平台构建软件包。有关更多信息，请参阅 rattler-build 文档中的交叉编译。

后续步骤

将包发布到 Amazon S3 conda 频道后，将您的 Deadline Cloud 队列配置为使用该频道：

- 为自定义 conda 包配置生产队列权限 — 向您的生产队列授予对 Amazon S3 conda 频道的只读访问权限。
- 向队列环境添加 conda 通道-配置队列环境以安装来自 Amazon S3 conda 通道的软件包。

为自定义 conda 包配置生产队列权限

您的生产队列需要队列的 S3 存储桶中`/Conda`前缀的只读权限。打开与生产队列关联的角色的 AWS Identity and Access Management (IAM) 页面，然后使用以下命令修改策略：

1. 打开 Deadline Cloud 控制台并导航到包生成队列的队列详细信息页面。
2. 选择队列服务角色，然后选择编辑队列。
3. 滚动至队列服务角色部分，然后选择在 IAM 控制台中查看此角色。
4. 从权限策略列表中，`AmazonDeadlineCloudQueuePolicy`为您的队列选择。
5. 从 “权限” 选项卡中，选择 “编辑”。

6. 向队列服务角色添加一个新部分，如下所示。*111122223333*用您自己的存储桶和账户替换*amzn-s3-demo-bucket*和。

```
{
   "Effect": "Allow",
   "Sid": "CustomCondaChannelReadOnly",
   "Action": [
    "s3:GetObject",
    "s3:ListBucket"
   ],
   "Resource": [
    "arn:aws:s3:::amzn-s3-demo-bucket",
    "arn:aws:s3:::amzn-s3-demo-bucket/Conda/*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "111122223333"
    }
   }
  },
```

向队列环境中添加 conda 频道

要使用 S3 conda 频道，您需要将`s3://amzn-s3-demo-bucket/Conda/Default`频道位置添加到提交给 Deadline Cloud 的任务`CondaChannels`参数中。Deadline Cloud 提供的提交者提供了用于指定自定义 conda 频道和套餐的字段。

您可以通过编辑生产队列的 conda 队列环境来避免修改每个作业。执行以下步骤：

1. 打开 Deadline Cloud 控制台并导航到生产队列的队列详细信息页面。
2. 选择 “环境” 选项卡。
3. 选择 Conda 队列环境，然后选择 “编辑”。
4. 选择 JSON 编辑器，然后在脚本中找到的参数定义`CondaChannels`。
5. 编辑该行，`default: "deadline-cloud"`使其从新创建的 S3 conda 通道开始：

```
default: "s3://amzn-s3-demo-bucket/Conda/Default deadline-cloud"
```

默认情况下，服务管理舰队为 conda 启用灵活的频道优先级。对于请求新频道和频道中`blender=4.5`是否都有 Blender 4.5 的任务，将从`deadline-cloud`频道列表中第一个频道中提取包裹。如果在第一个频道中找不到指定的包版本，则将按软件包版本的顺序检查后续频道。

对于客户管理的舰队，您可以使用 Deadline Cloud 示例存储库中的一个 c onda 队列环境示例来启用 conda 软件包。GitHub

为应用程序或插件创建 conda 软件包

conda 软件包是用任何语言编写的软件的压缩档案。Conda 支持各种操作系统和架构组合，因此您可以将完整的应用程序（如BlenderMaya、和）与 Python 和其他语言的库Nuke一起打包。有关 conda 软件包的更多信息，请参阅 conda 文档中的软件包。

要使用 conda 软件包，您需要将其安装到虚拟环境中。conda 虚拟环境有一个安装软件包的前缀目录。安装软件包时会使用硬链接或文件重新链接（如果支持），因此使用相同的软件包创建多个环境不会占用大量额外的磁盘空间。要使用虚拟环境，请将其激活以设置环境变量。激活运行软件包提供的脚本，使每个软件包都有机会修改 PATH 或其他环境变量。Conda 包通常包含应用程序或库，但灵活的激活意味着它们也可以指向安装在共享文件系统上的应用程序。

制作自定义软件包涉及三个阶段：配方包含构建说明，软件包是构建的工件（`.conda`或`.tar.bz2`文件），频道托管要安装的软件包。该`rattler-build publish`命令可以处理所有三个步骤——它可以将配方构建到包中并发布到频道，也可以直接使用包工件来发布它。

conda-forge 社区维护着各种开源软件的软件包配方，并在频道中托管软件包工件。`conda-forge`您可以将队列配置`conda-forge`为包源，然后构建依赖于 conda-forge 软件包运行的自定义包。对于 Linux，conda-forge 托管了一个完整的编译器工具链，包括 CUDA 支持，并选择了一致的编译和链接选项。你可以在自己的配方中使用 conda-forge 包作为依赖项，也可以将它们与自定义软件包一起安装在同一个环境中。

您可以将整个应用程序（包括依赖关系）组合成一个 conda 包。Deadline Cloud 在截止日期云频道中为服务管理的车队提供的软件包使用这种二进制重新打包方法。这会组织与安装文件相同的文件，以适合 conda 虚拟环境。

> **Note**
>
> 大型应用程序可能需要数十 GB 的可用磁盘空间来存放源存档、提取的文件和生成输出。确保使用具有足够可用空间的磁盘，用于软件包生成输出。

Package 打包应用程序

在为 conda 重新打包应用程序时，有两个目标：

- 应用程序的大多数文件应与主要 conda 虚拟环境结构分开。然后，环境可以将应用程序与来自其他来源（例如 conda- forge）的软件包混合在一起。
- 激活 conda 虚拟环境后，应该可以从 PATH 环境变量中访问该应用程序。

为 conda 重新打包应用程序

1. 编写 conda 构建配方，将应用程序安装到类似的子目录中。`$CONDA_PREFIX/opt/<application-name>`这会将其与标准前缀目录（如`bin`和`lib`）区分开来。
2. 向添加符号链接或启动脚本`$CONDA_PREFIX/bin`以运行应用程序二进制文件。

   或者，创建 activate.d 脚本，该`conda activate`命令将运行该脚本以将应用程序的二进制目录添加到 PATH 中。如果在Windows任何可以创建环境的地方都不支持符号链接，请改用应用程序启动或 activate.d 脚本。
3. 某些应用程序依赖于 Deadline Cloud 服务管理的队列上默认未安装的库。例如，对于非交互式作业，通常不需要 X11 窗口系统，但有些应用程序仍然要求它在没有图形界面的情况下运行。您必须在创建的包中提供这些依赖关系。
4. 如果应用程序支持插件，请提供一个明确的约定，即插件包在虚拟环境中与应用程序集成时应遵循该约定。例如，Maya2026 年的示例配方记录了Maya插件的这一惯例。
5. 确保您遵守所打包的应用程序的版权和许可协议。我们建议使用私有 Amazon S3 存储桶作为您的 conda 频道，以控制分发并限制对服务器场的包访问权限。

该`deadline-cloud`频道中套餐的示例配方可在上的 De adline Cloud 示例存储库中找到GitHub。

Package 一个插件

应用程序插件可以打包成自己的 conda 包。创建插件包时，请遵循以下准则：

- 将主机应用程序包作为构建依赖项和运行依赖项包含在构建配方中`recipe.yaml`。使用版本限制，这样编译配方只能与兼容的软件包一起安装。
- 按照主机应用程序包约定注册插件。

适配器套装

某些 Deadline Cloud 应用程序集成使用适配器来扩展应用程序界面，以简化作业模板的编写。适配器是一个命令行界面，支持运行后台守护程序、报告状态和应用路径映射。有关更多信息，请参阅上的 Ope n Job Description Adaptor 运行时。GitHub例如，deadline-cloud-for-mayaon GitHub 包括集成的作业提交 GUI 和一个Maya适配器，该适配器可在服务管理的队列中作为`maya-openjd`软件包使用。

Deadline Cloud 提交者提交的作业 GUIs 包括一个`CondaPackages`参数值，该参数值指定要包含在虚拟环境中以运行作业的 conda 包。的`CondaPackages`参数值Maya通常看起来像`maya=2026.* maya-openjd=0.15.* maya-mtoa`，可能包含插件包的替代条目。当队列环境设置用于运行作业的 conda 虚拟环境时，它会解析这些软件包名称和版本限制以使其兼容，并添加它们运行所需的所有依赖包。每个适配器和插件包都指定了与之兼容的内容，包括哪些版本Maya、Python 的版本以及其他依赖项。

要使用我们的示例（例如 maya-openjd 配方）构建自己的适配器包GitHub，您可以在 conda- forge 提供的 Python 包和其他依赖项的基础上构建。您可能需要先确定截止日期和openjd-adaptor-runtime配方以满足依赖关系。

为其创建 conda 构建配方 Blender

Blender可免费使用且易于使用 conda 打包，这使其成为学习如何为 Deadline Cloud（De AWS adline Cloud）创建 conda 包的好起点。Blender基金会为多个操作系统提供应用程序档案。Deadline Cloud 示例存储库中的 Blender 4.5 示例配方将这些存档打GitHub包为 conda 包。

了解食谱

reci pe.yaml 文件以 rattler-build 模板语法定义了软件包元数据 URLs、源代码和构建选项。配方只指定一次版本号，并 URLs 根据操作系统提供不同的来源。

中的`build`部分`recipe.yaml`关闭了二进制重定位和动态共享对象 (DSO) 链接检查。这些选项控制软件包安装到任何目录前缀的 conda 虚拟环境中的工作方式。`build`本节中使用的默认值是为单独打包每个依赖库而设计的，但是在对应用程序进行二进制重新打包时，需要对其进行更改。 Blender不需要任何 RPATH 调整，因为在构建应用程序存档时考虑了可重定位性。有关添加可重定位性的示例，请参见为 Maya 创建 conda 配方。

在软件包构建过程中，运行 build.sh 或 build_win.sh 脚本将文件安装到环境中。这些脚本将安装文件复制到`$PREFIX/opt/blender`，从`$PREFIX/bin`（onLinux）创建符号链接，并设置用于配置环境变量的激活脚本，例如`BLENDER_LOCATION`。启用Windows，激活脚本会将Blender目录添加到 PATH 中，而不是创建符号链接。

Windows编译脚本使用.bat 文件bash代替 cmd.exe .bat 文件，以实现跨平台的一致性。你可以安装 git fo Windows r 来bash提供软件包构建。

该配方还包括一个deadline-cloud.yaml文件，该文件指定了用于向 Deadline Cloud 提交自动包构建任务的 conda 平台和元数据。有关更多信息，请参阅提交软件包构建任务。

构建Blender软件包

rattler-build publish用于构建 Blender 4.5 配方并将包发布到频道。您可以发布到本地文件系统通道进行测试，也可以直接发布到 Amazon S3 频道进行生产使用。如果您在本地完成了构建和测试包中的设置，请从conda_recipes目录中运行以下命令。

```
rattler-build publish blender-4.5/recipe/recipe.yaml \
    --to file://$HOME/my-conda-channel \
    --build-number=+1
```

对于其他发布选项：

- 要发布到 Amazon S3 频道，请参阅将包发布到 S3 conda 频道。
- 要使用 Deadline Cloud 包生成队列自动构建，请参阅使用 Deadline Cloud 自动生成软件包。

使用Blender渲染作业测试你的软件包

构建 Blender 4.5 包后，您可以使用渲染作业对其进行测试。如果你没有Blender场景，请从Blender演示文件页面下载 Blender 3.5-Cozy Kitchen 场景。Deadline Cloud 示例存储库包含一个blender_render作业包和一个 conda 队列环境，您可以将其用于本地和云端测试。

本地测试

您可以使用 Open Job Description CLI 在您的工作站上运行作业模板。使用安装 CLI pip。

```
pip install openjd-cli
```

在示例存储库的job_bundles目录中，运行以下命令。*/path/to/scene.blend*替换为Blender场景文件的路径。

```
openjd run blender_render/template.yaml \
    --environment ../queue_environments/conda_queue_env_pyrattler.yaml \
    -p CondaPackages=blender=4.5 \
    -p CondaChannels=file://$HOME/my-conda-channel \
```

```
    -p BlenderSceneFile=/path/to/scene.blend \
    -p Frames=1
```

该--environment选项适用 conda 队列环境，该环境使用中指定的软件包创建一个 conda 虚拟环境。CondaPackages该CondaChannels参数告诉队列环境在哪里可以找到软件包。如果您发布到 Amazon S3 频道而不是本地频道，请将file://路径替换为您的s3://频道 URL。

在截止日期云上进行测试

将生产队列配置为使用 Amazon S3 conda 通道后，您可以将渲染作业提交到 Deadline Cloud。在示例存储库的job_bundles目录中，运行以下命令。

```
deadline bundle submit blender_render \
    -p CondaPackages=blender=4.5 \
    -p BlenderSceneFile=/path/to/scene.blend \
    -p Frames=1
```

使用 Deadline Cloud 监控器来跟踪任务的进度。在监视器中，选择作业的任务，然后选择查看日志。选择 “启动 Conda 会话” 操作以验证是否已在 Amazon S3 渠道中找到该包裹。

为其创建 conda 构建配方 Autodesk Maya

与诸如此类的开源应用程序相比，诸如此类的商业应用程序Autodesk Maya引入了额外的封装要求 Blender。该Blender配方在开源许可证下打包了一个简单的可重定位档案。商业应用程序通常通过安装程序分发，需要许可证管理配置。

商业应用的注意事项

打包商业应用程序时，需要考虑以下注意事项。详细信息说明了每个细节是如何适用的Maya。

- 许可-了解应用程序的许可权和限制。您可能需要配置许可证管理系统。阅读有关云权限的Autodesk订阅权益常见问题解答，了解云权限Maya。 Autodesk产品依赖于通常需要管理员访问权限才能配置的ProductInformation.pit文件。瘦客户机的产品功能提供了一种可重新定位的替代方案。有关更多信息，请参阅适用于 Maya MotionBuilder 的瘦客户机许可。
- 系统库依赖关系-某些应用程序依赖于未安装在服务管理的舰队工作主机上的库。 Maya取决于包括 freetype 和 fontconfig 在内的库。当这些库在系统包管理器中可用时（dnf例如 for） AL2023，则可以使用软件包管理器作为源。由于 RPM 包不是为可重定位而构建的，因此您需要使用诸如patchelf解决Maya安装前缀内的依赖关系之类的工具。
- 管理员访问权限才能进行安装-某些安装程序需要管理员访问权限。服务管理队列不提供管理员访问权限，因此您需要在单独的系统上安装应用程序，并为软件包版本创建文件存档。的Windows

安装程序Maya需要这种方法。配方中的 README.md 记录了一个使用新启动的亚马逊弹性计算云 (Amazon EC2) 实例的可重复过程。

- 插件集成 — 示例Maya包定义`MAYA_NO_HOME=1`将应用程序与用户级配置隔离开来，并在其中添加了模块搜索路径，`MAYA_MODULE_PATH`以便插件包可以在虚拟环境中放置`.mod`文件。有关完整的插件集成惯例，请参阅 Maya2026 示例配方。

了解食谱

re cipe.yaml 文件以 rattler-build 模板语法定义软件包元数据。查看该文件的以下部分：

- 来源 — 引用安装程序档案，包括 sha256 哈希值。在上Linux，源是Autodesk安装程序存档。在上Windows，源代码包括安装程序存档和Maya为云部署Autodesk做准备的`cleanMayaForCloud.py`脚本。更改源文件时更新哈希值，例如打包新版本时。
- build — 关闭默认的二进制重定位和 DSO 链接检查，因为自动机制无法正常用于Maya使用的库和二进制目录。开启后Linux，配方中包含`patchelf`了用于手动设置相对依赖项的构建依赖项 RPATHs。
- about — 有关用于浏览或处理 conda 频道内容的应用程序的元数据。

生成脚本（build.sh 为Linux，build_win.sh 为Windows）包含解释每个步骤的注释。这些脚本执行以下关键任务：

- 解压缩安装程序 — 将Maya安装文件提取到 conda 前缀中。由于安装程序格式不同，Linux和Windows脚本对此的处理方式有所不同。有关详细信息，请参阅构建脚本。
- 安装系统库依赖项 — 开Linux启后，脚本会下载并提取Maya需要但服务管理的队列主机上不存在的系统库。该脚本将这些库复制到Maya`lib`目录中，以便它们在 conda 环境中可用。
- RPATHs 使用 patchelf 设置相对路径 — 启用Linux，脚本用于`patchelf --add-rpath`向共享`$ORIGIN`库添加相对路径。这种方法遵循了 conda 的建议，即永远不要`LD_LIBRARY_PATH`在 conda 环境中使用。该脚本修补多个目录级别（`lib`、`lib/python*/site-packages`、`lib/python*/lib-dynload`）的库，以便每个库都能找到其相对于自己位置的依赖关系。该配方遵循设置`DT_RUNPATH`而不是的最佳实践`DT_RPATH`，它允许`LD_LIBRARY_PATH`在需要调试时覆盖搜索路径。
- 配置瘦客户机许可 — 该脚本设置了所记录的瘦客户机许可，Autodesk这样`ProductInformation.pit`文件就可以位于 conda 环境中，而不必要求系统级管理员访问权限。

- 设置激活脚本-脚本创建用于设置环境变量的激活和停用脚本MAYA_LOCATION，包括MAYA_VERSIONMAYA_NO_HOME、和MAYA_MODULE_PATH。开启后Windows，脚本会同时生成.sh和.bat激活文件，因为 Deadline Cloud 示例队列环境用于bash激活环境Windows。

构建Maya软件包

在构建Maya软件包之前，请从您的Autodesk帐户下载Maya安装程序。对于Linux，请将存档直接放到conda_recipes/archive_files目录中。对于Windows，请按照 README.md 中的步骤创建存档。

rattler-build publish用于生成和发布软件包。该Maya配方需要patchelf作为构建依赖项Linux，可从 conda- forge 获得。添加-c conda-forge以使依赖关系在构建期间可用。在conda_recipes目录中，运行以下命令。

```
rattler-build publish maya-2026/recipe/recipe.yaml \
    --to file://$HOME/my-conda-channel \
    --build-number=+1 \
    -c conda-forge
```

对于其他发布选项：

- 要发布到 Amazon S3 频道，请参阅将包发布到 S3 conda 频道。
- 要使用 Deadline Cloud 包生成队列自动构建，请参阅使用 Deadline Cloud 自动生成软件包。要同时生成Linux和Windows软件包，请使用submit-package-job脚本中的--all-platforms选项。

要使用Maya和渲染转盘示例Arnold，请同时构建MtoA插件和Maya适配器包。发布所有三个包后，你可以使用 Deadline Cloud 示例存储库中的带Maya/Arnold任务包的转盘提交测试渲染作业。请参见使用 Maya 渲染作业测试您的软件包。

为适配器创建 conda 构建配方 Maya

该maya-openjd软件包提供了Maya与 AWS 截止日期云（Deadline Cloud）作业提交集成的适配器。当你使用 Deadline Cloud 提交者 GUI 提交Maya渲染作业时，该CondaPackages参数包含在maya包maya-openjd旁边。在作业会话期间Maya，适配器负责启动、传递渲染参数以及管理应用程序生命周期。有关适配器的更多信息，请参阅适配器包。

了解食谱

maya-openjd 示例配方从发布到 PyPI 的deadline-cloud-for-maya源包中构建适配器。reci pe.yaml 使用pip将软件包安装到 conda 环境中。

配方依赖于 Python 和 Deadline Cloud 示例存储库中的另外两个包，你需要先构建这两个包：

- 截止日期 — Deadline Cloud 客户端库。
- openjd-adaptor-runtime— Open Job Description 适配器运行时。

Python 和其他依赖项可从 conda-forge 获得，因此在构建适配器包时请`-c conda-forge`添加到`rattler-build publish`命令中。

构建适配器包

该`maya-openjd`软件包依赖于 Deadline Cloud 示例存储库中的另外两个软件包。从`conda_recipes`目录中按顺序生成所有三个软件包。每个命令的`-c conda-forge`选项都是满足 Python 和 Python 库的配方依赖关系。

生成`deadline`软件包。

```
rattler-build publish deadline/recipe/recipe.yaml \
    --to file://$HOME/my-conda-channel \
    --build-number=+1 \
    -c conda-forge
```

生成`openjd-adaptor-runtime`软件包。

```
rattler-build publish openjd-adaptor-runtime/recipe/recipe.yaml \
    --to file://$HOME/my-conda-channel \
    --build-number=+1 \
    -c conda-forge
```

生成`maya-openjd`软件包。

```
rattler-build publish maya-openjd/recipe/recipe.yaml \
    --to file://$HOME/my-conda-channel \
    --build-number=+1 \
    -c conda-forge
```

对于其他发布选项：

- 要发布到 Amazon S3 频道，请参阅将包发布到 S3 conda 频道。
- 要使用 Deadline Cloud 包生成队列自动构建，请参阅使用 Deadline Cloud 自动生成软件包。

为插件创建 conda 构建配Autodesk Maya to Arnold (MtoA)方

该Maya to Arnold (MtoA)插件将Arnold渲染器添加为其中的Maya一个选项。mToA 示例配方演示了如何将插件打包为与宿主应用程序包集成的单独的 conda 包。

了解食谱

rec ipe.yaml 指定了编译和运行要求对maya软件包的依赖关系。此依赖项使用版本限制，因此插件只能安装兼容Maya版本。

该食谱使用与食Maya谱相同的源存档。生成脚本将在配置Maya软件包的$PREFIX/usr/autodesk/maya$MAYA_VERSION/modules目录中安装MtoA并创建一个mtoa.mod文件。MAYA_MODULE_PATH Arnold并Maya使用相同的许可技术，因此该Maya软件包已包含Arnold所需的许可信息。

构建MtoA软件包

在编译Maya软件包之前先构建MtoA软件包，因为MtoA这取决于编译Maya时。rattler-build publish用于生成和发布软件包。在conda_recipes目录中，运行以下命令。

```
rattler-build publish maya-mtoa-2026/recipe/recipe.yaml \
    --to file://$HOME/my-conda-channel \
    --build-number=+1
```

在解析依赖关系时，该rattler-build publish命令使用目标频道作为最高优先级的频道，因此您之前发布的maya软件包会自动可用。

对于其他发布选项：

- 要发布到 Amazon S3 频道，请参阅将包发布到 S3 conda 频道。
- 要使用 Deadline Cloud 包生成队列自动构建，请参阅使用 Deadline Cloud 自动生成软件包。

使用Maya渲染作业测试你的软件包

构建MayaMtoA、和maya-openjd包之后，您可以使用渲染作业对其进行测试。Deadline Cloud 示例存储库包含一个带有Maya/Arnold任务包的转盘，该转盘使用Maya和渲染动画。Arnold任务包还用于FFmpeg 对视频进行编码，该视频可从该conda-forge频道获得。

本地测试

您可以使用 Open Job Description CLI 在您的工作站上运行作业模板。使用安装 CLI pip。

```
pip install openjd-cli
```

在示例存储库的job_bundles目录中，运行以下命令。该ErrorOnArnoldLicenseFail=false参数告诉使用水印Arnold进行渲染，而不是在没有可用许可证时失败。

```
openjd run turntable_with_maya_arnold/template.yaml \
    --environment ../queue_environments/conda_queue_env_pyrattler.yaml \
    -p CondaPackages="maya maya-mtoa maya-openjd ffmpeg" \
    -p CondaChannels="file://$HOME/my-conda-channel conda-forge" \
    -p ErrorOnArnoldLicenseFail=false \
    -p FrameRange=1-5
```

该--environment选项适用 conda 队列环境，该环境使用中指定的软件包创建一个 conda 虚拟环境。CondaPackages该CondaChannels参数包括您的自定义软件包的本地渠道和conda-forge的本地渠道ffmpeg。如果您发布到 Amazon S3 频道而不是本地频道，请将file://路径替换为您的s3://频道 URL。

作业完成后，渲染的输出位于turntable_with_maya_arnold/output/目录中。

在截止日期云上进行测试

将生产队列配置为使用 Amazon S3 conda 通道后，将渲染作业提交到 Deadline Cloud。将conda-forge通道添加到 conda 队列环境中的CondaChannels参数中，以提供适配器所需的 Python 依赖项的源代码ffmpeg和 Python 依赖关系。在示例存储库的job_bundles目录中，运行以下命令。

```
deadline bundle submit turntable_with_maya_arnold
```

使用 Deadline Cloud 监控器来跟踪任务的进度。在监视器中，选择作业的任务，然后选择查看日志。选择 La unch Conda 会话操作以验证是否在 Amazon S3 频道中找到了mayamaya-mtoa、和maya-openjd软件包。

使用截止日期云自动生成软件包

对于 CI/CD 工作流程或需要为多个操作系统构建软件包时，可以创建 Deadline Cloud 软件包生成队列。队列计划在您的队列上构建任务，队列生成软件包并将其发布到您的亚马逊简单存储服务

(Amazon S3) Simple Storage Service conda 频道。这简化了在所有必需配置中为软件版本维护持续构建软件包的过程。

您可以使用 AWS CloudFormation (CloudFormation) 模板创建包生成队列，也可以从 Deadline Cloud 控制台手动创建包构建队列。该 CloudFormation 模板部署了一个完整的服务器场，其中包含生产队列和已配置的软件包生成队列。通过控制台创建队列可以更好地控制各个设置。

使用创建包生成队列 CloudFormation

您可以使用 CloudFormation 模板来创建包含包构建队列的 Deadline Cloud 场。该模板使用私有 Amazon S3 conda 通道配置生产队列和包生成队列。

在部署模板之前，请创建一个 Amazon S3 存储桶来存放任务附件和您的 conda 频道。您可以从 Amazon S3 控制台创建存储桶。部署模板时需要存储桶名称。

部署 CloudFormation 模板

1. 从 Deadline Cloud 示例存储库中下载 deadline-cloud-starter-farm-template.yaml 模板。 GitHub
2. 在CloudFormation 控制台中，选择创建堆栈，然后选择使用新资源（标准）。
3. 选择上传模板文件的选项，然后上传该`deadline-cloud-starter-farm-template.yaml`文件。
4. 输入堆栈的名称，例如**`StarterFarm`**，并提供任务附件的 Amazon S3 存储桶的名称和 conda 频道。
5. 按照 CloudFormation 控制台步骤完成堆栈创建。

有关模板参数和自定义选项的更多信息，请参阅 De adline Cloud 示例存储库中的入门农场自述文件。GitHub

从控制台创建包生成队列

按照《De adline Cloud 用户指南》中创建队列中的说明进行操作。进行以下更改：

- 在步骤 5 中，选择现有的 Amazon S3 存储桶。指定根文件夹名称（例如），**`DeadlineCloudPackageBuild`**以便生成工件与普通的 Deadline Cloud 附件分开。
- 在步骤 6 中，您可以将包构建队列与现有队列相关联，或者如果当前队列不合适，则可以创建全新的队列。
- 在步骤 9 中，为包生成队列创建一个新的服务角色。您将修改权限，为队列提供上传包和重新索引 conda 频道所需的权限。

配置软件包生成队列权限

要允许包构建队列访问队列的 Amazon S3 存储桶中的/Conda前缀，您必须修改队列的角色以授予其 read/write 访问权限。该角色需要以下权限，以便包生成任务可以上传新包并重新编入频道索引。

- `s3:GetObject`
- `s3:PutObject`
- `s3:ListBucket`
- `s3:GetBucketLocation`
- `s3:DeleteObject`

1. 打开 Deadline Cloud 控制台并导航到包生成队列的队列详细信息页面。
2. 选择队列服务角色，然后选择编辑队列。
3. 滚动至队列服务角色部分，然后选择在 IAM 控制台中查看此角色。
4. 从权限策略列表中，AmazonDeadlineCloudQueuePolicy为您的队列选择。
5. 从 “权限” 选项卡中，选择 “编辑”。
6. 向队列服务角色添加一个新部分，如下所示。*111122223333*用您自己的存储桶和账户替换*amzn-s3-demo-bucket*和。

```
{
   "Effect": "Allow",
   "Sid": "CustomCondaChannelReadWrite",
   "Action": [
    "s3:GetObject",
    "s3:PutObject",
    "s3:DeleteObject",
    "s3:ListBucket",
    "s3:GetBucketLocation"
   ],
   "Resource": [
    "arn:aws:s3:::amzn-s3-demo-bucket",
    "arn:aws:s3:::amzn-s3-demo-bucket/Conda/*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "111122223333"
    }
   }
```

```
  },
```

提交软件包构建任务

创建包生成队列并配置队列权限后，您可以提交任务来构建 conda 包。De adline Cloud 示例存储库中的`submit-package-job`脚本GitHub提交了 conda 配方的构建作业。

您需要以下项：

- 您的工作站上安装了 De adline Cloud CLI。
- 活动的 De AWS adline 云监视器（截止日期云监视器）登录会话。
- Deadline Cloud 示例存储库的克隆。

提交软件包生成任务

1. 打开 Deadline Cloud 配置 GUI，将默认服务器场和队列设置为你的包构建队列。

   ```
   deadline config gui
   ```

2. 切换到示例存储库中的`conda_recipes`目录。

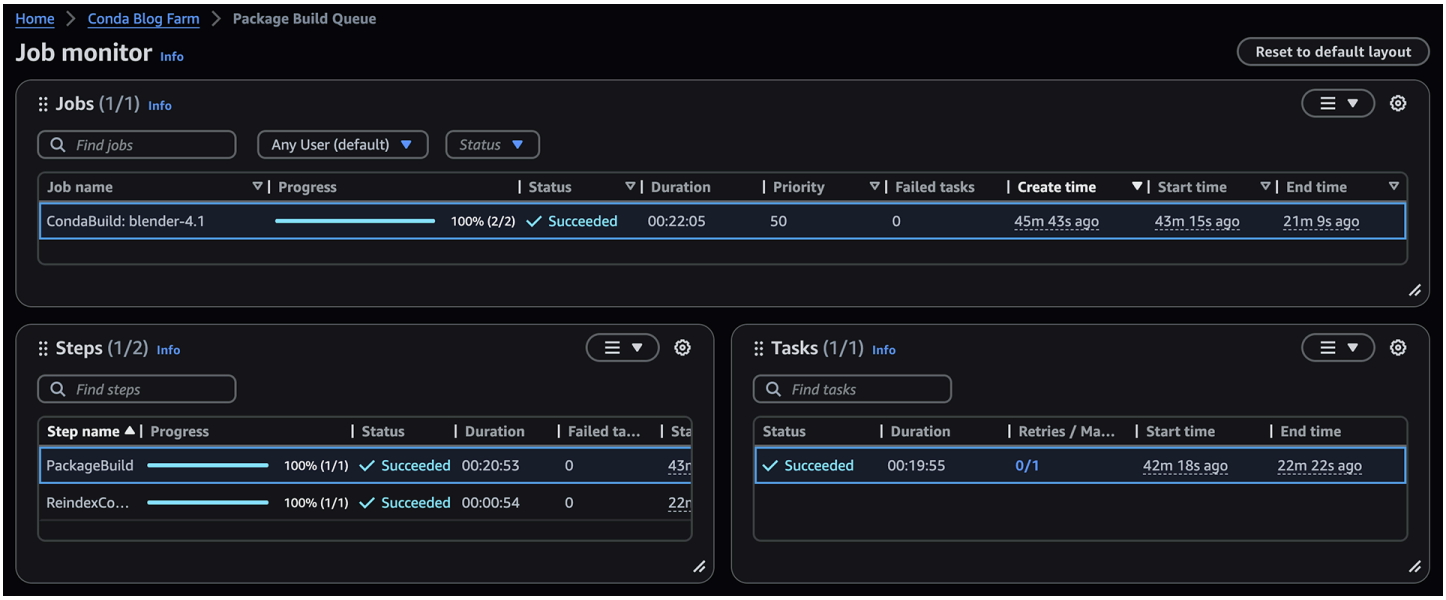
   ```
   cd deadline-cloud-samples/conda_recipes
   ```

3. 使用配方目录运行`submit-package-job`脚本。以下示例构建 Blender 4.5 配方。

   ```
   ./submit-package-job blender-4.5/
   ```

   如果配方需要您尚未下载的源存档，则脚本会提供下载说明。下载存档并再次运行脚本。

提交作业后，使用 Deadline Cloud 监控器查看任务的进度和状态。



监视器显示了工作的两个步骤：构建软件包，然后重新索引 conda 频道。右键单击包构建步骤的任务并选择 “查看日志” 时，监视器会显示会话操作：

- 同步附件-复制输入作业附件或装载虚拟文件系统。
- 启动 Conda — 队列环境操作。构建任务未指定 conda 包，因此此操作很快就会完成。
- La@@ unch CondaBuild Env — 使用构建 conda 包和重新索引频道所需的软件创建 conda 虚拟环境。
- 任务运行 — 生成软件包并将结果上传到 Amazon S3。

操作运行时，它们会向 Amazon CloudWatch (CloudWatch) 发送日志。任务完成后，选择 “查看所有任务的日志” 以查看有关环境设置和拆卸的其他日志。

使用管理员权限运行主机配置脚本

主机配置脚本允许您对服务管理的车队工作人员执行管理任务，例如安装软件。这些脚本以更高的权限运行（开sudo启Linux，管理员开启Windows），使您可以灵活地为系统配置工作器。

Deadline Cloud 在工作人员进入STARTING状态之后和运行任何任务之前运行脚本。

> **Important**
>
> 该脚本以提升的权限运行。您有责任确保脚本不会引入任何安全问题。
> 使用主机配置脚本时，您负责监控队列的运行状况。

主机配置脚本的常见用途包括：

- 安装需要管理员访问权限的软件
- 安装Docker容器
- 安装第三方云存储解决方案，例如LucidLink. 有关演练，请参阅 For M&E 博客上的 Deadline Cloud LucidLink 使用服务托管队列脚本 AWS 进行设置。

您可以使用控制台或使用创建和更新主机配置脚本 AWS CLI。

Console

1. 在舰队详细信息页面上，选择配置选项卡。
2. 在 “脚本” 字段中，输入要使用提升权限运行的脚本。您可以选择 “导入” 从您的工作站加载脚本。
3. 设置运行脚本的超时时间（以秒为单位）。默认值为 300 秒（5 分钟）。
4. 选择 “保存更改” 以保存脚本。

Create with CLI

使用以下 AWS CLI 命令创建带有主机配置脚本的队列。用您的信息替换*placeholder*文本。

```
aws deadline create-fleet \
--farm-id farm-12345 \
--display-name "fleet-name" \
--max-worker-count 1 \
--configuration '{
"serviceManagedEc2": {
  "instanceCapabilities": {
    "vCpuCount": {"min": 2},
    "memoryMiB": {"min": 4096},
    "osFamily": "linux",
    "cpuArchitectureType": "x86_64"
  },
  "instanceMarketOptions": {"type":"spot"}
}
}' \
--role-arn arn:aws:iam::111122223333:role/role-name \
--host-configuration '{ "scriptBody": "script body", "scriptTimeoutSeconds": timeout
 value}'
```

Update with CLI

使用以下 AWS CLI 命令更新队列的主机配置脚本。用您的信息替换*placeholder*文本。

```
aws deadline update-fleet \
--farm-id farm-12345 \
--fleet-id fleet-455678 \
--host-configuration '{ "scriptBody": "script body", "scriptTimeoutSeconds": timeout
 value}'
```

以下脚本演示：

- 脚本可用的环境变量
- 这些 AWS 凭据在外壳中起作用
- 该脚本正在提升的 shell 中运行

Linux

使用以下脚本显示脚本正在使用`root`权限运行：

```
# Print environment variables
set
# Check AWS Credentials
aws sts get-caller-identity
```

Windows

使用以下PowerShell脚本显示脚本正在以管理员权限运行：

```
Get-ChildItem env: | ForEach-Object { "$($_.Name)=$($_.Value)" }
aws sts get-caller-identity
function Test-AdminPrivileges {
  $currentUser = New-Object
 Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurrent())
  $isAdmin =
 $currentUser.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)

  return $isAdmin
}
```

```
if (Test-AdminPrivileges) {
  Write-Host "The current PowerShell session is elevated (running as
 Administrator)."
} else {
  Write-Host "The current PowerShell session is not elevated (not running as
 Administrator)."
}
exit 0
```

主机配置脚本疑难解答

运行主机配置脚本时：

- 成功时：工作人员负责工作
- 失败时（非零退出代码或崩溃）：
  - 工作人员关机了

  队列使用最新的主机配置脚本自动启动新的工作程序

要监视脚本，请执行以下操作：

1. 在 Deadline Cloud 控制台中打开舰队页面。
2. 选择查看工作人员以打开 Deadline Cloud 监视器。
3. 在 “监控” 页面中查看工作人员的状态。

> **Tip**
>
> 在测试主机配置脚本时，请将队列的最大工作器数设置为 1，以避免在迭代脚本时启动多个工作程序。

重要提示：

- 由于错误而关闭的工作人员不在监视器中的工作人员列表中。使用 CloudWatch Logs 查看以下日志组中的工作日志：

```
/aws/deadline/farm-XXXXX/fleet-YYYYY
```

在该日志组中，查找名为的流worker-*ZZZZZ*。

- CloudWatch 日志会根据您配置的保留期保留工作人员日志。

监控主机配置脚本的执行情况

使用主机配置脚本，您可以完全控制 Deadline Cloud 工作程序。您可以安装任何软件包、重新配置操作系统参数或挂载共享文件系统。借助此高级功能和 Deadline Cloud 扩展到数千名工作人员的功能，您可以监控配置脚本何时成功执行或失败。

我们建议使用以下解决方案来监控主机配置脚本的执行情况。

CloudWatch 日志监控

所有队列主机配置日志都流式传输到队列的 CloudWatch 日志组，特别是流式传输到工作人员的 CloudWatch 日志流。例如，`/aws/deadline/farm-123456789012/fleet-777788889999`是 farm `123456789012`、fleet 的日志组`777788889999`。

例如，每个 worker 都会配置一个专用的日志流`worker-123456789012`。主机配置日志包括日志标语，例如正在运行主机配置脚本和已完成运行主机配置脚本，退出代码：0。脚本的退出代码包含在已完成的横幅中，可以使用 CloudWatch 工具进行查询。

CloudWatch 日志见解

CloudWatch Logs Insights 提供了分析日志信息的高级功能。例如，以下 Log Insights 查询解析主机配置退出代码，按时间排序：

```
fields @timestamp, @message, @logStream, @log
| filter @message like /Finished running Host Configuration Script/
| parse @message /exit code: (?<exit_code>\d+)/
| display @timestamp, exit_code
| sort @timestamp desc
```

有关 Lo CloudWatch gs Insights 的更多信息，请参阅 Amazon Logs 用户指南中的使用 CloudWatch CloudWatch 日志见解分析日志数据。

工作代理结构化日志

Deadline Cloud 的工作代理将结构化 JSON 日志发布到 CloudWatch。工作人员代理提供各种结构化日志，用于分析工作人员的健康状况。有关更多信息，请参阅 De adline Cloud 工作者代理登录 GitHub。

结构化日志的属性被解压缩为 Log Insights 中的字段。您可以使用此 CloudWatch 功能来统计和分析主机配置启动失败。例如，可以使用 count 和 bin 查询来确定故障发生的频率：

```
fields @timestamp, @message, @logStream, @log
| sort @timestamp desc
| filter message like /Worker Agent host configuration failed with exit code/
| stats count(*) by exit_code, bin(1h)
```

CloudWatch 指标和警报的指标过滤器

您可以设置 CloudWatch 指标筛选条件以从日志中生成 CloudWatch 指标。指标筛选器允许您创建警报和仪表板，用于监控主机配置脚本的执行情况。

创建指标筛选条件

1. 打开控制 CloudWatch 台。
2. 在导航窗格中，选择日志，然后选择日志组。
3. 选择您的舰队的日志组。
4. 选择 Create metric filter（创建指标筛选条件）。
5. 使用以下方法之一定义您的过滤器模式：
   - 对于成功指标：

     ```
     {$.message = "*Worker Agent host configuration succeeded.*"}
     ```

   - 对于失败指标：

     ```
     {$.exit_code != 0 && $.message = "*Worker Agent host configuration failed with
      exit code*"}
     ```

6. 选择 “下一步” 以创建具有以下值的指标：
   - 指标命名空间：您的指标命名空间（例如，**MyDeadlineFarm**）
   - 指标名称：您请求的指标名称（例如，**host_config_failure**）
   - 指标值：**1**（每个实例的计数为 1）
   - 默认值：留空
   - 单位：**Count**

创建指标筛选器后，您可以将标准 CloudWatch 警报配置为在主机配置失败率上升时采取措施，或者将指标添加到 CloudWatch 仪表板中以进行 day-to-day操作和监控。

有关更多详细信息，请参阅 Amazon CloudWatch 日志用户指南中的筛选条件和模式语法。

在截止日期云中使用软件许可证

Deadline Cloud 提供了两种为您的工作提供软件许可证的方法：

- 基于使用量的许可 (UBL)-根据您的车队处理任务所用的小时数进行跟踪和计费。没有固定的许可证数量，因此您的车队可以根据需要进行扩展。UBL 是服务管理车队的标准配置。对于客户管理的车队，您可以连接适用于 UBL 的 Deadline Cloud 许可证端点。UBL 为你的 Deadline Cloud 工作人员提供渲染许可证，但它不为你的 DCC 应用程序提供许可证。
- 自带许可证 (BYOL) — 允许您在服务或客户管理的车队中使用现有的软件许可证。对于基于 Deadline Cloud 使用量的许可证不支持的软件，您可以使用 BYOL 连接到许可证服务器。通过连接到自定义许可证服务器，您可以将 BYOL 与服务托管队列一起使用。

主题

将 BYOL 和 UBL 结合起来

您可以将 BYOL 和 UBL 合并，这样您的员工就可以先使用您的现有许可证，并在自带设备许可证用完时自动回退到基于 Deadline Cloud 使用的许可证。当您的现有许可证数量有限，但在工作负载高峰期需要扩展到该容量以外时，这种方法非常有用。

组合许可的工作原理

配置组合许可时，队列环境会设置许可证环境变量，以便在 UBL 许可证端点之前列出 BYOL 许可证服务器。大多数第三方应用程序按照许可证服务器在环境变量中出现的顺序检查许可服务器。当工作人员申请许可证时，应用程序会首先联系您的 BYOL 许可证服务器。如果没有 BYOL 许可证可用，则应用程序会回退到 UBL 许可证端点。

中提供的 BYOL 队列环境模板Connect 将服务管理的车队连接到自定义许可服务器会自动配置此回退行为。队列环境中的 Python 脚本会将您的 BYOL 许可证服务器地址添加到现有 UBL 许可环境变量之前。要使用组合许可，请在队列环境脚本中保留您想要回退行为的产品的UBL部分。

要对特定产品仅使用 BYOL 而不使用 UBL 回退，请从脚本中删除该产品的 UBL 部分，然后将许可证环境变量直接添加到队列环境的`variables`部分。例如，要仅将 BYOL 用于 Cinema 4D，请从脚本中删除 Cinema 4D 部分并`g_licenseServerRLM: 127.0.0.1:7057`添加到该部分。`variables`

示例：使用带有 UBL 后备功能的 BYOL Cinema 4D 许可证

假设一家工作室在其本地网络的许可服务器上拥有现有 Cinema 4D 许可证。该工作室希望使用这些许可证进行 Deadline Cloud 渲染，但也希望通过在使用所有 BYOL 许可证时回退到 UBL 来扩展其许可证数量。

要配置此设置，请按照中的Connect 将服务管理的车队连接到自定义许可服务器步骤对队列环境模板进行以下更改：

使用 UBL 后备配置 BYOL Cinema 4D 许可证

1. 将`LicenseInstanceId`参数设置为有权访问 Cinema 4D 许可服务器的许可服务器或代理的亚马逊弹性计算云 (Amazon EC2) 实例 ID。
2. 将`LicensePorts`参数设置为包括端口`7057`（Cinema 4D RLM 许可证端口）。
3. 在 Python 脚本中，保留将 BYOL 服务器置于 UBL 配置之前的 Cinema 4D 部分：

   ```
   # Cinema4D
   os.environ["g_licenseServerRLM"] = f"127.0.0.1:7057;
   {os.environ.get('g_licenseServerRLM', '')}"
   print(f"openjd_env: g_licenseServerRLM={os.environ['g_licenseServerRLM']}")
   ```

   此配置设置`g_licenseServerRLM`为`127.0.0.1:7057;`*`UBL_endpoint`*`:7057`。Cinema 4D 首先会检查 BYOL 服务器。`127.0.0.1:7057`如果没有可用的许可证，Cinema 4D 将回退到 UBL 端点。
4. 删除您不使用的产品（例如 Arnold、Nuke 或 SideFX）的部分，以保持配置整洁。

如果您还有其他产品仅使用 BYOL 而不使用 UBL 回退，请将这些许可环境变量直接添加到队列环境的`variables`部分，并从 Python 脚本中删除相应的部分。

组合许可的注意事项

使用组合许可时，请记住以下注意事项：

- 某些应用程序不支持在单个环境变量中使用多个许可证服务器。例如，V-Ray 改用 XML 配置文件。队列环境模板单独处理 V-Ray 配置。有关更多信息，请参阅中队列环境模板中的 Connect 将服务管理的车队连接到自定义许可服务器 V-Ray 部分。
- 环境变量中许可证服务器的顺序决定了优先级。首先列出 BYOL 服务器，以便先使用现有许可证，然后再使用 UBL 许可证。
- 在 Windows worker 上，使用分号 (;) 而不是冒号 () 分隔环境变量中的许可证服务器条目。:有关配置许可证环境变量的更多信息，请参阅Connect 将客户管理的车队连接到许可证端点。
- 要在客户管理的队列中使用 UBL 回退，请设置许可证端点。有关更多信息，请参阅 Connect 将客户管理的车队连接到许可证端点。

Connect 将服务管理的车队连接到自定义许可服务器

您可以自带许可证服务器与 Deadline Cloud 服务托管队列一起使用。要自带许可证，您可以使用服务器场中的队列环境配置许可证服务器。要配置许可证服务器，您应该已经设置了服务器场和队列。

如何连接到软件许可证服务器取决于设备群的配置和软件供应商的要求。通常，您可以通过以下两种方式之一访问服务器：

- 直接发送到许可证服务器。您的工作人员使用 Internet 从软件供应商的许可证服务器获取许可证。您的所有工作人员都必须能够连接到服务器。
- 通过许可证代理。您的工作人员连接到本地网络中的代理服务器。仅允许代理服务器通过 Internet 连接到供应商的许可证服务器。

按照以下说明，您可以使用 Amazon EC2 Systems Manager (SSM) 将端口从工作程序实例转发到您的许可证服务器或代理实例。在下面的示例中，如果您的许可证服务器无法提供许可证，则将使用 Deadline Cloud 基于使用情况的许可。删除不适用于您的管道或您不想在许可证用尽后使用基于使用情况的许可的产品的部分。

主题

步骤 1：配置队列环境

您可以在队列中配置队列环境以访问您的许可证服务器。首先，使用以下方法之一确保您的 AWS 实例配置为具有许可证服务器访问权限：

- 许可证服务器-实例直接托管许可证服务器。
- 许可证代理-实例具有对许可证服务器的网络访问权限，并将许可证服务器端口转发到许可证服务器。有关如何配置许可证代理实例的详细信息，请参阅步骤 2：（可选）许可证代理实例设置。

有关配置许可证环境变量的信息，请参见步骤 3：将渲染应用程序连接到端点。对于自定义许可服务器设置，许可证服务器地址仍为本地主机，而不是 Amazon VPC 终端节点。

向队列角色添加所需权限

1. 从 Deadl ine Cloud 控制台中，选择前往控制面板。
2. 在控制面板中，选择服务器场，然后选择要配置的队列。
3. 从队列详细信息 > 服务角色中，选择角色。
4. 选择 “添加权限”，然后选择 “创建内联策略”。
5. 选择 JSON 策略编辑器，然后将以下文本复制并粘贴到编辑器中。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "ssm:StartSession"
            ],
            "Resource": [
                "arn:aws:ssm:us-east-1::document/AWS-
StartPortForwardingSession",
                "arn:aws:ec2:us-east-1:111122223333:instance/instance_id"
            ]
        }
    ]
}
```

6. 在保存新策略之前，请替换策略文本中的以下值：

- region替换为农场所在的地 AWS 区
- instance_id替换为您正在使用的许可证服务器或代理实例的实例 ID
- account_id替换为包含您的农场的 AWS 账号

7. 选择下一步。
8. 对于策略名称，请输入**LicenseForwarding**。
9. 选择创建策略以保存您的更改并使用所需权限创建策略。

向队列中添加新的队列环境

1. 如果尚未选择 De adline Cloud 控制台，请选择 “前往控制面板”。
2. 在控制面板中，选择服务器场，然后选择要配置的队列。
3. 选择 “队列环境” > “操作” > “使用 YAML 新建”。
4. 将以下文本复制并粘贴到 YAML 脚本编辑器中。

Windows

```
specificationVersion: "environment-2023-09"
parameterDefinitions:
 - name: LicenseInstanceId
   type: STRING
   description: >
    The Instance ID of the license server/proxy instance
   default: ""
 - name: LicenseInstanceRegion
   type: STRING
   description: >
    The region containing this farm
   default: ""
 - name: LicensePorts
   type: STRING
   description: >
    Comma-separated list of ports to be forwarded to the license server/proxy
    instance. Example: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
   default: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
environment:
 name: BYOL License Forwarding
 variables:
```

```
  example_LICENSE: 2701@localhost
 script:
  actions:
   onEnter:
    command: bash
    args: [ "{{Env.File.Enter}}" ]
   onExit:
    command: bash
    args: [ "{{Env.File.Exit}}" ]
  embeddedFiles:
   - name: Enter
     type: TEXT
     runnable: True
     data: |
      curl "https://s3.amazonaws.com/session-manager-downloads/plugin/latest/
windows/SessionManagerPlugin.zip" -o "{{Session.WorkingDirectory}}/ssm-
plugin.zip"
      powershell -Command "Expand-Archive -Path '{{Session.WorkingDirectory}}/
ssm-plugin.zip' -DestinationPath '{{Session.WorkingDirectory}}/ssm-plugin'
 -Force; Expand-Archive -Path '{{Session.WorkingDirectory}}/ssm-plugin/
package.zip' -DestinationPath '{{Session.WorkingDirectory}}/ssm-plugin/package'
 -Force"
      conda activate
      python "{{Env.File.StartSession}}" "{{Session.WorkingDirectory}}/ssm-
plugin/package/bin/session-manager-plugin.exe"
   - name: Exit
     type: TEXT
     runnable: True
     data: |
      echo Killing SSM Manager Plugin PIDs: $BYOL_SSM_PIDS
      for pid in ${BYOL_SSM_PIDS//,/ }; do kill $pid; done
   - name: StartSession
     type: TEXT
     data: |
      import boto3
      import json
      import subprocess
      import sys
      import os
      import tempfile

      instance_id = "{{Param.LicenseInstanceId}}"
      region = "{{Param.LicenseInstanceRegion}}"
      license_ports_list = "{{Param.LicensePorts}}".split(",")
```

```
      ssm_client = boto3.client("ssm", region_name=region)
      pids = []

      for port in license_ports_list:
        session_response = ssm_client.start_session(
          Target=instance_id,
          DocumentName="AWS-StartPortForwardingSession",
          Parameters={"portNumber": [port], "localPortNumber": [port]}
        )

        cmd = [
          sys.argv[1],
          json.dumps(session_response),
          region,
          "StartSession",
          "",
          json.dumps({"Target": instance_id}),
          f"https://ssm.{region}.amazonaws.com"
        ]

        process = subprocess.Popen(cmd, stdout=subprocess.DEVNULL,
 stderr=subprocess.DEVNULL)
        pids.append(process.pid)
        print(f"SSM Port Forwarding Session started for port {port}")

      print(f"openjd_env: BYOL_SSM_PIDS={','.join(str(pid) for pid in pids)}")

      # Enabling UBL after the BYOL has run out requires prepending the BYOL
 configuration to the existing license setup
      # Remove the sections that do not apply to your pipeline, or you do not
 want to use UBL after exhausting the BYOL licenses.
      # The port numbers used may not match what your license server is serving.

      # Arnold
      os.environ["ADSKFLEX_LICENSE_FILE"] = f"2701@localhost;
{os.environ.get('ADSKFLEX_LICENSE_FILE', '')}"
      print(f"openjd_env:
 ADSKFLEX_LICENSE_FILE={os.environ['ADSKFLEX_LICENSE_FILE']}")

      # Cinema4D
      os.environ["g_licenseServerRLM"] = f"localhost:7057;
{os.environ.get('g_licenseServerRLM', '')}"
```

```
      print(f"openjd_env:
 g_licenseServerRLM={os.environ['g_licenseServerRLM']}")

      # Nuke
      os.environ["foundry_LICENSE"] = f"6101@localhost;
{os.environ.get('foundry_LICENSE', '')}"
      print(f"openjd_env: foundry_LICENSE={os.environ['foundry_LICENSE']}")

      # SideFX
      os.environ["SESI_LMHOST"] = f"localhost:1715;
{os.environ.get('SESI_LMHOST', '')}"
      print(f"openjd_env: SESI_LMHOST={os.environ['SESI_LMHOST']}")

      # Redshift and Red Giant
      os.environ["redshift_LICENSE"] = f"7054@localhost;7055@localhost;
{os.environ.get('redshift_LICENSE', '')}"
      print(f"openjd_env: redshift_LICENSE={os.environ['redshift_LICENSE']}")

      # V-Ray doesn't support multiple license servers in a single environment
 variable
      # See https://documentation.chaos.com/space/LIC5/125050770/Sharing+a
+License+Configuration+in+a+Network
      vray_license = os.environ.get('VRAY_AUTH_CLIENT_SETTINGS', '')
      xml_content = """<VRLClient>
        <LicServer>
          <Host>localhost</Host>
          <Port>30304</Port>"""

      if vray_license and vray_license.startswith('licset://'):
          server_parts = vray_license.removeprefix('licset://').split(':')
          if len(server_parts) >= 2:
              xml_content += f"""
          <Host1>{server_parts[0]}</Host1>
          <Port1>{server_parts[1]}</Port1>"""

      xml_content += """
          <User></User>
          <Pass></Pass>
        </LicServer>
      </VRLClient>"""

      temp_dir = tempfile.gettempdir()
      xml_path = os.path.join(temp_dir, 'vrlclient.xml')
```

```
      with open(xml_path, 'w') as f:
          f.write(xml_content)

      os.environ["VRAY_AUTH_CLIENT_FILE_PATH"] = temp_dir
      print(f"openjd_env:
 VRAY_AUTH_CLIENT_FILE_PATH={os.environ['VRAY_AUTH_CLIENT_FILE_PATH']}")

      # Clear the existing VRAY_AUTH_CLIENT_SETTINGS so only the vrlclient.xml
 file is used.
      os.environ["VRAY_AUTH_CLIENT_SETTINGS"] = ''
      print(f"openjd_env:
 VRAY_AUTH_CLIENT_SETTINGS={os.environ['VRAY_AUTH_CLIENT_SETTINGS']}")

      # Print out the created xml file's contents
      print(f"V-Ray configuration file: {xml_path}")
      with open(xml_path, 'r') as f:
          print(f"{f.read()}")
```

Linux

```
specificationVersion: "environment-2023-09"
parameterDefinitions:
 - name: LicenseInstanceId
   type: STRING
   description: >
    The Instance ID of the license server/proxy instance
   default: ""
 - name: LicenseInstanceRegion
   type: STRING
   description: >
    The region containing this farm
   default: ""
 - name: LicensePorts
   type: STRING
   description: >
    Comma-separated list of ports to be forwarded to the license server/proxy
    instance. Example: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
   default: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
environment:
 name: BYOL License Forwarding
 variables:
```

```
  example_LICENSE: 2701@localhost
 script:
  actions:
   onEnter:
    command: bash
    args: [ "{{Env.File.Enter}}" ]
   onExit:
    command: bash
    args: [ "{{Env.File.Exit}}" ]
  embeddedFiles:
   - name: Enter
     type: TEXT
     runnable: True
     data: |
      curl https://s3.amazonaws.com/session-manager-downloads/plugin/
latest/linux_64bit/session-manager-plugin.rpm -Ls | rpm2cpio - | cpio -iv
 --to-stdout ./usr/local/sessionmanagerplugin/bin/session-manager-plugin >
 {{Session.WorkingDirectory}}/session-manager-plugin
      chmod +x {{Session.WorkingDirectory}}/session-manager-plugin
      conda activate
      python {{Env.File.StartSession}} {{Session.WorkingDirectory}}/session-
manager-plugin
   - name: Exit
     type: TEXT
     runnable: True
     data: |
      echo Killing SSM Manager Plugin PIDs: $BYOL_SSM_PIDS
      for pid in ${BYOL_SSM_PIDS//,/ }; do kill $pid; done
   - name: StartSession
     type: TEXT
     data: |
      import boto3
      import json
      import subprocess
      import sys
      import os
      import tempfile

      instance_id = "{{Param.LicenseInstanceId}}"
      region = "{{Param.LicenseInstanceRegion}}"
      license_ports_list = "{{Param.LicensePorts}}".split(",")

      ssm_client = boto3.client("ssm", region_name=region)
      pids = []
```

```
      for port in license_ports_list:
        session_response = ssm_client.start_session(
          Target=instance_id,
          DocumentName="AWS-StartPortForwardingSession",
          Parameters={"portNumber": [port], "localPortNumber": [port]}
        )

        cmd = [
          sys.argv[1],
          json.dumps(session_response),
          region,
          "StartSession",
          "",
          json.dumps({"Target": instance_id}),
          f"https://ssm.{region}.amazonaws.com"
        ]

        process = subprocess.Popen(cmd, stdout=subprocess.DEVNULL,
 stderr=subprocess.DEVNULL)
        pids.append(process.pid)
        print(f"SSM Port Forwarding Session started for port {port}")

      print(f"openjd_env: BYOL_SSM_PIDS={','.join(str(pid) for pid in pids)}")

      # Enabling UBL after the BYOL has run out requires prepending the BYOL
 configuration to the existing license setup
      # Remove the sections that do not apply to your pipeline, or you do not
 want to use UBL after exhausting the BYOL licenses.
      # The port numbers used may not match what your license server is serving.

      # Arnold
      os.environ["ADSKFLEX_LICENSE_FILE"] = f"2701@localhost:
{os.environ.get('ADSKFLEX_LICENSE_FILE', '')}"
      print(f"openjd_env:
 ADSKFLEX_LICENSE_FILE={os.environ['ADSKFLEX_LICENSE_FILE']}")

      # Nuke
      os.environ["foundry_LICENSE"] = f"6101@localhost:
{os.environ.get('foundry_LICENSE', '')}"
      print(f"openjd_env: foundry_LICENSE={os.environ['foundry_LICENSE']}")

      # SideFX
```

```
      os.environ["SESI_LMHOST"] = f"localhost:1715;
{os.environ.get('SESI_LMHOST', '')}"
      print(f"openjd_env: SESI_LMHOST={os.environ['SESI_LMHOST']}")

      # Redshift and Red Giant
      os.environ["redshift_LICENSE"] = f"7054@localhost:7055@localhost:
{os.environ.get('redshift_LICENSE', '')}"
      print(f"openjd_env: redshift_LICENSE={os.environ['redshift_LICENSE']}")

      # V-Ray doesn't support multiple license servers in a single environment
 variable
      # See https://documentation.chaos.com/space/LIC5/125050770/Sharing+a
+License+Configuration+in+a+Network
      vray_license = os.environ.get('VRAY_AUTH_CLIENT_SETTINGS', '')
      xml_content = """<VRLClient>
        <LicServer>
          <Host>localhost</Host>
          <Port>30304</Port>"""

      if vray_license and vray_license.startswith('licset://'):
          server_parts = vray_license.removeprefix('licset://').split(':')
          if len(server_parts) >= 2:
              xml_content += f"""
          <Host1>{server_parts[0]}</Host1>
          <Port1>{server_parts[1]}</Port1>"""

      xml_content += """
          <User></User>
          <Pass></Pass>
        </LicServer>
      </VRLClient>"""

      temp_dir = tempfile.gettempdir()
      xml_path = os.path.join(temp_dir, 'vrlclient.xml')

      with open(xml_path, 'w') as f:
          f.write(xml_content)

      os.environ["VRAY_AUTH_CLIENT_FILE_PATH"] = temp_dir
      print(f"openjd_env:
 VRAY_AUTH_CLIENT_FILE_PATH={os.environ['VRAY_AUTH_CLIENT_FILE_PATH']}")

      # Clear the existing VRAY_AUTH_CLIENT_SETTINGS so only the vrlclient.xml
 file is used.
```

```
    os.environ["VRAY_AUTH_CLIENT_SETTINGS"] = ''
    print(f"openjd_env:
VRAY_AUTH_CLIENT_SETTINGS={os.environ['VRAY_AUTH_CLIENT_SETTINGS']}")

    # Print out the created xml file's contents
    print(f"V-Ray configuration file: {xml_path}")
    with open(xml_path, 'r') as f:
        print(f"{f.read()}")
```

5. 在保存队列环境之前，请根据需要对环境文本进行以下更改：

   - 更新以下参数的默认值以反映您的环境：
     - LicenseInstanceID — 您的许可服务器或代理实例的 Amazon EC2 实例 ID
     - LicenseInstanceRegion— 包含您的农场 AWS 的地区
     - LicensePorts— 要转发到许可证服务器或代理实例的以逗号分隔的端口列表（例如 2700,2701）
   - 如果您想在自带许可证 (BYOL) 用尽后使用基于使用情况的许可 (UBL)，请确保您的许可证服务器的端口是正确的。如果您不想在 BYOL 用完后使用 UBL，请将所有必需的许可环境变量添加到变量部分。

     这些变量应 DCCs 将指向许可证服务器端口上的本地主机。例如，如果您的 Foundry 许可证服务器正在监听端口 6101，则应将变量添加为**`foundry_LICENSE: 6101@localhost`**。
6. （可选）您可以将优先级设置为 0，也可以将其更改为在多个队列环境中以不同的方式排列优先级。
7. 选择 “创建队列环境” 以保存新环境。

   设置队列环境后，提交到该队列的作业将从已配置的许可证服务器检索许可证。

步骤 2：（可选）许可证代理实例设置

除了使用许可证服务器之外，您还可以使用许可证代理。要创建许可证代理，请创建一个能够通过网络访问许可证服务器的新 Amazon Linux 2023 实例。如果需要，您可以使用 VPN 连接配置此访问权限。有关更多信息，请参阅 Amazon VPC 用户指南中的 VPN 连接。

要为 Deadline Cloud 设置许可证代理实例，请按照此过程中的步骤操作。在此新实例上执行以下配置步骤，以允许将许可证流量转发到您的许可证服务器

1. 要安装 HAProxy 软件包，请输入

```
sudo yum install haproxy
```

2. 使用以下内容更新/etc/haproxy/haproxy.cfg 配置文件的 listen license-server 部分：

   a. 将 LicensePort1 和 LicensePort2 替换为要转发到许可证服务器的端口号。添加或删除以逗号分隔的值以适应所需的端口数量。

   b. LicenseServerHost替换为许可证服务器的主机名或 IP 地址。

```
lobal
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    user        haproxy
    group       haproxy
    daemon

defaults
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s

listen license-server
     bind *:LicensePort1,*:LicensePort2
     server license-server LicenseServerHost
```

3. 要启用和启动该 HAProxy 服务，请运行以下命令：

```
sudo systemctl enable haproxy
sudo service haproxy start
```

完成这些步骤后，应将从转发队列环境发送到 localhost 的许可证请求转发到指定的许可证服务器。

第 3 步： CloudFormation 模板设置

您可以使用 CloudFormation 模板将整个服务器场配置为使用您自己的许可。

1. 修改下一步中提供的模板，将所有必需的许可环境变量添加到 “环境” 下BYOLQueue的 “变量” 部分。
2. 使用以下 CloudFormation 模板。

```
AWSTemplateFormatVersion: 2010-09-09
Description: "Create &ADC; resources for BYOL"

Parameters:
  LicenseInstanceId:
    Type: AWS::EC2::Instance::Id
    Description: Instance ID for the license server/proxy instance
  LicensePorts:
    Type: String
    Description: Comma-separated list of ports to forward to the license instance

Resources:
  JobAttachmentBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Sub byol-example-ja-bucket-${AWS::AccountId}-${AWS::Region}
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: AES256

  Farm:
    Type: AWS::Deadline::Farm
    Properties:
      DisplayName: BYOLFarm

  QueuePolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      ManagedPolicyName: BYOLQueuePolicy
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - s3:GetObject
              - s3:PutObject
              - s3:ListBucket
```

```
              - s3:GetBucketLocation
            Resource:
              - !Sub ${JobAttachmentBucket.Arn}
              - !Sub ${JobAttachmentBucket.Arn}/job-attachments/*
            Condition:
              StringEquals:
                aws:ResourceAccount: !Sub ${AWS::AccountId}
          - Effect: Allow
            Action: logs:GetLogEvents
            Resource: !Sub arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/deadline/${Farm.FarmId}/*
          - Effect: Allow
            Action:
              - s3:ListBucket
              - s3:GetObject
            Resource:
              - "*"
            Condition:
              ArnLike:
                s3:DataAccessPointArn:
                  - arn:aws:s3:*:*:accesspoint/deadline-software-*
              StringEquals:
                s3:AccessPointNetworkOrigin: VPC

  BYOLSSMPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      ManagedPolicyName: BYOLSSMPolicy
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - ssm:StartSession
            Resource:
              - !Sub arn:aws:ssm:${AWS::Region}::document/AWS-
StartPortForwardingSession
              - !Sub arn:aws:ec2:${AWS::Region}:${AWS::AccountId}:instance/
${LicenseInstanceId}


  WorkerPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
```

```
      ManagedPolicyName: BYOLWorkerPolicy
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - logs:CreateLogStream
            Resource: !Sub arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/deadline/${Farm.FarmId}/*
            Condition:
              ForAnyValue:StringEquals:
                aws:CalledVia:
                  - deadline.amazonaws.com
          - Effect: Allow
            Action:
              - logs:PutLogEvents
              - logs:GetLogEvents
            Resource: !Sub arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/deadline/${Farm.FarmId}/*


  QueueRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: BYOLQueueRole
      ManagedPolicyArns:
        - !Ref QueuePolicy
        - !Ref BYOLSSMPolicy
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - sts:AssumeRole
            Principal:
              Service:
                - credentials.deadline.amazonaws.com
                - deadline.amazonaws.com
            Condition:
              StringEquals:
                aws:SourceAccount: !Sub ${AWS::AccountId}
              ArnEquals:
                aws:SourceArn: !Ref Farm
```

```
  WorkerRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: BYOLWorkerRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AWSDeadlineCloud-FleetWorker
        - !Ref WorkerPolicy
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - sts:AssumeRole
            Principal:
              Service: credentials.deadline.amazonaws.com


  Queue:
    Type: AWS::Deadline::Queue
    Properties:
      DisplayName: BYOLQueue
      FarmId: !GetAtt Farm.FarmId
      RoleArn: !GetAtt QueueRole.Arn
      JobRunAsUser:
        Posix:
          Group: ""
          User: ""
        RunAs: WORKER_AGENT_USER
      JobAttachmentSettings:
        RootPrefix: job-attachments
        S3BucketName: !Ref JobAttachmentBucket

  Fleet:
    Type: AWS::Deadline::Fleet
    Properties:
      DisplayName: BYOLFleet
      FarmId: !GetAtt Farm.FarmId
      MinWorkerCount: 1
      MaxWorkerCount: 2
      Configuration:
        ServiceManagedEc2:
          InstanceCapabilities:
            VCpuCount:
              Min: 4
```

```
              Max: 16
            MemoryMiB:
              Min: 4096
              Max: 16384
            OsFamily: LINUX
            CpuArchitectureType: x86_64
          InstanceMarketOptions:
            Type: on-demand
      RoleArn: !GetAtt WorkerRole.Arn

  QFA:
    Type: AWS::Deadline::QueueFleetAssociation
    Properties:
      FarmId: !GetAtt Farm.FarmId
      FleetId: !GetAtt Fleet.FleetId
      QueueId: !GetAtt Queue.QueueId

  CondaQueueEnvironment:
    Type: AWS::Deadline::QueueEnvironment
    Properties:
      FarmId: !GetAtt Farm.FarmId
      Priority: 5
      QueueId: !GetAtt Queue.QueueId
      TemplateType: YAML
      Template: |
        specificationVersion: 'environment-2023-09'
        parameterDefinitions:
        - name: CondaPackages
          type: STRING
          description: >
            This is a space-separated list of conda package match specifications to
 install for the job.
            E.g. "blender=3.6" for a job that renders frames in Blender 3.6.

            See https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/
pkg-specs.html#package-match-specifications
          default: ""
          userInterface:
            control: LINE_EDIT
            label: Conda Packages
        - name: CondaChannels
          type: STRING
          description: >
```

```
            This is a space-separated list of conda channels from which to install
 packages. &ADC; SMF packages are
            installed from the "deadline-cloud" channel that is configured by
 &ADC;.

            Add "conda-forge" to get packages from the https://conda-forge.org/
 community, and "defaults" to get packages
            from Anaconda Inc (make sure your usage complies with https://
www.anaconda.com/terms-of-use).
          default: "deadline-cloud"
          userInterface:
            control: LINE_EDIT
            label: Conda Channels
        environment:
          name: Conda
          script:
            actions:
              onEnter:
                command: "conda-queue-env-enter"
                args: ["{{Session.WorkingDirectory}}/.env", "--packages",
 "{{Param.CondaPackages}}", "--channels", "{{Param.CondaChannels}}"]
              onExit:
                command: "conda-queue-env-exit"

  BYOLQueueEnvironment:
    Type: AWS::Deadline::QueueEnvironment
    Properties:
      FarmId: !GetAtt Farm.FarmId
      Priority: 10
      QueueId: !GetAtt Queue.QueueId
      TemplateType: YAML
      Template: !Sub |
        specificationVersion: "environment-2023-09"
        parameterDefinitions:
        - name: LicenseInstanceId
          type: STRING
          description: >
            The Instance ID of the license server/proxy instance
          default: ""
        - name: LicenseInstanceRegion
          type: STRING
          description: >
            The region containing this farm
          default: ""
```

```
        - name: LicensePorts
          type: STRING
          description: >
            Comma-separated list of ports to be forwarded to the license server/
proxy
            instance. Example:
 "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
          default: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
        environment:
        name: BYOL License Forwarding
        variables:
          example_LICENSE: 2701@localhost
        script:
          actions:
          onEnter:
            command: bash
            args: [ "{{Env.File.Enter}}" ]
          onExit:
            command: bash
            args: [ "{{Env.File.Exit}}" ]
          embeddedFiles:
          - name: Enter
            type: TEXT
            runnable: True
            data: |
              curl https://s3.amazonaws.com/session-manager-downloads/plugin/
latest/linux_64bit/session-manager-plugin.rpm -Ls | rpm2cpio - | cpio -iv
 --to-stdout ./usr/local/sessionmanagerplugin/bin/session-manager-plugin >
 {{Session.WorkingDirectory}}/session-manager-plugin
              chmod +x {{Session.WorkingDirectory}}/session-manager-plugin
              conda activate
              python {{Env.File.StartSession}} {{Session.WorkingDirectory}}/
session-manager-plugin
          - name: Exit
            type: TEXT
            runnable: True
            data: |
              echo Killing SSM Manager Plugin PIDs: $BYOL_SSM_PIDS
              for pid in ${BYOL_SSM_PIDS//,/ }; do kill $pid; done
          - name: StartSession
            type: TEXT
            data: |
              import boto3
              import json
```

```
              import subprocess
              import sys
              import os
              import tempfile

              instance_id = "{{Param.LicenseInstanceId}}"
              region = "{{Param.LicenseInstanceRegion}}"
              license_ports_list = "{{Param.LicensePorts}}".split(",")

              ssm_client = boto3.client("ssm", region_name=region)
              pids = []

              for port in license_ports_list:
                session_response = ssm_client.start_session(
                  Target=instance_id,
                  DocumentName="AWS-StartPortForwardingSession",
                  Parameters={"portNumber": [port], "localPortNumber": [port]}
                )

                cmd = [
                  sys.argv[1],
                  json.dumps(session_response),
                  region,
                  "StartSession",
                  "",
                  json.dumps({"Target": instance_id}),
                  f"https://ssm.{region}.amazonaws.com"
                ]

                process = subprocess.Popen(cmd, stdout=subprocess.DEVNULL,
stderr=subprocess.DEVNULL)
                pids.append(process.pid)
                print(f"SSM Port Forwarding Session started for port {port}")

              print(f"openjd_env: BYOL_SSM_PIDS={','.join(str(pid) for pid in
pids)}")

              # Enabling UBL after the "bring your own license" (BYOL) has run out
requires prepending the BYOL configuration to the existing license setup
              # Remove the sections that do not apply to your pipeline, or you do
not want to use UBL after exhausting the BYOL licenses.
              # The port numbers used may not match what your license server is
serving.
```

```
                # Arnold
                os.environ["ADSKFLEX_LICENSE_FILE"] = f"2701@localhost:
{os.environ.get('ADSKFLEX_LICENSE_FILE', '')}"
                print(f"openjd_env:
 ADSKFLEX_LICENSE_FILE={os.environ['ADSKFLEX_LICENSE_FILE']}")

                # Nuke
                os.environ["foundry_LICENSE"] = f"6101@localhost:
{os.environ.get('foundry_LICENSE', '')}"
                print(f"openjd_env: foundry_LICENSE={os.environ['foundry_LICENSE']}")

                # SideFX
                os.environ["SESI_LMHOST"] = f"localhost:1715;
{os.environ.get('SESI_LMHOST', '')}"
                print(f"openjd_env: SESI_LMHOST={os.environ['SESI_LMHOST']}")

                # Redshift and Red Giant
                os.environ["redshift_LICENSE"] = f"7054@localhost:7055@localhost:
{os.environ.get('redshift_LICENSE', '')}"
                print(f"openjd_env:
 redshift_LICENSE={os.environ['redshift_LICENSE']}")

                # V-Ray doesn't support multiple license servers in a single
 environment variable
                # See https://documentation.chaos.com/space/LIC5/125050770/Sharing+a
+License+Configuration+in+a+Network
                vray_license = os.environ.get('VRAY_AUTH_CLIENT_SETTINGS', '')
                xml_content = """<VRLClient>
                  <LicServer>
                    <Host>localhost</Host>
                    <Port>30304</Port>"""

                if vray_license and vray_license.startswith('licset://'):
                    server_parts = vray_license.removeprefix('licset://').split(':')
                    if len(server_parts) >= 2:
                        xml_content += f"""
                    <Host1>{server_parts[0]}</Host1>
                    <Port1>{server_parts[1]}</Port1>"""

                xml_content += """
                    <User></User>
                    <Pass></Pass>
                  </LicServer>
                </VRLClient>"""
```

```
            temp_dir = tempfile.gettempdir()
            xml_path = os.path.join(temp_dir, 'vrlclient.xml')

            with open(xml_path, 'w') as f:
                f.write(xml_content)

            os.environ["VRAY_AUTH_CLIENT_FILE_PATH"] = temp_dir
            print(f"openjd_env:
VRAY_AUTH_CLIENT_FILE_PATH={os.environ['VRAY_AUTH_CLIENT_FILE_PATH']}")

            # Clear the existing VRAY_AUTH_CLIENT_SETTINGS so only the
vrlclient.xml file is used.
            os.environ["VRAY_AUTH_CLIENT_SETTINGS"] = ''
            print(f"openjd_env:
VRAY_AUTH_CLIENT_SETTINGS={os.environ['VRAY_AUTH_CLIENT_SETTINGS']}")

            # Print out the created xml file's contents
            print(f"V-Ray configuration file: {xml_path}")
            with open(xml_path, 'r') as f:
                print(f"{f.read()}")
```

3. 部署 CloudFormation 模板时，请提供以下参数：

   - 使用您的许可服务器或代理实例的 Amazon EC2 实例 ID 更新 ID LicenseInstance
   - LicensePorts使用逗号分隔的要转发到许可证服务器或代理实例的端口列表更新（例如 2700,2701）
   - 通过在模板**example_LICENSE: 2700@localhost**中替换来添加许可证环境变量

4. 部署模板以使用自带许可证功能来设置您的农场。

Connect 将客户管理的车队连接到许可证端点

De AWS adline Cloud 基于使用量的许可证服务器为选定的第三方产品提供按需许可证。使用基于使用量的许可证，您可以按使用量付费。您只需按使用时间付费。基于使用情况的许可为你的 Deadline Cloud 工作人员提供渲染许可证，但不为你的 DCC 应用程序提供许可证。

只要 Deadline Cloud 工作人员可以与许可证服务器通信，基于 Deadline Cloud 使用情况的许可证服务器就可以用于任何类型的舰队。许可证服务器是在服务管理的舰队中自动设置的。只有客户管理的车队才需要进行以下设置。

要创建许可服务器，您需要为服务器场的 VPC 设置一个安全组，该组允许第三方许可证的流量。

主题

步骤 1：创建安全组

使用 Amazon VPC 控制台为您的服务器场的 VPC 创建安全组。将安全组配置为允许以下入站规则：

- Autodesk Maya 和 Arnold — 2701-2702，TCP， IPv4 IPv6
- Cinema 4D — 7057，TCP， IPv4 IPv6
- Foundry Nuke — 6101、TCP、 IPv4 IPv6
- Red Giant — 7055，TCP， IPV4
- Redshift — 7054，TCP， IPv4 IPv6
- SideFX Houdini、Mantra 和 Karma — 1715-1717 年，TCP， IPv4 IPv6
- VRay — 30304，TCP， IPV4

每条入站规则的来源都是舰队的工作人员安全组。

有关创建安全组的更多信息，请参阅 Amazon Virtual Private Cloud 用户指南中的创建安全组。

步骤 2：设置许可证端点

许可证端点为第三方产品提供对许可证服务器的访问权限。许可证请求将发送到许可证端点。端点会将它们路由到相应的许可证服务器。许可证服务器跟踪使用限制和授权。在 Deadline Cloud 中创建许可证端点将在您的 VPC 中配置一个 AWS PrivateLink 接口终端节点。这些终端节点按标准定 AWS PrivateLink 价计费。有关更多信息，请参阅AWS PrivateLink 定价。

有了相应的权限，您就可以创建许可证终端节点。有关创建许可证端点所需的策略，请参阅允许创建许可证端点的策略。

您可以在 Deadline Cloud 控制台的控制面板中创建许可证终端节点。

1. 在左侧导航窗格中，选择许可证端点，然后选择创建许可证端点。
2. 在“创建许可证端点”页面中，完成以下操作：

- 选择 VPC。
- 选择包含您的 Deadline Cloud 工作线程的子网。您最多可以选择 10 个子网。
- 选择您在步骤 1 中创建的安全组。对于更复杂的场景，您最多可以选择 10 个安全组。
- （可选）选择添加新标签并添加一个或多个标签。最多可以添加 50 个标签。

3. 选择创建许可证端点。许可证端点创建后，它会显示在许可证终端节点页面上。
4. 在计量产品部分中，选择添加产品，然后选择要添加到许可证端点的产品。选择添加。

要从许可证端点中删除产品，请在按流量计费的产品部分，选择该产品，然后选择删除。在确认中，再次选择移除。

步骤 3：将渲染应用程序连接到端点

设置许可证端点后，应用程序使用该端点的方法与使用第三方许可证服务器的方式相同。通常，您可以通过将环境变量或其他系统设置（例如 Microsoft Windows 注册表项）设置为许可证服务器的端口和地址来配置应用程序的许可证服务器。

要获取许可证终端节点 DNS 名称，请在控制台中选择许可证端点，然后在 DNS 名称部分中选择复制图标。

配置示例

Example— Autodesk Maya 和 Arnold

> **Note**
>
> 你可以一起使用 Autodesk Maya 和 Arnold，也可以单独使用。对于 Autodesk Maya 使用端口 2702，对于 Arnold 使用端口 2701。

对于 Autodesk Maya，请将环境变量ADSKFLEX_LICENSE_FILE设置为：

```
2702@VPC_Endpoint_DNS_Name
```

对于 Arnold，请将环境变量ADSKFLEX_LICENSE_FILE设置为：

```
2701@VPC_Endpoint_DNS_Name
```

对于 Autodesk Maya 和 Arnold，将环境变量ADSKFLEX_LICENSE_FILE设置为：

```
2702@VPC_Endpoint_DNS_Name:2701@VPC_Endpoint_DNS_Name
```

> **Note**
>
> 对于Windows工作人员，使用分号 (;) 代替冒号 (:) 来分隔端点。

Example— Cinema 4D

将环境变量设置g_licenseServerRLM为：

```
VPC_Endpoint_DNS_Name:7057
```

创建环境变量后，您应该能够使用与以下命令行类似的命令行来渲染图像：

```
"C:\Program Files\Maxon Cinema 4D 2025\Commandline.exe" -render ^
    "C:\Users\User\MyC4DFileWithRedshift.c4d" -frame 0 ^
    -oimage "C:\Users\Administrator\User\MyOutputImage.png
```

Example— 铸造核弹

将环境变量设置foundry_LICENSE为：

```
6101@VPC_Endpoint_DNS_Name
```

要测试许可是否正常运行，你可以在终端中运行Nuke：

```
~/nuke/Nuke14.0v5/Nuke14.0 -x
```

Example— 红巨人

将环境变量设置redshift_LICENSE为：

```
7055@VPC_Endpoint_DNS_Name
```

请注意，Red Giant 和 Redshift 具有相同的redshift_LICENSE环境变量。如果要同时使用这两个应用程序，则可以将环境变量设置为：

```
7054@VPC_Endpoint_DNS_Name:7055@VPC_Endpoint_DNS_Name
```

> **Note**
>
> 对于Windows工作人员，使用分号 (;) 代替冒号 (:) 来分隔端点。

要测试许可是否正常运行，请确保安装了 After Effects 和 Red Giant。然后，你可以使用类似于以下命令的命令来渲染项目：

```
C:\Program Files\Adobe\Adobe After Effects 2025\Support Files\aerender.exe -comp "Comp
 1" -project
         C:\Users\MyUser\myAfterEffectsProjectUsingRedGiant.aep -output
           C:\Users\MyUser\myMovieWithRedGiant.mp4
```

Example— Redshift

将环境变量设置redshift_LICENSE为：

```
7054@VPC_Endpoint_DNS_Name
```

创建环境变量后，您应该能够使用与以下命令行类似的命令行来渲染图像：

```
C:\ProgramData\redshift\bin\redshiftCmdLine.exe ^
    C:\demo\proxy\RS_Proxy_Demo.rs ^
    -oip C:\demo\proxy\images
```

Example— SideFX Houdini、Mantra 和 Karma

运行如下命令：

```
/opt/hfs19.5.640/bin/hserver -S
 "http://VPC_Endpoint_DNS_Name:1715;http://VPC_Endpoint_DNS_Name:1716;http://
VPC_Endpoint_DNS_Name:1717;"
```

要测试许可是否正常运行，你可以通过以下命令渲染 Houdini 场景：

```
/opt/hfs19.5.640/bin/hython ~/forpentest.hip -c "hou.node('/out/mantra1').render()"
```

Example– VRay

将环境变量设置VRAY_AUTH_CLIENT_SETTINGS为：

```
licset://VPC_Endpoint_DNS_Name:30304
```

将环境变量设置VRAY_AUTH_CLIENT_FILE_PATH为：

```
/null
```

要测试许可是否正常运行，您可以使用类似于以下 VRay 命令的命令来渲染图像：

```
/usr/Chaos/V-Ray/bin/vray -sceneFile=/root/my_scene.vrscene -display=0
```

步骤 4：删除许可证端点

删除客户管理的队列时，请记得删除您的许可证端点。如果您不删除许可证端点，则将继续向您收取 AWS PrivateLink 固定费用

您可以在 Deadline Cloud 控制台中从控制面板中删除许可证终端节点。

1. 从左侧导航窗格中，选择许可证终端节点。
2. 选择要删除的端点并选择删除，然后再次选择删除进行确认。

在截止日期云中使用 AI 代理

使用 AI 代理在 Deadline Cloud 中编写任务捆绑包、开发 conda 包和排除作业故障。本主题解释了什么是 AI 代理、有效使用它们的关键点以及帮助代理了解 Deadline Cloud 的资源。

AI 代理是一种使用大型语言模型 (LLM) 自主执行任务的软件工具。AI 代理可以读取和写入文件、运行命令以及根据反馈迭代解决方案。示例包括命令行工具，例如 Kiro 和集成了 IDE 的助手。

使用 AI 代理的要点

以下关键点可帮助您在将 AI 代理与 Deadline Cloud 配合使用时获得更好的结果。

- 提供基础 — AI 代理在可以访问相关文档、规格和示例时表现最佳。您可以通过将代理指向特定的文档页面、共享现有示例代码作为参考、将相关的开源存储库克隆到本地工作区以及为第三方应用程序提供文档来提供基础。
- 指定成功标准-定义代理的预期结果和技术要求。例如，当您要求代理开发任务包时，请指定任务输入、参数和预期输出。如果您不确定规格，请代理商先提出选项，然后一起完善要求。
- 启用反馈循环 — 当 AI 代理可以测试其解决方案并接收反馈时，他们可以更有效地进行迭代。与其指望在第一次尝试时得到有效的解决方案，不如让代理能够运行其解决方案并查看结果。当代理可以访问状态更新、日志和验证错误时，这种方法效果很好。例如，在开发任务包时，允许代理提交任务并查看日志。
- 期望迭代 — 即使有良好的上下文，代理也可能偏离正轨，或者做出与您的环境不匹配的假设。观察代理如何完成任务，并在此过程中提供指导。如果代理遇到困难，可以添加缺失的上下文，通过指向特定的日志文件来帮助发现错误，在发现要求时对其进行细化，并添加负面要求以明确说明代理应避免的内容。

代理上下文资源

以下资源可帮助 AI 代理理解 Deadline Cloud 的概念并生成准确的输出。

- Deadl ine C@@ loud 模型上下文协议 (MCP) 服务器 — 对于支持模型上下文协议的代理，截止日期云存储库包含 Dead line Cloud 客户端，其中包括用于与作业交互的 MCP 服务器。
- AWS文档 MCP 服务器 — 对于支持 MCP 的代理，请将AWS文档 MCP 服务器配置为允许代理直接访问AWS文档，包括 Deadline Cloud 用户指南和开发人员指南。
- Open Job Description 规范 — 上的公开职位描述规范 GitHub 定义了作业模板的架构。当代理需要了解作业模板的结构和语法时，请引用此存储库。

- deadline-cloud-samples— deadline-cloud-samples存储库包含示例作业包、conda 配方以及常见应用程序和用例的CloudFormation模板。
- aws-deadline GitHub 组织 — aws-dead line GitHub 组织包含许多第三方应用程序的参考插件，您可以将其用作其他集成的示例。

提示示例：编写工作包

以下示例提示演示如何使用 AI 代理创建任务捆绑包，用于训练 LoRa（低等级适应）适配器以生成 AI 图像。该提示说明了前面讨论的要点：它通过指向相关存储库提供了基础，定义了任务包输出的成功标准，并概述了迭代开发的反馈循环。

```
Write a pair of job bundles for Deadline Cloud that use the diffusers Python library to
 train a LoRA adapter on a set of images and then generate images from it.

Requirements:
- The training job takes a set of JPEG images as input, uses an image description,
 LoRA rank, learning rate, batch size, and number of training steps as parameters, and
 outputs a `.safetensors` file.
- The generation job takes the `.safetensors` file as input and the number of images
 to generate, then outputs JPEG images. The jobs use Stable Diffusion 1.5 as the base
 model.
- The jobs run `diffusers` as a Python script. Install the necessary packages using
 conda by setting the job parameters:
    - `CondaChannels`: `conda-forge`
    - `CondaPackages`: list of conda packages to install

For context, clone the following repositories to your workspace and review their
 documentation and code:
- OpenJobDescription specification: https://github.com/OpenJobDescription/openjd-
specifications/blob/mainline/wiki/2023-09-Template-Schemas.md
- Deadline Cloud sample job bundles: https://github.com/aws-deadline/deadline-cloud-
samples/tree/mainline/job_bundles
- diffusers library: https://github.com/huggingface/diffusers

Read through the provided context before you start. To develop a job bundle, iterate
 with the following steps until the submitted job succeeds. If a step fails, update the
 job bundle and restart the loop:
1. Create a job bundle.
2. Validate the job template syntax: `openjd check`
3. Submit the job to Deadline Cloud: `deadline bundle submit`
4. Wait for the job to complete: `deadline job wait`
5. View the job status and logs: `deadline job logs`
```

```
6. Download the job output: `deadline job download-output`

To verify the training and generation jobs work together, iterate with the following
 steps until the generation job produces images that resemble the dog in the training
 data:
1. Develop and submit a training job using the training images in `./exdog`
2. Wait for the job to succeed then download its output.
3. Develop and submit a generation job using the LoRA adapter from the training job.
4. Wait for the job to succeed then download its output.
5. Inspect the generated images. If they resemble the dog in the training data, you're
 done. Otherwise, review the job template, job parameters, and job logs to identify and
 fix the issue.
```

监控AWS截止日期云

监控是维护 Deadline Cloud（De AWS adline Cloud）和您的AWS解决方案的可靠性、可用性和性能的重要组成部分。从AWS解决方案的所有部分收集监控数据，以便在出现多点故障时可以更轻松地对其进行调试。在开始监控 Deadline Cloud 之前，您应该创建一个包含以下问题的答案的监控计划：

- 监控目的是什么？
- 您将监控哪些资源？
- 监控这些资源的频率如何？
- 您将使用哪些监控工具？
- 谁负责执行监控任务？
- 出现错误时应通知谁？

AWS和 Deadline Cloud 提供了可用于监控资源和应对潜在事件的工具。其中一些工具可以为您进行监控，有些工具需要手动干预。您应该尽可能自动执行监控任务。

- Amazon 会实时 CloudWatch监控您的AWS资源和您运行AWS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 CloudWatch 跟踪您的 Amazon EC2 实例的 CPU 使用率或其他指标，并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。

  截止日期云有三个 CloudWatch 指标。
- Amazon Lo CloudWatch gs 使您能够监控、存储和访问来自亚马逊 EC2 实例和其他来源的日志文件。 CloudTrail CloudWatch 日志可以监视日志文件中的信息，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 日志用户指南。
- Amazon EventBridge 可用于自动化您的AWS服务，并自动响应系统事件，例如应用程序可用性问题或资源更改。来自AWS服务的事件几乎是实时的。 EventBridge 您可以编写简单的规则来指示您关注的事件，并指示要在事件匹配规则时执行的自动化操作。有关更多信息，请参阅 Amazon EventBridge 用户指南。
- AWS CloudTrail捕获由您的账户或代表您的AWS账户进行的 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。您可以识别哪些用户和帐户拨打了电话AWS、发出呼叫的源 IP 地址以及呼叫发生的时间。有关更多信息，请参阅 AWS CloudTrail《用户指南》。

主题

- 使用记录 Deadline Cloud API 调用 AWS CloudTrail
- 使用监控 CloudWatch
- 使用管理截止日期云事件 Amazon EventBridge

使用记录 Deadline Cloud API 调用 AWS CloudTrail

Deadline Cloud 与AWS CloudTrail一项服务集成，该服务提供用户、角色或角色所执行操作的记录 AWS 服务。 CloudTrail 将所有 API 调用捕获 Deadline Cloud 为事件。捕获的调用包括来自 Deadline Cloud 控制台的调用和对 Deadline Cloud API 操作的代码调用。使用收集的信息 CloudTrail，您可以确定向哪个请求发出 Deadline Cloud、发出请求的 IP 地址、发出请求的时间以及其他详细信息。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根用户凭证还是用户凭证发出的。
- 请求是否代表 IAM Identity Center 用户发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

CloudTrail 在您创建账户 AWS 账户 时在您的账户中处于活动状态，并且您自动可以访问 CloudTrail 活动历史记录。 CloudTrail 事件历史记录提供了过去 90 天中记录的管理事件的可查看、可搜索、可下载且不可变的记录。 AWS 区域有关更多信息，请参阅《AWS CloudTrail 用户指南》中的 “使用 CloudTrail 事件历史记录”。查看活动历史记录不 CloudTrail收取任何费用。

要持续记录 AWS 账户 过去 90 天内的事件，请创建跟踪或 CloudTrailLake 事件数据存储。

CloudTrail 步道

跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。使用创建的所有跟踪 AWS 管理控制台 都是多区域的。您可以通过使用 AWS CLI创建单区域或多区域跟踪。建议创建多区域跟踪，因为您可以捕获账户 AWS 区域 中的所有活动。如果您创建单区域跟踪，则只能查看跟踪的 AWS 区域中记录的事件。有关跟踪的更多信息，请参阅《AWS CloudTrail 用户指南》中的为您的 AWS 账户创建跟踪和为组织创建跟踪。

通过创建跟踪，您可以免费将正在进行的管理事件的一份副本传送到您的 Amazon S3 存储桶，但会收取 Amazon S3 存储费用。 CloudTrail 有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。有关 Amazon S3 定价的信息，请参阅 Amazon S3 定价。

CloudTrail 湖泊事件数据存储

CloudTrail L@@ ak e 允许你对自己的事件运行基于 SQL 的查询。 CloudTrail Lake 将基于行的 JSON 格式的现有事件转换为 Apache ORC 格式。ORC 是一种针对快速检索数据进行优化的列式存储格式。事件将被聚合到事件数据存储中，它是基于您通过应用高级事件选择器选择的条件的不可变的事件集合。应用于事件数据存储的选择器用于控制哪些事件持续存在并可供您查询。有关 CloudTrail Lake 的更多信息，请参阅《AWS CloudTrail 用户指南》中的 “使用 AWS CloudTrail Lake”。

CloudTrail 湖泊事件数据存储和查询会产生费用。创建事件数据存储时，您可以选择要用于事件数据存储的定价选项。定价选项决定了摄取和存储事件的成本，以及事件数据存储的默认和最长保留期。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

Deadline Cloud 中的数据事件 CloudTrail

数据事件可提供对资源或在资源中所执行资源操作（例如，读取或写入 Amazon S3 对象）的相关信息。这些也称为数据面板操作。数据事件通常是高容量活动。默认情况下， CloudTrail 不记录数据事件。 CloudTrail 事件历史记录不记录数据事件。

记录数据事件将收取额外费用。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

您可以使用 CloudTrail 控制台或 CloudTrail API 操作记录 Deadline Cloud 资源类型的数据事件。AWS CLI有关如何记录数据事件的更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 AWS 管理控制台记录数据事件和使用 AWS Command Line Interface记录数据事件。

下表列出了您可以记录数据事件的 Deadline Cloud 资源类型。数据事件类型（控制台）列显示要从控制 CloudTrail 台上的数据事件类型列表中选择的值。res ources.type 值列显示该`resources.type`值，您将在使用或配置高级事件选择器时指定该值。 AWS CLI CloudTrail APIs“ APIs 记录到的数据 CloudTrail” 列显示了 CloudTrail 针对该资源类型记录的 API 调用。

数据事件类型（控制台）	resources.type 值	数据 APIs 已记录到 CloudTrail
截止日期舰队	`AWS::Deadline::Fleet`	• SearchWorkers
截止日期队列	`AWS::Deadline::Fleet`	• SearchJobs
截止日期工作人员	`AWS::Deadline::Worker`	• GetWorker • ListSessionsForWorker • UpdateWorkerSchedule

数据事件类型（控制台）	resources.type 值	数据 APIs 已记录到 CloudTrail
		• BatchGetJobEntity • ListWorkers
截止日期 Job	`AWS::Deadline::Job`	• ListStepConsumers • UpdateTask • ListJobs • GetStep • ListSteps • GetJob • GetTask • GetSession • ListSessions • CreateJob • ListSessionActions • ListTasks • CopyJobTemplate • UpdateSession • UpdateStep • UpdateJob • ListJobParameterDefinitions • GetSessionAction • ListStepDependencies • SearchTasks • SearchSteps

您可以将高级事件选择器配置为在 `eventName`、`readOnly` 和 `resources.ARN` 字段上进行筛选，从而仅记录那些对您很重要的事件。有关这些字段的更多信息，请参阅《AWS CloudTrail API 参考》中的 AdvancedFieldSelector。

Deadline Cloud 中的管理事件 CloudTrail

管理事件提供有关对中的资源执行的管理操作的信息 AWS 账户。这些也称为控制面板操作。默认情况下，CloudTrail 记录管理事件。

AWS Deadline Cloud 将以下 Deadline Cloud 控制平面操作记录 CloudTrail 为管理事件。

- associate-member-to-farm
- associate-member-to-fleet
- associate-member-to-job
- associate-member-to-queue
- assume-fleet-role-for-读
- assume-fleet-role-for-工人
- assume-queue-role-for-读
- assume-queue-role-for-用户
- assume-queue-role-for-工人
- 创建预算
- 创建农场
- create-fleet
- create-license-endpoint
- 创建限制
- 创建监视器
- 创建队列
- create-queue-environment
- create-queue-fleet-association
- create-queue-limit-association
- create-storage-profile
- 创建工作者
- 删除预算
- 删除农场
- delete-fleet
- delete-license-endpoint

- 删除限制
- delete-metered-product
- 删除监视器
- 删除队列
- delete-queue-environment
- delete-queue-fleet-association
- delete-queue-limit-association
- delete-storage-profile
- 删除工作人员
- disassociate-member-from-farm
- disassociate-member-from-fleet
- disassociate-member-from-job
- disassociate-member-from-queue
- get-application-version
- 获取预算
- get-farm
- get-feature-map
- get-fleet
- get-license-endpoint
- 获取限制
- get-Monitor
- 获取队列
- get-queue-environment
- get-queue-fleet-association
- get-queue-limit-association
- get-sessions-statistics-aggregation
- get-storage-profile
- get-storage-profile-for-队列
- list-available-metered-products
- 清单预算

- list-farm-members
- 列出农场
- list-fleet-members
- 列表舰队
- list-job-members
- list-license-endpoints
- 列表限制
- list-metered-products
- 列表监视器
- list-queue-environments
- list-queue-fleet-associations
- list-queue-limit-associations
- list-queue-members
- list-queues
- list-storage-profiles
- list-storage-profiles-for-队列
- list-tags-for-resource
- put-metered-product
- start-sessions-statistics-aggregation
- tag-resource
- untag-resource
- 更新预算
- 更新农场
- 更新舰队
- 更新限制
- 更新监视器
- 更新队列
- update-queue-environment
- update-queue-fleet-association
- update-queue-limit-association

- update-storage-profile
- 更新工作者

Deadline Cloud 事件示例

事件代表来自任何来源的单个请求，包括有关所请求的 API 操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此事件不会按任何特定顺序出现。

以下示例显示了一个演示该`CreateFarm`操作 CloudTrail 的事件。

```
{
    "eventVersion": "0",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE-PrincipalID:EXAMPLE-Session",
        "arn": "arn:aws:sts::111122223333:assumed-role/EXAMPLE-UserName/EXAMPLE-
Session",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE-accessKeyId",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE-PrincipalID",
                "arn": "arn:aws:iam::111122223333:role/EXAMPLE-UserName",
                "accountId": "111122223333",
                "userName": "EXAMPLE-UserName"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-03-08T23:25:49Z"
            }
        }
    },
    "eventTime": "2021-03-08T23:25:49Z",
    "eventSource": "deadline.amazonaws.com",
    "eventName": "CreateFarm",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "EXAMPLE-userAgent",
    "requestParameters": {
        "displayName": "example-farm",
```

```
        "kmsKeyArn": "arn:aws:kms:us-west-2:111122223333:key/111122223333",
        "X-Amz-Client-Token": "12abc12a-1234-1abc-123a-1a11bc1111a",
        "description": "example-description",
        "tags": {
            "purpose_1": "e2e"
            "purpose_2": "tag_test"
        }
    },
    "responseElements": {
        "farmId": "EXAMPLE-farmID"
    },
    "requestID": "EXAMPLE-requestID",
    "eventID": "EXAMPLE-eventID",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333"
    "eventCategory": "Management",
}
```

JSON 示例显示了 AWS 区域、IP 地址和其他 `requestParameters` “”，例如 `displayName` “kmsKeyArn” 和 “”，可以帮助您识别事件。

有关 CloudTrail 录音内容的信息，请参阅《AWS CloudTrail 用户指南》中的CloudTrail录制内容。

使用监控 CloudWatch

Amazon CloudWatch (CloudWatch) 收集原始数据并将其处理成可读的近乎实时的指标。你可以打开 CloudWatch 控制台，查看和筛选 De https://console.aws.amazon.com/cloudwatch/adline Cloud 指标。

这些统计数据会保存 15 个月，因此您可以访问历史信息，从而更好地了解 Web 应用程序或服务的性能。还可以设置特定阈值监视警报，在达到对应阈值时发送通知或采取行动。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

Deadline Cloud 有两种日志：任务日志和工作人员日志。任务日志是指将执行日志作为脚本运行或 DCC 运行时。任务日志可能会显示诸如资源加载、图块渲染或未找到纹理之类的事件。

工作器日志显示工作器代理进程。这些可能包括诸如工作器代理何时启动、注册自身、报告进度、加载配置或完成任务之类的事情。

这些日志的命名空间是/aws/deadline/*。

对于 Deadline Cloud，工作人员将这些日志上传到 CloudWatch 日志。默认情况下，日志永不过期。如果任务输出了大量数据，则可能会产生额外的成本。有关更多信息，请参阅 Amazon CloudWatch 定价。

您可以调整每个日志组的保留策略。较短的保留期会删除旧日志，并有助于降低存储成本。要保留日志，您可以在删除日志之前将其存档到 Amazon 简单存储服务。有关更多信息，请参阅亚马逊 CloudWatch 用户指南中的使用控制台将日志数据导出到 Amazon S3。

> **Note**
>
> CloudWatch 日志读取受限制AWS。如果您计划招募许多艺术家，我们建议您联系AWS客户支持并申请增加GetLogEvents配额 CloudWatch。此外，我们建议您在不进行调试时关闭日志跟踪门户。

有关更多信息，请参阅 Amazon CloudWatch 用户指南中的CloudWatch 日志配额。

CloudWatch 指标

Deadline Cloud 向亚马逊发送指标 CloudWatch。您可以使用AWS 管理控制台AWS CLI、或 API 列出 Deadline Cloud 发送到的指标 CloudWatch。默认情况下，每个数据点涵盖活动开始时间之后的 1 分钟。有关如何使用AWS 管理控制台或查看可用指标的信息AWS CLI，请参阅 Amazon CloudWatch 用户指南中的查看可用指标。

客户管理的车队指标

AWS/DeadlineCloud命名空间包含客户管理的车队的以下指标：

指标	说明	单位
RecommendedFleetSize	Deadline Cloud 推荐你用来处理作业的工作人员数量。您可以使用此指标来扩大或缩减车队的员工人数。	计数
UnhealthyWorkerCount	分配给处理不健康作业的员工人数。	计数

您可以使用以下维度来完善客户管理的车队指标：

维度	说明
FarmId	此维度筛选您向指定服务器场请求的数据。
FleetId	此维度筛选您向指定工作人员队列请求的数据。

许可指标

AWS/DeadlineCloud命名空间包含以下许可指标：

指标	说明	单位
LicensesInUse	正在使用的许可证会话数。	计数

您可以使用以下维度来完善许可指标：

维度	说明
FleetId	使用此维度将数据筛选到指定的服务托管队列。对于客户管理的车队，请改用 LicenseEndpointId 维度。
LicenseEndpointId	使用此维度将数据筛选到指定的许可证端点。
产品	使用此维度将数据筛选到指定的计量产品。

资源限制指标

AWS/DeadlineCloud命名空间包含以下资源限制指标：

指标	说明	单位
CurrentCount	按此限制建模的正在使用的资源数量。	计数

指标	说明	单位
MaxCount	按此限制建模的最大资源数。如果您使用 API 将该maxCount值设置为 -1，则 Deadline Cloud 不会发出该MaxCount指标。	计数

您可以使用以下维度来细化并发限制指标：

维度	说明
FarmId	此维度筛选您向指定服务器场请求的数据。
LimitId	此维度会将您请求的数据筛选到指定限制。

建议的警报

借 CloudWatch助，您可以创建警报，以便在突破阈值时监控指标并向您发送通知或执行其他操作。有关配置 CloudWatch 警报的更多信息，请参阅 Amazon CloudWatch 用户指南。

我们建议您为以下 Deadline Cloud 指标设置警报：

LicensesInUse

尺寸： FleetId， LicenseEndpointId

警报描述：此警报可检测服务托管车队或许可证端点的有效许可证会话何时接近您的账户配额。如果发生这种情况，您可以提高许可证会话的账户配额。使用 Service Quotas 查看您当前的配额并请求增加配额。要了解更多信息，请参阅 S ervice Quotas 用户指南。

意图：通过在许可证达到配额限制之前监控使用情况，防止许可证签出失败。

统计数据：Maximum

建议阈值：许可证会话配额的 90%

阈值理由：将阈值设置为配额的百分比，这样您就可以在限额达到限制之前采取行动。

时间段：1 分钟

要发出警报的数据点数：1

评估期：1

比较运算符：GREATER_THAN_THRESHOLD

其他资源

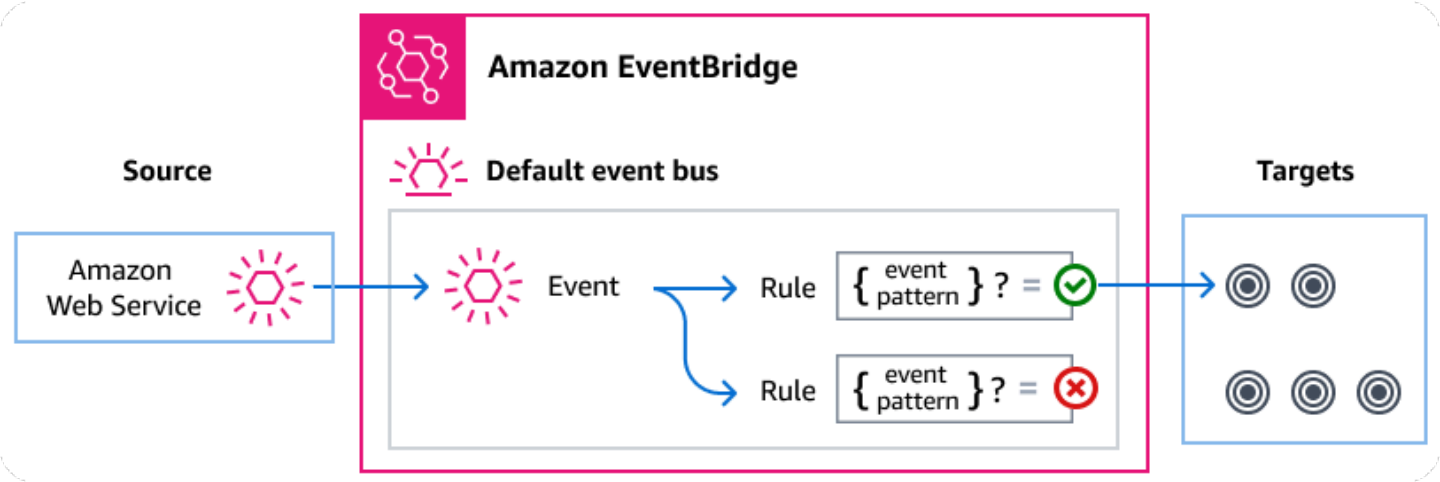
- 亚马逊 CloudWatch 用户指南
- Service Quotas User Guide

使用管理截止日期云事件 Amazon EventBridge

Amazon EventBridge 是一项无服务器服务，它使用事件将应用程序组件连接在一起，使您可以更轻松地构建可扩展的事件驱动应用程序。事件驱动型架构是一种构建松耦合软件系统的风格，这些系统通过发出和响应事件 来协同工作。事件代表资源或环境中的变化。

下面将介绍操作方式：

与许多 AWS 服务一样，Deadline Cloud 生成事件并将其发送到 EventBridge 默认事件总线。（默认事件总线会在每个 AWS 账户中自动配置。）事件总线是接收事件并将其传送到零个或多个目的地或目标的路由器。为事件总线指定的规则会在事件到达时进行评估。每条规则都会检查事件是否与规则的事件模式相匹配。如果事件确实匹配，事件总线会将事件发送到指定的目标。



主题

- 截止日期云活动

- 使用 EventBridge 规则交付截止日期云事件
- 截止日期云活动详情参考

截止日期云活动

Deadline Cloud 会自动将以下 EventBridge 事件发送到默认事件总线。与规则的事件模式相匹配的事件会尽力交付给指定的目标。事件可能不按顺序传送。

有关更多信息，请参阅《Amazon EventBridge 用户指南》中的 EventBridge 事件。

事件详细信息类型	说明
已达到预算阈值	当队列达到其分配预算的一定百分比时发送。
Job 生命周期状态变更	当任务的生命周期状态发生变化时发送。
Job 运行状态更改	当作业中任务的总体状态发生变化时发送。
步骤生命周期状态更改	当任务中某个步骤的生命周期状态发生变化时发送。
Step Run 状态更改	当步骤中任务的总体状态发生变化时发送。
任务运行状态更改	任务状态发生变化时发送。

使用 EventBridge 规则交付截止日期云事件

要让 EventBridge 默认事件总线将 Deadline Cloud 事件发送到目标，您必须创建规则。每条规则都包含一个事件模式，该模式与事件总线上接收到的每个事件进行 EventBridge 匹配。如果事件数据与指定的事件模式匹配，则将该事件 EventBridge 传送到规则的目标。

有关创建事件总线规则的全面说明，请参阅《EventBridge 用户指南》中的创建对事件作出反应的规则。

创建与 Deadline Cloud 事件匹配的事件模式

每个事件模式是一个 JSON 对象，其中包含：

- 标识发送事件的服务的 `source` 属性。对于 Deadline Cloud 事件，来源是`aws.deadline`。
- （可选）：包含要匹配的事件类型数组的 `detail-type` 属性。

- （可选）：包含要匹配的其他事件数据的 detail 属性。

例如，以下事件模式与 Deadline Cloud 指定farmId的所有舰队规模建议变更事件相匹配：

```
{
  "source": ["aws.deadline"],
  "detail-type": ["Fleet Size Recommendation Change"],
  "detail": {
     "farmId": "farm-12345678900000000000000000000000"
  }
}
```

有关写入事件模式的更多信息，请参阅《EventBridge 用户指南》中的事件模式。

截止日期云活动详情参考

来自 AWS 服务的所有事件都有一组公共字段，其中包含有关事件的元数据，例如作为事件来源的 AWS 服务、事件的生成时间、事件发生的账户和区域等。有关这些常规字段的定义，请参阅《Amazon EventBridge 用户指南》中的事件结构参考。

此外，每个事件都有一个 detail 字段，其中包该特定事件专有的数据。以下参考文献定义了各种 Deadline Cloud 事件的详细信息字段。

在使用 EventBridge 选择和管理 Deadline Cloud 事件时，请记住以下几点：

- 来自 Deadline Cloud 的所有事件的source字段设置为aws.deadline。
- detail-type 字段指定事件类型。

  例如 Fleet Size Recommendation Change。
- detail 字段包含该特定事件专有的数据。

有关构建事件模式以使规则与 Deadline Cloud 事件相匹配的信息，请参阅Amazon EventBridge 用户指南中的事件模式。

有关事件及其 EventBridge 处理方式的更多信息，请参阅《Amazon EventBridge 用户指南》中的Amazon EventBridge 事件。

主题

- 已达到预算阈值事件

- 舰队规模建议变更事件
- Job 生命周期状态更改事件
- Job 运行状态更改事件
- 步骤生命周期状态更改事件
- Step Run 状态更改事件
- 任务运行状态更改事件

已达到预算阈值事件

您可以使用 “已达到预算阈值” 事件来监控已使用的预算百分比。当使用的百分比超过以下阈值时，Deadline Cloud 会发送事件：

- 10、20、30、40、50、60、70、75、80、85、90、95、96、97、98、99、100

随着预算接近上限，Deadline Cloud 发送已达到预算阈值事件的频率会增加。此频率使您能够在预算接近上限时密切监视预算，并采取措施防止超支。您也可以设置自己的预算阈值。当使用量超过您的自定义阈值时，Deadline Cloud 会发送一个事件。

如果您更改预算金额，则 Deadline Cloud 下次发送 “已达到预算阈值” 事件时，它将基于已使用的预算的当前百分比。例如，如果您在已达到限额的 100 美元预算的基础上再加上 50 美元，则下一个达到预算阈值事件将表明预算为 75%。

以下是 `Budget Threshold Reached` 事件的详细信息字段。

之所以包含`source`和`detail-type`字段，是因为它们包含 Deadline Cloud 事件的特定值。有关所有事件中包含的其他元数据字段的定义，请参阅Amazon EventBridge 用户指南中的事件结构参考。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Budget Threshold Reached",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "budgetId": "budget-12345678900000000000000000000000",
```

```
        "thresholdInPercent": 0
    }
}
```

`detail-type`

标识事件的类型。

对于这一事件，此值为 `Budget Threshold Reached`。

`source`

标识生成事件的服务。对于 Deadline Cloud 事件，此值为`aws.deadline`。

`detail`

包含关于事件信息的 JSON 对象。生成事件的服务决定该字段的内容。

对于此事件，此数据包括：

`farmId`

包含任务的服务器场的标识符。

`budgetId`

已达到阈值的预算的标识符。

`thresholdInPercent`

已使用的预算百分比。

舰队规模建议变更事件

当您将队列配置为使用基于事件的自动缩放时，Deadline Cloud 会发送可用于管理队列的事件。这些事件中的每一个都包含有关舰队当前规模和请求规模的信息。有关使用 EventBridge 事件和示例 Lambda 函数来处理事件的示例，请参阅使用 De adline Cloud 规模推荐功能自动扩展您的 Amazon EC2 队列。

发生以下情况时，将发送舰队规模建议更改事件：

- 当建议的舰队规模发生变化 oldFleetSize 并且与之不同时 newFleetSize。
- 当服务检测到实际舰队规模与建议的舰队规模不匹配时。您可以从 GetFleet 操作响应中的 workerCount 中获取实际的舰队规模。当活跃的 Amazon EC2 实例未能注册为 Deadline Cloud 工作程序时，可能会发生这种情况。

以下是 Fleet Size Recommendation Change 事件的详细信息字段。

之所以包含source和detail-type字段，是因为它们包含 Deadline Cloud 事件的特定值。有关所有事件中包含的其他元数据字段的定义，请参阅Amazon EventBridge 用户指南中的事件结构参考。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Fleet Size Recommendation Change",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "fleetId": "fleet-12345678900000000000000000000000",
        "oldFleetSize": 1,
        "newFleetSize": 5,
    }
}
```

detail-type

标识事件的类型。

对于这一事件，此值为 Fleet Size Recommendation Change。

source

标识生成事件的服务。对于 Deadline Cloud 事件，此值为aws.deadline。

detail

包含关于事件信息的 JSON 对象。生成事件的服务决定该字段的内容。

对于此事件，此数据包括：

farmId

包含任务的服务器场的标识符。

fleetId

需要更改规模的舰队的标识符。

`oldFleetSize`

舰队的当前规模。

`newFleetSize`

舰队的推荐新规模。

Job 生命周期状态更改事件

当您创建或更新任务时，Deadline Cloud 会将生命周期状态设置为显示用户最近启动的操作的状态。

对于任何生命周期状态更改（包括作业的创建时间），都会发送作业生命周期状态更改事件。

以下是 `Job Lifecycle Status Change` 事件的详细信息字段。

之所以包含`source`和`detail-type`字段，是因为它们包含 Deadline Cloud 事件的特定值。有关所有事件中包含的其他元数据字段的定义，请参阅Amazon EventBridge 用户指南中的事件结构参考。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Job Lifecycle Status Change",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "previousLifecycleStatus": "UPDATE_IN_PROGRESS",
        "lifecycleStatus": "UPDATE_SUCCEEDED"
    }
}
```

`detail-type`

标识事件的类型。

对于这一事件，此值为 `Job Lifecycle Status Change`。

`source`

标识生成事件的服务。对于 Deadline Cloud 事件，此值为`aws.deadline`。

`detail`

包含关于事件信息的 JSON 对象。生成事件的服务决定该字段的内容。

对于此事件，此数据包括：

`farmId`

包含任务的服务器场的标识符。

`queueId`

包含任务的队列的标识符。

`jobId`

任务的标识符。

`previousLifecycleStatus`

任务即将离开的生命周期状态。首次提交工作时，此字段不包括在内。

`lifecycleStatus`

任务正在进入的生命周期状态。

Job 运行状态更改事件

一项作业由许多任务组成。每项任务都有一个状态。将所有任务的状态合并在一起，得出作业的总体状态。有关更多信息，请参阅 Deadline Cloud 用户指南中的 Deadlin AWS e Cloud 中的作业状态。

在以下情况下会发送作业运行状态更改事件：

- 组合`taskRunStatus`字段会发生变化。
- 除非作业处于 READY 状态，否则该作业将重新排队。

在以下情况下，不会发送作业运行状态更改事件：

- 作业是首次创建的。要监控作业的创建，请监控作业生命周期状态更改事件的更改。

- 作业的taskRunStatusCounts字段会发生变化，但合并的作业任务运行状态不会改变。

以下是 Job Run Status Change 事件的详细信息字段。

之所以包含source和detail-type字段，是因为它们包含 Deadline Cloud 事件的特定值。有关所有事件中包含的其他元数据字段的定义，请参阅Amazon EventBridge 用户指南中的事件结构参考。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Job Run Status Change",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "previousTaskRunStatus": "RUNNING",
        "taskRunStatus": "SUCCEEDED",
        "taskRunStatusCounts": {
            "PENDING": 0,
            "READY": 0,
            "RUNNING": 0,
            "ASSIGNED": 0,
            "STARTING": 0,
            "SCHEDULED": 0,
            "INTERRUPTING": 0,
            "SUSPENDED": 0,
            "CANCELED": 0,
            "FAILED": 0,
            "SUCCEEDED": 20,
            "NOT_COMPATIBLE": 0
       }
    }
}
```

`detail-type`

标识事件的类型。

对于这一事件，此值为 Job Run Status Change。

source

标识生成事件的服务。对于 Deadline Cloud 事件，此值为aws.deadline。

detail

包含关于事件信息的 JSON 对象。生成事件的服务决定该字段的内容。

对于此事件，此数据包括：

farmId

包含任务的服务器场的标识符。

queueId

包含任务的队列的标识符。

jobId

任务的标识符。

previousTaskRunStatus

任务运行状态为任务即将离开。

taskRunStatus

任务正在进入的任务运行状态。

taskRunStatusCounts

每种状态下作业的任务数。

步骤生命周期状态更改事件

当您创建或更新事件时，Deadline Cloud 会设置作业的生命周期状态，以描述用户最近发起的操作的状态。

在以下情况下，会发送步骤生命周期状态更改事件：

- 步骤更新开始（UPDATE_IN_PROGRESS）。
- 步骤更新成功完成（UPDATE_SUCCELEDED）。
- 步骤更新失败（UPDATE_FAILED）。

首次创建步骤时不会发送事件。要监控步骤的创建，请监控 Job 生命周期状态更改事件的更改。

以下是 Step Lifecycle Status Change 事件的详细信息字段。

之所以包含source和detail-type字段，是因为它们包含 Deadline Cloud 事件的特定值。有关所有事件中包含的其他元数据字段的定义，请参阅Amazon EventBridge 用户指南中的事件结构参考。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Step Lifecycle Status Change",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "stepId": "step-12345678900000000000000000000000",
        "previousLifecycleStatus": "UPDATE_IN_PROGRESS",
        "lifecycleStatus": "UPDATE_SUCCEEDED"
    }
}
```

`detail-type`

标识事件的类型。

对于这一事件，此值为 Step Lifecycle Status Change。

`source`

标识生成事件的服务。对于 Deadline Cloud 事件，此值为aws.deadline。

`detail`

包含关于事件信息的 JSON 对象。生成事件的服务决定该字段的内容。

对于此事件，此数据包括：

`farmId`

包含任务的服务器场的标识符。

`queueId`

包含任务的队列的标识符。

`jobId`

任务的标识符。

`stepId`

当前作业步骤的标识符。

`previousLifecycleStatus`

步骤即将离开的生命周期状态。

`lifecycleStatus`

步骤正在进入的生命周期状态。

Step Run 状态更改事件

作业中的每个步骤都由许多任务组成。每项任务都有一个状态。将任务状态组合在一起，以提供步骤和作业的总体状态。

在以下情况下会发送步骤运行状态更改事件：

- 合并后的`taskRunStatus`变化。
- 除非该步骤处于 READY 状态，否则该步骤将重新排队。

在以下情况下不会发送事件：

- 步骤是首先创建的。要监控步骤的创建，请监控 Job 生命周期状态更改事件的更改。
- 步骤会`taskRunStatusCounts`发生变化，但组合步骤任务的运行状态不会改变。

以下是 `Step Run Status Change` 事件的详细信息字段。

之所以包含`source`和`detail-type`字段，是因为它们包含 Deadline Cloud 事件的特定值。有关所有事件中包含的其他元数据字段的定义，请参阅Amazon EventBridge 用户指南中的事件结构参考。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
```

```
    "detail-type": "Step Run Status Change",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "stepId": "step-12345678900000000000000000000000",
        "previousTaskRunStatus": "RUNNING",
        "taskRunStatus": "SUCCEEDED",
        "taskRunStatusCounts": {
            "PENDING": 0,
            "READY": 0,
            "RUNNING": 0,
            "ASSIGNED": 0,
            "STARTING": 0,
            "SCHEDULED": 0,
            "INTERRUPTING": 0,
            "SUSPENDED": 0,
            "CANCELED": 0,
            "FAILED": 0,
            "SUCCEEDED": 20,
            "NOT_COMPATIBLE": 0
       }
    }
}
```

`detail-type`

标识事件的类型。

对于这一事件，此值为 `Step Run Status Change`。

`source`

标识生成事件的服务。对于 Deadline Cloud 事件，此值为`aws.deadline`。

`detail`

包含关于事件信息的 JSON 对象。生成事件的服务决定该字段的内容。

对于此事件，此数据包括：

`farmId`

包含任务的服务器场的标识符。

`queueId`

包含任务的队列的标识符。

`jobId`

任务的标识符。

`stepId`

当前作业步骤的标识符。

`previousTaskRunStatus`

步骤即将离开的运行状态。

`taskRunStatus`

该步骤正在进入的运行状态。

`taskRunStatusCounts`

每种状态下步骤的任务数。

任务运行状态更改事件

该`runStatus`字段将在任务运行时更新。在以下情况下会发送事件：

- 任务的运行状态会发生变化。
- 除非任务处于 READY 状态，否则该任务将重新排队。

在以下情况下不会发送事件：

- 任务首先创建。要监控任务的创建，请监控 Job 生命周期状态更改事件是否有更改。

以下是 `Task Run Status Change` 事件的详细信息字段。

之所以包含`source`和`detail-type`字段，是因为它们包含 Deadline Cloud 事件的特定值。有关所有事件中包含的其他元数据字段的定义，请参阅Amazon EventBridge 用户指南中的事件结构参考。

```
{
```

```
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Task Run Status Change",
    "source": "aws.aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "stepId": "step-12345678900000000000000000000000",
        "taskId": "task-12345678900000000000000000000000-0",
        "previousRunStatus": "RUNNING",
        "runStatus": "SUCCEEDED"
    }
}
```

`detail-type`

标识事件的类型。

对于这一事件，此值为 `Fleet Size Recommendation Change`。

`source`

标识生成事件的服务。对于 Deadline Cloud 事件，此值为`aws.deadline`。

`detail`

包含关于事件信息的 JSON 对象。生成事件的服务决定该字段的内容。

对于此事件，此数据包括：

`farmId`

包含任务的服务器场的标识符。

`queueId`

包含任务的队列的标识符。

`jobId`

任务的标识符。

stepId

当前作业步骤的标识符。

taskId

正在运行的任务的标识符。

previousRunStatus

任务即将离开的运行状态。

runStatus

任务正在进入的运行状态。

使用查询会话统计信息聚合数据 AWS CLI

要跟踪成本、分析资源使用情况或确定哪些用户消耗的资源最多，您可以使用 AWS Command Line Interface (AWS CLI) 查询 Deadline Cloud（De AWS adline Cloud）场的聚合会话统计信息。会话统计 API 提供有关成本、运行时间和使用情况的数据，您可以按队列、队列、实例类型或用户等各种维度对这些数据进行分组。

查询会话统计信息是一个异步过程。首先，启动聚合请求，然后使用聚合 ID 检索结果。

启动聚合请求

要启动聚合请求，请运行`start-sessions-statistics-aggregation`命令。以下示例按用户 ID 对特定队列的统计数据进行分组。用您的信息替换*`placeholder`*文本。

```
aws deadline start-sessions-statistics-aggregation \
    --farm-id farm-id \
    --resource-ids '{"queueIds":["queue-id"]}' \
    --start-time 2025-11-24T10:00:00Z \
    --end-time 2025-11-25T18:00:00Z \
    --group-by '["USER_ID"]' \
    --period HOURLY \
    --statistics '["SUM"]' \
    --timezone UTC-08:00 \
    --region region-name
```

您可以按其他维度对统计数据进行分组`QUEUE_ID`，例如`FLEET_ID`、`JOB_ID`、`INSTANCE_TYPE`、或`LICENSE_PRODUCT`。有关所有可用参数的更多信息，请参阅《 AWS CLI 命令参考》start-sessions-statistics-aggregation中的。

响应包含聚合 ID：

```
{
    "aggregationId": "92b35143f2d04641979bc9b777232f38"
}
```

检索结果

使用聚合 ID 运行`get-sessions-statistics-aggregation`命令以检索结果。用您的信息替换*`placeholder`*文本。

```
aws deadline get-sessions-statistics-aggregation \
    --farm-id farm-id \
    --aggregation-id aggregation-id \
    --region region-name
```

以下示例显示了按用户 ID 对统计数据进行分组时的响应。该userId字段包含一个 UUID，您必须将其映射到用户名才能识别用户：

```
{
    "statistics": [
        {
            "userId": "f9c1f3f0-1031-70dc-4d25-30d7225b04a0",
            "count": 1,
            "costInUsd": {
                "sum": 0.0
            },
            "runtimeInSeconds": {
                "sum": 53.773
            },
            "aggregationStartTime": "2025-11-24T22:00:00Z",
            "aggregationEndTime": "2025-11-24T23:00:00Z"
        }
    ],
    "status": "COMPLETED"
}
```

要查找与关联的用户名userId，请参阅the section called “使用 userID 检索用户元数据”。

有关 API 的更多信息，请参阅 De adline Cloud API 参考。

主题

- the section called “使用 userID 检索用户元数据”

在身份存储中使用 userID 检索用户元数据和属性

> **Note**
>
> 此过程也适用于 SearchJobsAPI 返回的字段，该createdBy字段使用相同的用户 ID 格式。

会话统计信息中的userId字段包含以下值之一：

- AWS Identity and Access Management (IAM) 角色 ARN，例如：。arn:aws:sts::123456789012:assumed-role/Admin/user-Isengard
- IAM 身份中心用户 ID (UUID)，例如：f9c1f3f0-1031-70dc-4d25-30d7225b04a0。

对于 IAM 角色 ARNs，用户名在 ARN 本身中可见。对于 IAM 身份中心用户 IDs，您可以使用 IAM 身份中心身份存储 API 查找用户名。

要识别与 IAM 身份中心用户 ID 关联的用户名，请按以下步骤操作。在开始之前，请从 IAM 身份中心设置中获取身份存储 ID。有关更多信息，请参阅 the section called “查找您的身份存储 ID”。

映射用户 ID

1. 运行以下命令，*IdentityStoreId*替换为您的身份存储 ID 和*userUUID*userId来自会话统计信息响应的：

```
aws identitystore describe-user \
    --identity-store-id IdentityStoreId \
    --user-id userUUID
```

2. 查看响应，其中包含用户名：

```
{
    "UserName": "jdoe",
    "UserId": "f9c1f3f0-1031-70dc-4d25-30d7225b04a0",
    "Name": {
        "FamilyName": "Doe",
        "GivenName": "Jane"
    },
    "DisplayName": "Jane Doe",
    "Emails": [{
        "Value": "jdoe@example.com",
        "Type": "work",
        "Primary": true
    }],
    "IdentityStoreId": "d-xxxxxxxxxx"
}
```

查找您的身份存储 ID

要将用户映射 IDs 到用户名，您需要身份存储 ID。您可以使用 IAM 身份中心控制台或. 找到身份存储 ID AWS CLI。

控制台

要使用控制台查找您的身份存储 ID，请按以下步骤操作。

1. 登录 AWS 管理控制台并打开 IAM 身份中心控制台。
2. 在导航窗格中，选择设置。
3. 复制 I AM 身份中心身份存储 ID 值。格式为 d-xxxxxxxxxx。

AWS CLI

运行以下命令，*region-name*替换为配置您的 IAM 身份中心实例的区域：

```
aws sso-admin list-instances --region region-name
```

响应包括IdentityStoreId：

```
{
    "Instances": [
        {
            "CreatedDate": "2025-11-19T15:45:55.160000-08:00",
            "IdentityStoreId": "d-xxxxxxxxxx",
            "InstanceArn": "arn:aws:sso:::instance/ssoins-xxxxxxxxxxxxxxxx",
            "OwnerAccountId": "123456789012",
            "Status": "ACTIVE"
        }
    ]
}
```

验证用户映射

将用户 ID 映射到用户名后，您可以在 IAM Identity Center 控制台中验证用户 ID 是否与预期用户匹配。要验证用户映射，请按以下步骤操作。

1. 登录 AWS 管理控制台并打开 IAM 身份中心控制台。

2. 在导航窗格中，选择 Users（用户）。
3. 从 AWS CLI 响应中选择用户名。
4. 在 “一般信息” 部分，验证用户 ID 是否与您的会话统计数据`userId`中的用户名相匹配。

其他资源

- IAM Identity Center 用户指南
- IAM 身份中心身份存储 API 参考

中的安全性 Deadline Cloud

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在云 AWS 服务 中运行的基础架构 AWS Cloud。 AWS 还为您提供可以安全使用的服务。 Third-party 作为AWS 合规计划合规计划合规计划合的一部分，审计师定期测试和验证我们安全的有效性。要了解适用于的合规计划 AWS Deadline Cloud，请参阅 “按合规计划划分AWS 服务 的范围” 中的 “按合规计划AWS 服务”。
- 云端安全 — 您的责任由您 AWS 服务 使用的内容决定。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。

本文档可帮助您了解在使用时如何应用分担责任模型 Deadline Cloud。以下主题向您介绍如何进行配置 Deadline Cloud 以满足您的安全和合规性目标。您还将学习如何使用其他方法 AWS 服务 来帮助您监控和保护您的 Deadline Cloud 资源。

主题

中的数据保护 Deadline Cloud

AWS 责任共担模式适用于保护 AWS Deadline Cloud中的数据。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础结构上的内容的控制。您还负责您

所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题解答AWS条款。 有关欧洲数据保护的信息，请参阅通用数据保护条例（GDPR）中心。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》https://aws.amazon.com/compliance/fips/。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您使用控制台、API Deadline Cloud 或 SDK 或以其他 AWS 服务 方式使用控制台 AWS CLI、API 或 AWS SDK 的情况。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供 URL，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

在 Deadline Cloud 作业模板的姓名字段中输入的数据也可能包含在账单或诊断日志中，不应包含机密或敏感信息。

主题

- 静态加密
- 传输中加密
- 密钥管理
- Inter-network 交通隐私
- 选择退出

静态加密

AWS Deadline Cloud 使用存储在 AWS Key Management Service (AWS KMS) 中的加密密钥对静态数据进行加密，从而保护敏感数据。所有可用 AWS 区域 的地方 Deadline Cloud 都提供静态加密。

加密数据意味着如果没有有效的密钥，用户或应用程序就无法读取保存在磁盘上的敏感数据。只有拥有有效托管密钥的一方才能解密数据。

Deadline Cloud 当服务托管队列工作程序实例终止时，会删除 Amazon 弹性块存储卷。

有关如何 Deadline Cloud 使用 AWS KMS 静态加密数据的信息，请参阅密钥管理。

传输中加密

对于传输中的数据， AWS Deadline Cloud 使用传输层安全 (TLS) 1.2 或 1.3 来加密在服务和工作程序之间发送的数据。我们要求使用 TLS 1.2，建议使用 TLS 1.3。此外，如果您使用虚拟私有云 (VPC)，则可以使用 AWS PrivateLink 在您的 VPC 和之间建立私有连接 Deadline Cloud。

密钥管理

创建新服务器场时，您可以选择以下密钥之一来加密服务器场数据：

- AWS 拥有的 KMS 密钥-如果您在创建服务器场时未指定密钥，则为默认加密类型。KMS 密钥归所有者 AWS Deadline Cloud。您无法查看、管理或使用 AWS 自有密钥。但是，您无需采取任何措施来保护加密数据的密钥。有关更多信息，请参阅AWS Key Management Service 开发者指南中的AWS 自有密钥。
- 客户托管的 KMS 密钥-您在创建服务器场时指定客户托管密钥。服务器场中的所有内容均使用 KMS 密钥进行加密。密钥存储在您的账户中，由您创建、拥有和管理，并 AWS KMS 收取费用。您对 KMS 密钥拥有完全控制权。您可以执行以下任务：
  - 制定和维护关键政策
  - 建立和维护 IAM 策略和授权
  - 启用和禁用密钥策略
  - 添加 标签
  - 创建密钥别名

  您无法手动轮换用于 Deadline Cloud 服务器场的客户拥有的密钥。支持密钥的自动轮换。

  有关更多信息，请参阅《AWS Key Management Service 开发者指南》中的客户拥有的密钥。

要创建客户托管密钥，请按照《AWS Key Management Service 开发人员指南》中创建对称客户托管密钥的步骤进行操作。

操作方法 Deadline Cloud uses AWS KMS 补助金

Deadline Cloud 需要获得授权才能使用您的客户托管密钥。当您创建使用客户托管密钥加密的场时，Deadline Cloud 会向发送CreateGrant请求 AWS KMS 以获取您指定的 KMS 密钥的访问权限，从而代表您创建授权。

Deadline Cloud 使用多个授权。每项拨款都由需要加密或解密您的数据的不同部分使用。 Deadline Cloud Deadline Cloud 还使用授权来允许访问用于代表您存储数据的其他 AWS 服务，例如亚马逊简单存储服务、Amazon Elastic Block Store 或 OpenSearch。

Deadline Cloud 允许管理服务管理队列中的计算机的授权包括 Deadline Cloud 账号和角色，GranteePrincipal而不是服务委托人。虽然不常见，但这是使用为服务器场指定的客户托管 KMS 密钥为服务托管队伍中的工作人员加密 Amazon EBS 卷所必需的。

客户自主管理型密钥策略

密钥政策控制对客户托管密钥的访问。每个密钥必须只有一个密钥策略，其中包含用于确定谁可以使用密钥以及如何使用密钥的声明。在创建客户托管密钥时，您可以指定密钥策略。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的管理对客户托管密钥的访问。

适用的最低 IAM 政策 CreateFarm

要使用您的客户托管密钥通过控制台或 CreateFarm API 操作创建农场，必须允许以下 AWS KMS API 操作：

- kms:CreateGrant：向客户托管密钥添加授权。授予对指定 AWS KMS 密钥的控制台访问权限。有关更多信息，请参阅AWS Key Management Service 开发者指南中的使用授权。
- kms:Decrypt— Deadline Cloud 允许解密服务器场中的数据。
- kms:DescribeKey— 提供客户管理的密钥详细信息 Deadline Cloud 以允许验证密钥。
- kms:GenerateDataKey— Deadline Cloud 允许使用唯一的数据密钥对数据进行加密。

以下策略声明授予CreateFarm操作所需的权限。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DeadlineCreateGrants",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:CreateGrant",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234567890abcdef0",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "deadline.us-west-2.amazonaws.com"
                }
            }
        }
    ]
}
```

只读操作的最低 IAM 政策

使用您的客户托管密钥进行只读 Deadline Cloud 操作，例如获取有关农场、队列和队列的信息。必须允许以下 AWS KMS API 操作：

- kms:Decrypt— Deadline Cloud 允许解密服务器场中的数据。
- kms:DescribeKey— 提供客户管理的密钥详细信息 Deadline Cloud 以允许验证密钥。

以下策略声明授予只读操作所需的权限。

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "DeadlineReadOnly",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "deadline.us-west-2.amazonaws.com"
                }
            }
        }
    ]
}
```

读写操作的最低 IAM 策略

使用您的客户托管密钥进行读写 Deadline Cloud 操作，例如创建和更新服务器场、队列和队列。必须允许以下 AWS KMS API 操作：

- kms:Decrypt— Deadline Cloud 允许解密服务器场中的数据。
- kms:DescribeKey— 提供客户管理的密钥详细信息 Deadline Cloud 以允许验证密钥。
- kms:GenerateDataKey— Deadline Cloud 允许使用唯一的数据密钥对数据进行加密。

以下策略声明授予CreateFarm操作所需的权限。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DeadlineReadWrite",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
```

```
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "deadline.us-west-2.amazonaws.com"
                }
            }
        }
    ]
}
```

监控您的加密密钥

当您在 Deadline Cloud 服务器场中使用 AWS KMS 客户托管密钥时，您可以使用AWS CloudTrail或Amazon CloudWatch Logs 来跟踪 Deadline Cloud 发送到的请求 AWS KMS。

CloudTrail 补助金活动

以下示例 CloudTrail 事件发生在创建授权时，通常是在您调用CreateFarmCreateMonitor、或CreateFleet操作时。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:SampleUser01",
        "arn": "arn:aws::sts::111122223333:assumed-role/Admin/SampleUser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws::iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
```

```
                "creationDate": "2024-04-23T02:05:26Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "deadline.amazonaws.com"
    },
    "eventTime": "2024-04-23T02:05:35Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "deadline.amazonaws.com",
    "userAgent": "deadline.amazonaws.com",
    "requestParameters": {
        "operations": [
            "CreateGrant",
            "Decrypt",
            "DescribeKey",
            "Encrypt",
            "GenerateDataKey"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:deadline:farmId": "farm-abcdef1234567890098765432 1fedcba",
                "aws:deadline:accountId": "111122223333"
            }
        },
        "granteePrincipal": "deadline.amazonaws.com",
        "keyId": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "retiringPrincipal": "deadline.amazonaws.com"
    },
    "responseElements": {
        "grantId": "6bbe819394822a400fe5e3a75d0e9ef16c1733143fff0c1fc00dc7ac282a18a0",
        "keyId": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
```

```
            "ARN": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE44444"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

CloudTrail 用于解密的事件

使用客户托管的 KMS 密钥解密值时会发生以下示例 CloudTrail 事件。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:SampleUser01",
        "arn": "arn:aws::sts::111122223333:assumed-role/SampleRole/SampleUser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws::iam::111122223333:role/SampleRole",
                "accountId": "111122223333",
                "userName": "SampleRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-04-23T18:46:51Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "deadline.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:51:44Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "deadline.amazonaws.com",
    "userAgent": "deadline.amazonaws.com",
```

```
    "requestParameters": {
        "encryptionContext": {
            "aws:deadline:farmId": "farm-abcdef1234567890098765432 1fedcba",
            "aws:deadline:accountId": "111122223333",
            "aws-crypto-public-key": "AotL+SAMPLEVALUEiOMEXAMPLEaaqNOTREALaGTESTONLY
+p/5H+EuKd4Q=="
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "responseElements": null,
    "requestID": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeffffff",
    "eventID": "ffffffff-eeee-dddd-cccc-bbbbbbaaaaaa",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

CloudTrail 加密事件

使用客户托管的 KMS 密钥对值进行加密时，会发生以下示例 CloudTrail 事件。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:SampleUser01",
        "arn": "arn:aws::sts::111122223333:assumed-role/SampleRole/SampleUser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
```

```
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws::iam::111122223333:role/SampleRole",
                "accountId": "111122223333",
                "userName": "SampleRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-04-23T18:46:51Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "deadline.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:52:40Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "deadline.amazonaws.com",
    "userAgent": "deadline.amazonaws.com",
    "requestParameters": {
        "numberOfBytes": 32,
        "encryptionContext": {
            "aws:deadline:farmId": "farm-abcdef1234567890098765432 1fedcba",
            "aws:deadline:accountId": "111122223333",
            "aws-crypto-public-key": "AotL+SAMPLEVALUEiOMEXAMPLEaaqNOTREALaGTESTONLY
+p/5H+EuKd4Q=="
        },
        "keyId": "arn:aws::kms:us-
west-2:111122223333:key/abcdef12-3456-7890-0987-654321fedcba"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE33333"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
```

```
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

删除客户管理的 KMS 密钥

删除 AWS Key Management Service (AWS KMS) 中客户管理的 KMS 密钥具有破坏性，并且具有潜在的危险。这将删除密钥材料以及与此密钥关联的所有元数据，并且不可撤销。删除客户托管 KMS 密钥后，您不能再解密用该此密钥加密的数据。删除密钥意味着数据变得不可恢复。

这就是为什么客户 AWS KMS 在删除 KMS 密钥之前有长达 30 天的等待期。默认的等待期限为 30 天。

关于等待期限

由于删除客户管理的 KMS 密钥具有破坏性和潜在危险，因此我们要求您将等待期设置为 7-30 天。默认的等待期限为 30 天。

但是，实际等待时间可能比您预定的时间长达 24 小时。要获取删除密钥的实际日期和时间，请使用DescribeKey操作。您还可以在 AWS KMS 控制台中的密钥详细信息页面的常规配置部分中参阅密钥计划删除日期。注意时区。

在等待期限内，客户托管密钥状态和密钥状态为等待删除。

- 待删除的客户托管 KMS 密钥不能用于任何加密操作。
- AWS KMS 不会轮换待删除的客户托管 KMS 密钥的支持密钥。

有关删除客户托管的 KMS 密钥的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的删除客户主密钥。

Inter-network 交通隐私

AWS Deadline Cloud 支持亚马逊 Virtual Private Cloud（亚马逊 VPC）来保护连接。Amazon VPC 提供三种功能，以供您用来提高和监控虚拟私有云 (VPC) 的安全性：

您可以使用在 VPC 内运行的亚马逊弹性计算云 (Amazon EC2) 实例来设置客户托管队列 (CMF)。通过部署要使用的 Amazon VPC 终端节点 AWS PrivateLink，您的 CMF 中的工作人员与 Deadline Cloud 终端节点之间的流量将保留在您的 VPC 内。此外，您可以将您的 VPC 配置为限制您的实例访问互联网。

在服务管理的车队中，无法通过互联网联系到员工，但他们确实可以访问互联网并通过互联网连接到 Deadline Cloud 服务。每个服务管理的队列都在自己的隔离网络中运行，而工作人员实例仍然专用于单个客户。

选择退出

AWS Deadline Cloud 收集某些运营信息以帮助我们发展和改进 Deadline Cloud。收集的数据包括您的 AWS 帐户 ID 和用户 ID 之类的信息，以便在您遇到问题时我们可以正确识别您的身份 Deadline Cloud。我们还收集 Deadline Cloud 特定信息，例如资源 ID（适用时为 farmID 或 queueID）、产品名称（例如 JobAttachments WorkerAgent、等）和产品版本。

您可以使用应用程序配置选择退出此数据收集。与之交互的每台计算机 Deadline Cloud，包括客户工作站和车队员工，都需要单独选择退出。

Deadline Cloud 显示器-台式机

Deadline Cloud monitor-desktop 会收集操作信息，例如何时发生崩溃以及何时打开应用程序，以帮助我们知道您的应用程序何时出现问题。要选择不收集这些操作信息，请前往设置页面并清除 “开启数据收集以衡量 Deadline Cloud Monitor 的性能”。

在您选择退出后，桌面显示器将不再发送操作数据。之前收集的所有数据都将被保留，并且仍可用于改进服务。有关更多信息，请参阅 数据隐私 FAQ。

AWS Deadline Cloud CLI 和工具

AWS Deadline Cloud CLI、提交者和工作人员代理都会收集操作信息，例如何时发生崩溃以及何时提交作业，以帮助我们知道您在使用这些应用程序时遇到问题。要选择不收集此操作信息，请使用以下任一方法：

- 在终端中输入**`deadline config set telemetry.opt_out true`**。

  当以当前用户身份运行时，这将选择退出 CLI、提交者和工作器代理。
- 安装 Deadline Cloud 工作器代理时，添加**`--telemetry-opt-out`**命令行参数。例如 **`./install.sh --farm-id $FARM_ID --fleet-id $FLEET_ID --telemetry-opt-out`**。
- 在运行工作器代理、CLI 或提交器之前，请设置环境变量：**`DEADLINE_CLOUD_TELEMETRY_OPT_OUT=true`**

在您选择退出后，这些 Deadline Cloud 工具将不再发送操作数据。之前收集的所有数据都将被保留，并且仍可用于改进服务。有关更多信息，请参阅 数据隐私 FAQ。

Deadline Cloud 中的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（有权限）使用 Deadline Cloud 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

受众

您的使用方式 AWS Identity and Access Management (IAM) 因您的角色而异：

- 服务用户：如果您无法访问功能，请从管理员处请求权限（请参阅问题排查 AWS 截止日期云身份和访问权限）
- 服务管理员：确定用户访问权限并提交权限请求（请参阅截止日期云如何与 IAM 配合使用）
- IAM 管理员：编写用于管理访问权限的策略（请参阅Identity-based 截止日期云的策略示例）

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份进行身份验证 AWS 账户根用户，或者通过担任 IAM 角色进行身份验证。

您可以使用来自身份源的证书 AWS IAM Identity Center （例如（IAM Identity Center）、单点登录身份验证或 Google/Facebook 证书，以联合身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的如何登录您的 AWS 账户。

对于编程访问，AWS 提供 SDK 和 CLI 来对请求进行加密签名。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的AWS 签名版本 4。

AWS 账户 根用户

创建时 AWS 账户，首先会有一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关需要根用户凭证的任务，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户使用与身份提供商的联合身份验证才能 AWS 服务 使用临时证书进行访问。

联合身份是指来自您的企业目录、Web 身份提供商的用户 Directory Service ，或者 AWS 服务 使用来自身份源的凭据进行访问的用户。联合身份代入可提供临时凭证的角色。

要集中管理访问权限，建议使用。 AWS IAM Identity Center有关更多信息，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 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户使用身份提供商的联合身份验证才能 AWS 使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。 AWS 在委托人提出请求时评估这些政策。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

Identity-based 政策

Identity-based 策略是您附加到身份（用户、组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

Identity-based 策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。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。

Resource-based 政策

Resource-based 策略是您附加到资源的 JSON 策略文档。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。您必须在基于资源的策略中指定主体。

Resource-based 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 IAM 中的 AWS 托管策略。

其他策略类型

AWS 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，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）– 指定 AWS Organizations中组织或组织单元的最大权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。

- 资源控制策略（RCP）– 设置对账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的资源控制策略（RCP）。
- 会话策略 – 在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。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。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

截止日期云如何与 IAM 配合使用

在使用 IAM 管理 Deadline Cloud 的访问权限之前，请先了解哪些可用于 Deadline Cloud 的 IAM 功能。

您可以搭配使用的 IAM 功能 AWS 截止日期云

IAM 功能	截止日期云支持
Identity-based 政策	是
Resource-based 政策	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否
ABAC（策略中的标签）	是
临时凭证	是
转发访问会话（FAS）	是
服务角色	是

IAM 功能	截止日期云支持
Service-linked 角色	否

要全面了解 Deadline Cloud 和其他功能如何 AWS 服务 与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中与 IAM 配合使用的AWS 服务。

Identity-based 截止日期云的政策

支持基于身份的策略：是

Identity-based 策略是您可以附加到身份（例如 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》中的使用客户管理型策略定义自定义 IAM 权限。

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。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。

Identity-based 截止日期云的策略示例

要查看 Deadline Cloud 基于身份的策略的示例，请参阅。Identity-based 截止日期云的策略示例

Resource-based 截止日期云中的政策

支持基于资源的策略：否

Resource-based 策略是您附加到资源的 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。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户访问，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。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

截止日期云的政策行动

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。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。在策略中包含操作以授予执行关联操作的权限。

要查看 Deadline Cloud 操作列表，请参阅《服务授权参考》中的 De AWS adline Cloud 定义的操作。

Deadline Cloud 中的策略操作在操作前使用以下前缀：

```
deadline
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "deadline:action1",
      "deadline:action2"
         ]
```

要查看 Deadline Cloud 基于身份的策略的示例，请参阅。Identity-based 截止日期云的策略示例

截止日期云的政策资源

支持策略资源：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。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。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于不支持资源级权限的操作，请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

要查看 Deadline Cloud 资源类型及其 ARN 的列表，请参阅《服务授权参考》中的 De AWS adline Cloud 定义的资源。要了解您可以使用哪些操作来指定每种资源的 ARN，请参阅 Deadline Clou d 定义的 AWS 操作。

要查看 Deadline Cloud 基于身份的策略的示例，请参阅。Identity-based 截止日期云的策略示例

截止日期云的策略条件密钥

支持特定于服务的策略条件键：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。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。

`Condition` 元素根据定义的条件指定语句何时执行。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。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

要查看 Deadline Cloud 条件密钥列表，请参阅《服务授权参考》中的 D e AWS adline Cloud 条件密钥。要了解可以使用条件键的操作和资源，请参阅 De AWS adline Cloud 定义的操作。

要查看 Deadline Cloud 基于身份的策略的示例，请参阅。Identity-based 截止日期云的策略示例

截止日期云中的 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，但它们不使用 JSON 策略文档格式。

带有截止日期云的 ABAC

支持 ABAC（策略中的标签）：是

Attribute-based 访问控制 (ABAC) 是一种授权策略，它根据称为标签的属性来定义权限。您可以将标签附加到 IAM 实体和 AWS 资源，然后设计 ABAC 策略以允许在委托人的标签与资源上的标签匹配时进行操作。

要基于标签控制访问，您需要使用 `aws:ResourceTag/`*`key-name`*`aws:RequestTag/`*`key-name`* 或 `aws:TagKeys` 条件键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》中的使用 ABAC 授权定义权限。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）。

在截止日期云中使用临时证书

支持临时凭证：是

临时证书提供对 AWS 资源的短期访问权限，并且是在您使用联合身份或切换角色时自动创建的。AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的临时安全凭证和使用 IAM 的。AWS 服务

截止日期云的转发访问会话

支持转发访问会话（FAS）：是

转发访问会话 (FAS) 使用调用主体的权限 AWS 服务，再加上 AWS 服务 向下游服务发出请求的请求。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

截止日期云的服务角色

支持服务角色：是

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。

> **Warning**
>
> 更改服务角色的权限可能会中断 Deadline Cloud 的功能。仅当 Deadline Cloud 提供相关指导时才编辑服务角色。

Service-linked 截止日期云的角色

支持服务相关角色：否

服务相关角色是一种链接到的服务角色。 AWS 服务该服务可以代替您执行操作。 Service-linked 角色出现在您的， AWS 账户 并且归服务所有。IAM 管理员可以查看但不能编辑服务关联角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找Service-linked 角色列Yes中包含的服务。选择是链接以查看该服务的服务相关角色文档。

Identity-based 截止日期云的策略示例

默认情况下，用户和角色无权创建或修改 Deadline Cloud 资源。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 策略。

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（控制台）。

有关 Deadline Cloud 定义的操作和资源类型（包括每种资源类型的 ARN 格式）的详细信息，请参阅《服务授权参考》中的 De AWS adline Cloud 的操作、资源和条件密钥。

主题

- 策略最佳实践
- 使用截止日期云控制台
- 访问控制台的策略
- 向队列提交作业的政策
- 允许创建许可证端点的策略
- 允许监控特定服务器场队列的策略

策略最佳实践

Identity-based 策略决定了某人是否可以在您的账户中创建、访问或删除 Deadline Cloud 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，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管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 托管策略或工作职能的AWS 托管策略。
- 应用最低权限：在使用 IAM 策略设置权限时，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 策略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。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 CloudFormation。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 策略，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 策略语言（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，以帮助您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》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。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》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用截止日期云控制台

要访问 De AWS adline Cloud 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 Deadline Cloud 资源的详细信息 AWS 账户。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以使用 Deadline Cloud 控制台，还需要将 Deadline Cloud *ConsoleAccess* 或*ReadOnly* AWS 托管策略附加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

访问控制台的策略

要授予对 Deadline Cloud 控制台中所有功能的访问权限，请将此身份策略附加到您想要拥有完全访问权限的用户或角色。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Sid": "EC2InstanceTypeSelection",
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeInstanceTypeOfferings",
            "ec2:DescribeInstanceTypes",
            "ec2:GetInstanceTypesFromInstanceRequirements",
            "pricing:GetProducts"
        ],
        "Resource": ["*"]
    },
    {
        "Sid": "VPCResourceSelection",
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups"
```

```
        ],
        "Resource": ["*"]
    },
    {
        "Sid": "ViewVpcLatticeResources",
        "Effect": "Allow",
        "Action": [
            "vpc-lattice:ListResourceConfigurations",
            "vpc-lattice:GetResourceConfiguration",
            "vpc-lattice:GetResourceGateway"
        ],
        "Resource": ["*"]
    },
    {
        "Sid": "ManageVpcEndpointsViaDeadline",
        "Effect": "Allow",
        "Action": [
            "ec2:CreateVpcEndpoint",
            "ec2:DescribeVpcEndpoints",
            "ec2:DeleteVpcEndpoints",
        "ec2:CreateTags"
        ],
        "Resource": ["*"],
        "Condition": {
        "StringEquals": { "aws:CalledViaFirst": "deadline.amazonaws.com" }
        }
    },
    {
        "Sid": "ChooseJobAttachmentsBucket",
        "Effect": "Allow",
        "Action": ["s3:GetBucketLocation", "s3:ListAllMyBuckets"],
        "Resource": "*"
    },
    {
        "Sid": "CreateDeadlineCloudLogGroups",
        "Effect": "Allow",
        "Action": ["logs:CreateLogGroup"],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/deadline/*",
        "Condition": {
        "StringLike": { "aws:CalledViaFirst": "deadline.amazonaws.com" }
        }
    },
    {
        "Sid": "ValidateDependencies",
```

```
        "Effect": "Allow",
        "Action": ["s3:ListBucket"],
        "Resource": "*",
        "Condition": {
        "StringLike": { "aws:CalledViaFirst": "deadline.amazonaws.com" }
        }
    },
    {
        "Sid": "RoleSelection",
        "Effect": "Allow",
        "Action": ["iam:GetRole", "iam:ListRoles",
"iam:ListAttachedRolePolicies"],
        "Resource": "*"
    },
    {
        "Sid": "PassRoleToDeadlineCloud",
        "Effect": "Allow",
        "Action": ["iam:PassRole"],
        "Condition": {
        "StringLike": { "iam:PassedToService": "deadline.amazonaws.com" }
    },
        "Resource": "*"
    },
    {
        "Sid": "KMSKeySelection",
        "Effect": "Allow",
        "Action": ["kms:ListKeys", "kms:ListAliases"],
        "Resource": "*"
    },
    {
        "Sid": "IdentityStoreReadOnly",
        "Effect": "Allow",
        "Action": [
            "identitystore:DescribeUser",
            "identitystore:DescribeGroup",
            "identitystore:ListGroups",
            "identitystore:ListUsers",
            "identitystore:IsMemberInGroups",
            "identitystore:ListGroupMemberships",
            "identitystore:ListGroupMembershipsForMember",
            "identitystore:GetGroupMembershipId"
    ],
        "Resource": "*"
    },
```

```
    {
        "Sid": "OrganizationAndIdentityCenterIdentification",
        "Effect": "Allow",
        "Action": [
            "sso:ListDirectoryAssociations",
            "organizations:DescribeAccount",
            "organizations:DescribeOrganization",
            "sso:DescribeRegisteredRegions",
            "sso:GetManagedApplicationInstance",
            "sso:GetSharedSsoConfiguration",
            "sso:ListInstances",
            "sso:GetApplicationAssignmentConfiguration",
            "sso:GetSSOStatus",
            "sso:ListRegions",
            "sso:DescribeRegion"
    ],
        "Resource": "*"
    },
    {
        "Sid": "ManagedDeadlineCloudIDCApplication",
        "Effect": "Allow",
        "Action": [
            "sso:CreateApplication",
            "sso:PutApplicationAssignmentConfiguration",
            "sso:PutApplicationAuthenticationMethod",
            "sso:PutApplicationGrant",
            "sso:DeleteApplication",
            "sso:UpdateApplication"
    ],
        "Resource": "*",
        "Condition": {
        "StringLike": { "aws:CalledViaFirst": "deadline.amazonaws.com" }
        }
    },
    {
        "Sid": "ChooseSecret",
        "Effect": "Allow",
        "Action": ["secretsmanager:ListSecrets"],
        "Resource": "*"
    },
    {
        "Sid": "DeadlineMembershipActions",
        "Effect": "Allow",
        "Action": [
```

```
            "deadline:AssociateMemberToFarm",
            "deadline:AssociateMemberToFleet",
            "deadline:AssociateMemberToQueue",
            "deadline:AssociateMemberToJob",
            "deadline:DisassociateMemberFromFarm",
            "deadline:DisassociateMemberFromFleet",
            "deadline:DisassociateMemberFromQueue",
            "deadline:DisassociateMemberFromJob",
            "deadline:ListFarmMembers",
            "deadline:ListFleetMembers",
            "deadline:ListQueueMembers",
            "deadline:ListJobMembers"
    ],
        "Resource": ["*"]
    },
    {
        "Sid": "DeadlineControlPlaneActions",
        "Effect": "Allow",
        "Action": [
            "deadline:CreateMonitor",
            "deadline:GetMonitor",
            "deadline:UpdateMonitor",
            "deadline:DeleteMonitor",
            "deadline:ListMonitors",
            "deadline:CreateFarm",
            "deadline:GetFarm",
            "deadline:UpdateFarm",
            "deadline:DeleteFarm",
            "deadline:ListFarms",
            "deadline:CreateQueue",
            "deadline:GetQueue",
            "deadline:UpdateQueue",
            "deadline:DeleteQueue",
            "deadline:ListQueues",
            "deadline:CreateFleet",
            "deadline:GetFleet",
            "deadline:UpdateFleet",
            "deadline:DeleteFleet",
            "deadline:ListFleets",
            "deadline:ListWorkers",
            "deadline:CreateQueueFleetAssociation",
            "deadline:GetQueueFleetAssociation",
            "deadline:UpdateQueueFleetAssociation",
            "deadline:DeleteQueueFleetAssociation",
```

```
            "deadline:ListQueueFleetAssociations",
            "deadline:CreateQueueEnvironment",
            "deadline:GetQueueEnvironment",
            "deadline:UpdateQueueEnvironment",
            "deadline:DeleteQueueEnvironment",
            "deadline:ListQueueEnvironments",
            "deadline:CreateLimit",
            "deadline:GetLimit",
            "deadline:UpdateLimit",
            "deadline:DeleteLimit",
            "deadline:ListLimits",
            "deadline:CreateQueueLimitAssociation",
            "deadline:GetQueueLimitAssociation",
            "deadline:DeleteQueueLimitAssociation",
            "deadline:UpdateQueueLimitAssociation",
            "deadline:ListQueueLimitAssociations",
            "deadline:CreateStorageProfile",
            "deadline:GetStorageProfile",
            "deadline:UpdateStorageProfile",
            "deadline:DeleteStorageProfile",
            "deadline:ListStorageProfiles",
            "deadline:ListStorageProfilesForQueue",
            "deadline:ListBudgets",
            "deadline:TagResource",
            "deadline:UntagResource",
            "deadline:ListTagsForResource",
            "deadline:CreateLicenseEndpoint",
            "deadline:GetLicenseEndpoint",
            "deadline:DeleteLicenseEndpoint",
            "deadline:ListLicenseEndpoints",
            "deadline:ListAvailableMeteredProducts",
            "deadline:ListMeteredProducts",
            "deadline:PutMeteredProduct",
            "deadline:DeleteMeteredProduct",
            "deadline:GetMonitorSettings",
            "deadline:UpdateMonitorSettings"
        ],
        "Resource": ["*"]
      }]
}
```

向队列提交作业的政策

在此示例中，您创建了一个范围缩小策略，该策略授予向特定服务器场中的特定队列提交作业的权限。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "SubmitJobsFarmAndQueue",
            "Effect": "Allow",
            "Action": "deadline:CreateJob",
            "Resource": "arn:aws:deadline:us-east-1:111122223333:farm/FARM_A/
queue/QUEUE_B/job/*"
        }
    ]
}
```

允许创建许可证端点的策略

在此示例中，您将创建一个范围缩小策略，该策略授予创建和管理许可证端点所需的权限。使用此策略为与您的服务器场关联的 VPC 创建许可证终端节点。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Sid": "CreateLicenseEndpoint",
        "Effect": "Allow",
        "Action": [
            "deadline:CreateLicenseEndpoint",
            "deadline:DeleteLicenseEndpoint",
            "deadline:GetLicenseEndpoint",
            "deadline:ListLicenseEndpoints",
            "deadline:PutMeteredProduct",
            "deadline:DeleteMeteredProduct",
            "deadline:ListMeteredProducts",
            "deadline:ListAvailableMeteredProducts",
```

```
            "ec2:CreateVpcEndpoint",
            "ec2:DescribeVpcEndpoints",
            "ec2:DeleteVpcEndpoints"
        ],
        "Resource": [
            "arn:aws:deadline:*:111122223333:*",
            "arn:aws:ec2:*:111122223333:vpc-endpoint/*"
        ]
    }]
}
```

允许监控特定服务器场队列的策略

在此示例中，您创建了一个范围缩小策略，该策略授予监视特定服务器场特定队列中作业的权限。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Sid": "MonitorJobsFarmAndQueue",
        "Effect": "Allow",
        "Action": [
            "deadline:SearchJobs",
            "deadline:ListJobs",
            "deadline:GetJob",
            "deadline:SearchSteps",
            "deadline:ListSteps",
            "deadline:ListStepConsumers",
            "deadline:ListStepDependencies",
            "deadline:GetStep",
            "deadline:SearchTasks",
            "deadline:ListTasks",
            "deadline:GetTask",
            "deadline:ListSessions",
            "deadline:GetSession",
            "deadline:ListSessionActions",
            "deadline:GetSessionAction"
        ],
        "Resource": [
            "arn:aws:deadline:us-east-1:123456789012:farm/FARM_A/queue/QUEUE_B",
            "arn:aws:deadline:us-east-1:123456789012:farm/FARM_A/queue/QUEUE_B/*"
```

```
        ]
    }]
}
```

AWS 截止日期云的托管策略

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住， AWS 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于使用案例的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务 的 API 操作时更新 AWS 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

AWS 托管策略：AWSDeadlineCloud-FleetWorker

您可以将`AWSDeadlineCloud-FleetWorker`策略附加到您的 AWS Identity and Access Management (IAM) 身份。

此策略向该队列中的工作人员授予连接服务并从该服务接收任务所需的权限。

权限详细信息

该策略包含以下权限：

- `deadline`— 允许校长管理车队中的员工。

有关策略详情的 JSON 列表，请参阅 AWSDeadlineCloud-FleetWorkerAWS 托管策略参考指南。

AWS 托管策略：AWSDeadlineCloud-WorkerHost

您可以将 `AWSDeadlineCloud-WorkerHost` 策略附加到 IAM 身份。

此策略授予最初连接到服务所需的权限。它可以用作亚马逊弹性计算云 (Amazon EC2) 实例配置文件。

权限详细信息

该策略包含以下权限：

- deadline— 允许用户创建工作人员、为工作人员担任车队角色以及将标签应用于工作人员

有关策略详情的 JSON 列表，请参阅 AWSDeadlineCloud-WorkerHostAWS 托管策略参考指南。

AWS 托管策略：AWSDeadlineCloud-UserAccessFarms

您可以将 AWSDeadlineCloud-UserAccessFarms 策略附加到 IAM 身份。

此策略允许用户根据其所属的服务器场及其成员级别访问服务器场数据。

权限详细信息

该策略包含以下权限：

- deadline— 允许用户访问服务器场数据。
- ec2— 允许用户查看有关 Amazon EC2 实例类型的详细信息。
- identitystore— 允许用户查看用户名和组名。
- kms— 允许用户为其 AWS Key Management Service （ IAM 身份中心AWS KMS ）实例配置 AWS IAM Identity Center () 客户管理的密钥。

有关策略详情的 JSON 列表，请参阅 AWSDeadlineCloud-UserAccessFarmsAWS 托管策略参考指南。

AWS 托管策略：AWSDeadlineCloud-UserAccessFleets

您可以将 AWSDeadlineCloud-UserAccessFleets 策略附加到 IAM 身份。

此政策允许用户根据其所属的农场及其成员级别访问舰队数据。

权限详细信息

该策略包含以下权限：

- deadline— 允许用户访问服务器场数据。

- ec2— 允许用户查看有关 Amazon EC2 实例类型的详细信息。
- identitystore— 允许用户查看用户名和组名。

有关策略详情的 JSON 列表，请参阅 AWSDeadlineCloud-UserAccessFleetsAWS 托管策略参考指南。

AWS 托管策略：AWSDeadlineCloud-UserAccessJobs

您可以将 AWSDeadlineCloud-UserAccessJobs 策略附加到 IAM 身份。

此政策允许用户根据其所属的农场及其成员级别访问作业数据。

权限详细信息

该策略包含以下权限：

- deadline— 允许用户访问服务器场数据。
- ec2— 允许用户查看有关 Amazon EC2 实例类型的详细信息。
- identitystore— 允许用户查看用户名和组名。

有关策略详情的 JSON 列表，请参阅 AWSDeadlineCloud-UserAccessJobsAWS 托管策略参考指南。

AWS 托管策略：AWSDeadlineCloud-UserAccessQueues

您可以将 AWSDeadlineCloud-UserAccessQueues 策略附加到 IAM 身份。

此策略允许用户根据其所属服务器场及其成员级别访问队列数据。

权限详细信息

该策略包含以下权限：

- deadline— 允许用户访问服务器场数据。
- ec2— 允许用户查看有关 Amazon EC2 实例类型的详细信息。
- identitystore— 允许用户查看用户名和组名。

有关策略详情的 JSON 列表，请参阅 AWSDeadlineCloud-UserAccessQueuesAWS 托管策略参考指南。

截止日期云更新至 AWS 托管策略

查看自该服务开始跟踪这些更改以来 Deadline Cloud AWS 托管政策更新的详细信息。要获得有关此页面变更的自动提醒，请在 Deadlin e Cloud 文档历史记录页面上订阅 RSS 提要。

更改	描述	日期
AWSDeadlineCloud-UserAccessFarms – 更改	Deadline Cloud 添加了新操作，`kms:Decrypt` 因此您可以在 IAM 身份中心实例中使用 AWS KMS 客户管理的密钥。	2025年12月22日
AWSDeadlineCloud-WorkerHost – 更改	Deadline Cloud 添加了新的操作`deadline:TagResource` ，并允许您添加和查看与车队中的工作人员相关的标签。`deadline:ListTagsForResource`	2025 年 5 月 30 日
AWSDeadlineCloud-UserAccessFarms – 更改 AWSDeadlineCloud-UserAccessJobs – 更改 AWSDeadlineCloud-UserAccessQueues – 更改	Deadline Cloud 添加了新的操作`deadline:GetJobTemplate` 并`deadline:ListJobParameterDefinitions` 允许您重新提交作业。	2024 年 10 月 7 日
截止日期云开始跟踪变更	Deadline Cloud 开始跟踪其 AWS 托管政策的变更。	2024 年 4 月 2 日

服务角色

截止日期云如何使用 IAM 服务角色

Deadline Cloud 会自动担任 IAM 角色并为员工、工作和 Deadline Cloud 监控器提供临时证书。这种方法消除了手动凭证管理，同时通过基于角色的访问控制来维护安全性。

在创建监视器、队列和队列时，您可以指定 Deadline Cloud 代表您担任的 IAM 角色。然后，工作人员和 Deadline Cloud 监控器会收到来自这些角色的临时凭证进行访问 AWS 服务。

舰队角色

配置队列角色以授予 Deadline Cloud 工作人员接收工作和报告工作进度所需的权限。

通常，您不必自己配置此角色。可以在 Deadline Cloud 控制台中为您创建此角色以包含必要的权限。使用以下指南了解此角色的详细信息以进行故障排除。

以编程方式创建或更新队列时，请使用或 API 操作指定舰队角色 ARN。`CreateFleet` `UpdateFleet`

舰队角色的作用

舰队角色为工作人员提供以下权限：

- 接收新工作并向 Deadline Cloud 服务报告正在进行的工作进度
- 管理工作人员的生命周期和状态
- 将日志事件记录到 Amazon CloudWatch 日志中以获取工作日志

设置舰队角色信任策略

您的舰队角色必须信任 Deadline Cloud 服务，并且范围仅限于您的特定服务器场。

作为最佳实践，信任策略应包括保护混乱副手的安全条件。要了解有关混乱副手保护的更多信息，请参阅 De adline Cloud 用户指南中的困惑副手。

- `aws:SourceAccount`确保只有来自同一个人的资源 AWS 账户 才能担任此角色。
- `aws:SourceArn`将角色担任限制为特定的 Deadline Cloud 场。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDeadlineCredentialsService",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "credentials.deadline.amazonaws.com"
      },
```

```
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "YOUR_ACCOUNT_ID"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:deadline:REGION:YOUR_ACCOUNT_ID:farm/YOUR_FARM_ID"
        }
      }
    }
  ]
}
```

附加舰队角色权限

将以下 AWS 托管策略附加到您的队列角色：

AWSDeadlineCloud-FleetWorker

此托管策略为以下各项提供权限：

- deadline:AssumeFleetRoleForWorker-允许工作人员刷新其凭证。
- deadline:UpdateWorker-允许工作人员更新其状态（例如，退出时更改为“已停止”）。
- deadline:UpdateWorkerSchedule-用于获取工作和报告进度。
- deadline:BatchGetJobEntity-用于获取工作信息。
- deadline:AssumeQueueRoleForWorker-用于在任务执行期间访问队列角色凭证。

为加密场添加 KMS 权限

如果您的服务器场是使用 KMS 密钥创建的，请将这些权限添加到您的队列角色中，以确保工作人员可以访问服务器场中的加密数据。

仅当您的服务器场具有关联的 KMS 密钥时，才需要 KMS 权限。kms:ViaService条件必须使用格式deadline.*{region}*.amazonaws.com。

创建队列时，会为该队列创建 CloudWatch 日志组。Deadline Cloud 服务使用工作人员的权限来创建专门针对该特定工作人员的日志流。工作器设置并运行后，工作人员将使用这些权限将日志事件直接发送到 Lo CloudWatch gs。

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

```
    {
      "Sid": "CreateLogStream",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream"
      ],
      "Resource": "arn:aws:logs:REGION:YOUR_ACCOUNT_ID:log-group:/aws/
deadline/YOUR_FARM_ID/*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "deadline.REGION.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid": "ManageLogEvents",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:GetLogEvents"
      ],
      "Resource": "arn:aws:logs:REGION:YOUR_ACCOUNT_ID:log-group:/aws/
deadline/YOUR_FARM_ID/*"
    },
    {
      "Sid": "ManageKmsKey",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:GenerateDataKey"
      ],
      "Resource": "YOUR_FARM_KMS_KEY_ARN",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "deadline.REGION.amazonaws.com"
        }
      }
    }
  ]
}
```

修改舰队角色

舰队角色的权限不可自定义。所描述的权限始终是必需的，添加其他权限无效。

Customer-managed 舰队主持人角色

如果您在 Amazon EC2 实例或本地主机上使用客户管理的队列，请设置 WorkerHost 角色。

这个 WorkerHost 角色是做什么的

该 WorkerHost 角色在客户管理的车队主机上引导员工。它为主机提供了执行以下操作所需的最低权限：

- 在截止日期云中创建工作人员
- 扮演舰队角色以获取操作凭证
- 使用舰队标签标记工作人员（如果启用了标签传播）

设置 WorkerHost 角色权限

将以下 AWS 托管策略附加到您的 WorkerHost 角色：

AWSDeadlineCloud-WorkerHost

此托管策略为以下各项提供权限：

- `deadline:CreateWorker`-允许主持人注册新工作人员。
- `deadline:AssumeFleetRoleForWorker`-允许主机扮演舰队角色。
- `deadline:TagResource`-允许在创建过程中标记工作人员（如果启用）。
- `deadline:ListTagsForResource`-允许读取舰队标签进行传播。

了解引导流程

该 WorkerHost 角色仅在工作器初始启动期间使用：

1. 工作器代理使用 WorkerHost 凭据在主机上启动。
2. 它会调用在 Deadlin `deadline:CreateWorker` e Cloud 上注册。
3. 然后它会调用`deadline:AssumeFleetRoleForWorker`以获取舰队角色证书。
4. 从现在开始，工作人员仅使用舰队角色凭证进行所有操作。

在工作人员开始运行后不使用该 WorkerHost 角色。 Service-managed 车队不需要此政策。在 Service-managed 舰队中，引导是自动执行的。

队列角色

队列角色由工作人员在处理任务时担任。此角色提供完成任务所需的权限。

以编程方式创建或更新队列时，请使用`CreateQueue`或`UpdateQueue` API 操作指定队列角色 ARN。

设置队列角色信任策略

您的队列角色必须信任 Deadline Cloud 服务。

作为最佳实践，信任策略应包括保护混乱副手的安全条件。要了解有关混乱副手保护的更多信息，请参阅 De adline Cloud 用户指南中的困惑副手。

- `aws:SourceAccount`确保只有来自同一个人的资源 AWS 账户 才能担任此角色。
- `aws:SourceArn`将角色担任限制为特定的 Deadline Cloud 场。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "credentials.deadline.amazonaws.com",
          "deadline.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "YOUR_ACCOUNT_ID"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:deadline:us-west-2:123456789012:farm/{farm-id}"
        }
      }
    }
  ]
}
```

了解队列角色权限

队列角色不使用单个托管策略。相反，当您在控制台中配置队列时，Deadline Cloud 会根据您的配置为您的队列创建自定义策略。

此自动创建的策略提供对以下内容的访问权限：

Job 附件

对您指定 Amazon S3 存储桶的任务输入和输出文件具有读写权限：

```
{
  "Effect": "Allow",
  "Action": [
    "s3:GetObject",
    "s3:PutObject",
    "s3:ListBucket",
    "s3:GetBucketLocation"
  ],
  "Resource": [
    "arn:aws:s3:::YOUR_JOB_ATTACHMENTS_BUCKET",
    "arn:aws:s3:::YOUR_JOB_ATTACHMENTS_BUCKET/YOUR_PREFIX/*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:ResourceAccount": "YOUR_ACCOUNT_ID"
    }
  }
}
```

作业日志

读取该队列中作业 CloudWatch 日志的访问权限。每个队列都有自己的日志组，每个会话都有自己的日志流：

```
{
  "Effect": "Allow",
  "Action": [
    "logs:GetLogEvents"
  ],
  "Resource": "arn:aws:logs:REGION:YOUR_ACCOUNT_ID:log-group:/aws/
deadline/YOUR_FARM_ID/*"
```

```
}
```

Third-party 软件

可以下载由 Deadline Cloud 支持的第三方软件（例如 Maya、Blender 等）：

```
{
  "Effect": "Allow",
  "Action": [
    "s3:ListBucket",
    "s3:GetObject"
  ],
  "Resource": "*",
  "Condition": {
    "ArnLike": {
      "s3:DataAccessPointArn": "arn:aws:s3:*:*:accesspoint/deadline-software-*"
    },
    "StringEquals": {
      "s3:AccessPointNetworkOrigin": "VPC"
    }
  }
}
```

为您的任务添加权限

为您的队列角色添加任务需要访问的权限。 AWS 服务 在编写 OpenJobDescription 步骤脚本时，AWS CLI 和 SDK 将自动使用您的队列角色提供的凭据。使用它来访问完成工作所需的其他服务。

示例使用案例包括：

- 用于获取自定义数据
- 通过隧道传输到自定义许可证服务器的 SSM 权限
- CloudWatch 用于发布自定义指标
- Deadline Cloud 允许为动态工作流程创建新作业

如何使用队列角色证书

Deadline Cloud 提供队列角色凭证给

- 工作执行期间的工作人员
- 用户在与作业附件和日志交互时通过 Deadline Cloud CLI 和监控器

Deadline Cloud 为每个队列创建单独的 CloudWatch 日志日志组。作业使用队列角色凭据将日志写入队列的日志组。Deadline Cloud CLI 和监控器使用队列角色（通过`deadline:AssumeQueueRoleForRead`）从队列的日志组中读取作业日志。Deadline Cloud CLI 和监控器使用队列角色（通过`deadline:AssumeQueueRoleForUser`）上传或下载作业附件数据。

监视者角色

配置监控角色以授予 Deadline Cloud 监控器 Web 和桌面应用程序访问您的 Deadline Cloud 资源的权限。

以编程方式创建或更新监控器时，请使用`CreateMonitor`或`UpdateMonitor` API 操作指定监控角色 ARN。

监视者角色的用途

监视者角色使 Deadline Cloud 监控器能够为最终用户提供访问以下内容的权限：

- Deadline Cloud 集成提交者、CLI 和监控器所需的基本功能
- 面向最终用户的自定义功能

设置监控角色信任策略

您的监控角色必须信任 Deadline Cloud 服务。

作为最佳实践，信任策略应包括保护混乱副手的安全条件。要了解有关混乱副手保护的更多信息，请参阅 De adline Cloud 用户指南中的困惑副手。

`aws:SourceAccount`确保只有来自同一个人的资源 AWS 账户 才能担任此角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.deadline.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "YOUR_ACCOUNT_ID"
        }
```

```
      }
    }
  ]
}
```

附加监视者角色权限

将以下所有 AWS 托管策略附加到您的监控角色以进行基本操作：

- AWSDeadlineCloud-UserAccessFarms
- AWSDeadlineCloud-UserAccessFleets
- AWSDeadlineCloud-UserAccessJobs
- AWSDeadlineCloud-UserAccessQueues

监视者角色的工作原理

使用 Deadline Cloud 监控器时，服务用户使用 AWS IAM Identity Center （ IAM Identity Center ）登录，并担任监控角色。监视器应用程序使用代入的角色凭据来显示监视器用户界面，包括服务器场、队列、队列和其他信息的列表。

使用 Deadline Cloud monitor 桌面应用程序时，还会使用与最终用户提供的配置文件名称相对应的命名 AWS 凭据配置文件在工作站上提供这些凭据。在 AWS SDK 和工具参考指南中了解有关命名配置文件的更多信息。

这个命名的个人资料是 Deadline CLI 和提交者访问 Deadline Cloud 资源的方式。

为高级用例自定义监控角色

您可以自定义监视者角色以修改用户在每个访问级别（查看者、参与者、管理者、所有者）可以执行的操作，或者添加高级工作流程的权限。

自定义访问级别权限

附加到监控角色的四个 AWS 托管策略控制每个访问级别可以执行的操作。您可以使用`deadline:MembershipLevel`条件键向监控角色添加自定义策略，以授予或限制特定访问级别的权限。

例如，要允许 Contributors 更新和取消作业（通常仅限于管理员和所有者），请添加如下政策：

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "deadline:UpdateJob",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "deadline:MembershipLevel": "CONTRIBUTOR"
        }
      }
    }
  ]
}
```

根据此政策，投稿人除了提交工作外，还可以更新和取消作业。

为高级工作流程添加权限

您可以向监控角色添加自定义 IAM 策略，以向所有监控用户授予额外权限。这对于高级脚本工作流程非常有用，在这些工作流程中，用户需要访问 AWS 服务 超出标准的 Deadline Cloud 功能的权限。

修改您的监视者角色时，请遵循以下准则：

- 不要移除任何托管策略。移除这些策略会中断监视器功能。

Deadline 云监控器如何使用监控角色凭据

Deadline Cloud monitor 会在您进行身份验证时自动获取监控角色凭据 此功能使桌面应用程序能够提供比标准 Web 浏览器更强大的监控功能。

当你使用 Deadline Cloud 监控器登录时，它会自动创建一个可供你使用 AWS CLI 或任何其他 AWS 工具的配置文件。此配置文件使用监控角色证书， AWS 服务 根据您的监控角色中的权限为您提供编程访问权限。

Deadline Cloud 提交者的工作方式相同，他们使用 Deadline Cloud monitor 创建的个人资料以适当的角色权限 AWS 服务 进行访问。

最后期限云角色的高级自定义

您可以通过其他权限扩展 Deadline Cloud 角色，以启用基本渲染工作流程之外的高级用例。这种方法利用 Deadline Cloud 的访问管理系统， AWS 服务 根据队列成员资格控制对其他人的访问权限。

与团队协作 AWS CodeCommit

为您的队列角色添加 AWS CodeCommit 权限，以便在项目存储库上启用团队协作。这种方法将 Deadline Cloud 的访问管理系统用于其他用例——只有有权访问特定队列的用户才能获得这些 AWS CodeCommit 权限，从而允许您通过 Deadline Cloud 队列成员资格管理每个项目的存储库访问权限。

这对于美术师需要访问存储在存储 AWS CodeCommit 库中的项目特定资源、脚本或配置文件作为其渲染工作流程一部分的场景非常有用。

添加 AWS CodeCommit 队列角色权限

向您的队列角色添加以下权限以启用 AWS CodeCommit 访问权限：

```
{
  "Effect": "Allow",
  "Action": [
    "codecommit:GitPull",
    "codecommit:GitPush",
    "codecommit:GetRepository",
    "codecommit:ListRepositories"
  ],
  "Resource": "arn:aws:codecommit:REGION:YOUR_ACCOUNT_ID:PROJECT_REPOSITORY"
}
```

在艺术家工作站上设置凭证提供者

将每个艺术家工作站配置为使用 Deadline Cloud 队列凭据进行 AWS CodeCommit 访问。此设置在每个工作站上完成一次。

配置凭证提供商

1. 将凭证提供商配置文件添加到您的 AWS 配置文件 (`~/.aws/config`)：

   ```
   [profile queue-codecommit]
   credential_process = deadline queue export-credentials --farm-id farm-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX --queue-id queue-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   ```

2. 将 Git 配置为使用此配置文件 AWS CodeCommit 存储库：

   ```
   git config --global credential.https://git-codecommit.REGION.amazonaws.com.helper
    '!aws codecommit credential-helper --profile queue-codecommit $@'
   ```

```
git config --global credential.https://git-
codecommit.REGION.amazonaws.com.UseHttpPath true
```

用实际*queue-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*的服务器场和队列 ID 替换*farm-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*和。*REGION*替换为您 AWS 所在的地区（例如，us-west-2）。

使用 AWS CodeCommit 使用队列凭证

配置完成后，Git 操作将在访问 AWS CodeCommit 仓库时自动使用队列角色凭证。该`deadline queue export-credentials`命令返回如下所示的临时证书：

```
{
  "Version": 1,
  "AccessKeyId": "ASIA...",
  "SecretAccessKey": "...",
  "SessionToken": "...",
  "Expiration": "2025-11-10T23:02:23+00:00"
}
```

这些凭据会根据需要自动刷新，Git 操作将无缝运行：

```
git clone https://git-codecommit.REGION.amazonaws.com/v1/repos/PROJECT_REPOSITORY
git pull
git push
```

现在，艺术家无需单独的 AWS CodeCommit 凭据即可使用队列权限访问项目存储库。只有有权访问特定队列的用户才能访问关联的存储库，从而通过 Deadline Cloud 的队列成员资格系统实现精细的访问控制。

问题排查 AWS 截止日期云身份和访问权限

使用以下信息来帮助您诊断和修复在使用 Deadline Cloud 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 Deadline Cloud 中执行操作
- 我无权执行 iam：PassRole
- 我想允许我以外的人进入 AWS 账户 访问我的截止日期云资源

我无权在 Deadline Cloud 中执行操作

如果您收到错误提示，指明您无权执行某个操作，则必须更新策略以允许执行该操作。

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 deadline:*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:
 deadline:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 deadline:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到错误消息，说您无权执行该iam:PassRole操作，则必须更新您的策略以允许您将角色传递给 Deadline Cloud。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的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 Deadline Cloud 的 IAM 用户marymajor尝试使用控制台在 Deadline Cloud 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，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人进入 AWS 账户 访问我的截止日期云资源

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 Deadline Cloud 是否支持这些功能，请参阅截止日期云如何与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 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 中的跨账户资源访问。

的合规性验证 Deadline Cloud

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。有关您在使用时的合规责任的更多信息 AWS 服务，请参阅AWS 安全文档。

韧性在 Deadline Cloud

AWS 全球基础设施是围绕 AWS 区域 可用区构建的。 AWS 区域 提供多个物理分隔和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。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失效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、容错能力和可扩展性。

有关 AWS 区域 和可用区的更多信息，请参阅AWS 全球基础设施。

AWS Deadline Cloud 不会备份存储在任务附件 S3 存储桶中的数据。您可以使用任何标准 Amazon S3 备份机制（例如 S 3 版本控制或 AWS Backup）启用任务附件数据的备份。

截止日期云中的基础设施安全

作为一项托管服务，De AWS adline Cloud 受到 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 调用通过网络访问 Deadline Cloud。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（短暂的）或 ECDHE（椭圆曲线短暂的 Diffie-Hellman）。 Diffie-Hellman大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

Deadline Cloud 不支持使用 AWS PrivateLink 虚拟私有云 (VPC) 端点策略。它使用 AWS PrivateLink 默认策略，即授予对终端节点的完全访问权限。有关更多信息，请参阅AWS PrivateLink 用户指南中的默认终端节点策略。

截止日期云中的配置和漏洞分析

AWS 处理基本的安全任务，例如客户机操作系统 (OS) 和数据库修补、防火墙配置和灾难恢复。这些流程已通过相应第三方审核和认证。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（白皮书）

AWS Deadline Cloud 管理服务管理或客户管理的车队上的任务：

- 对于服务管理的舰队，Deadline Cloud 管理客户机操作系统。
- 对于客户管理的车队，您负责管理操作系统。

有关 De AWS adline Cloud 的配置和漏洞分析的更多信息，请参阅

- 截止日期云的安全最佳实践

Cross-service 混乱的副手预防

混淆代理问题是一个安全性问题，即不具有某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。在中 AWS，跨服务模仿可能会导致混乱的副手问题。 Cross-service 当一个服务（调用服务）调用另一个服务（被调用的服务）时，可能会发生模仿行为。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。

我们建议在资源策略中使用aws:SourceArn和aws:SourceAccount全局条件上下文密钥来限制为资源 AWS Deadline Cloud 提供其他服务的权限。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

防止混淆代理问题最有效的方法是使用具有资源完整 Amazon 资源名称（ARN）的 aws:SourceArn 全局条件上下文键。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字符（*）的 aws:SourceArn 全局上下文条件键。例如 arn:aws:deadline:*:*123456789012*:*。

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键来限制权限。

以下示例显示了如何在中使用aws:SourceArn和aws:SourceAccount全局条件上下文键 Deadline Cloud 来防止出现混淆的副手问题。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "deadline.amazonaws.com"
    },
    "Action": "deadline:CreateFarm",
    "Resource": [
      "*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:deadline:*:111122223333:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "111122223333"
      }
    }
  }
}
```

访问 AWS Deadline Cloud 使用接口端点 (AWS PrivateLink)

您可以使用 AWS PrivateLink 在您的 VPC 和之间创建私有连接 AWS Deadline Cloud。您可以像在 VPC 中 Deadline Cloud 一样进行访问，无需使用互联网网关、NAT 设备、VPN 连接或 Direct Connect 连接。VPC 中的实例不需要公有 IP 地址即可访问 Deadline Cloud。

您可以通过创建由 AWS PrivateLink提供支持的接口端点来建立此私有连接。我们将在您为接口端点启用的每个子网中创建一个端点网络接口。这些是请求者托管的网络接口，用作发往 Deadline Cloud的流量的入口点。

Deadline Cloud 还提供双堆栈端点。 Dual-stack 端点支持通过 IPv6 和 IPv4 发出的请求。

有关更多信息，请参阅《AWS PrivateLink 指南》中的通过 AWS PrivateLink访问 AWS 服务。

的注意事项 Deadline Cloud

在为设置接口终端节点之前 Deadline Cloud，请参阅AWS PrivateLink 指南中的使用接口 VPC 终端节点访问 AWS 服务。

Deadline Cloud 支持通过接口端点调用其所有 API 操作。

默认情况下，允许通过接口终端节点进行完全访问。 Deadline Cloud 或者，您可以将安全组与终端节点网络接口相关联，以控制 Deadline Cloud 通过该接口终端节点的流量。

Deadline Cloud 还支持 VPC 终端节点策略。有关更多信息，请参阅 AWS PrivateLink 指南中的使用端点策略控制对 VPC 端点的访问权限。

Deadline Cloud 端点

Deadline Cloud 使用四个端点访问服务 AWS PrivateLink -两个用于 IPv4，两个用于 IPv6。

工作人员使用`scheduling.deadline.region.amazonaws.com`端点从队列中获取任务、向其Deadline Cloud报告进度以及将任务输出发送回去。如果您使用的是客户管理的队列，则调度终端节点是您唯一需要创建的终端节点，除非您使用的是管理操作。例如，如果一个任务创建了更多作业，则需要启用管理端点才能调用该`CreateJob`操作。

Deadline Cloud 监视器使用`management.deadline.region.amazonaws.com`来管理服务器场中的资源，例如创建和修改队列和队列或获取作业、步骤和任务的列表。

AWS 软件开发工具包和 CLI 会自动将`management`和`scheduling`前缀添加到终端节点。如果要禁用此行为，请参阅《软件开发工具包和AWS 工具参考指南》中的主机前缀注入部分。

Deadline Cloud 还需要以下 AWS 服务端点的终端节点：

- 如果您在没有互联网连接的子网中设置客户管理的队列，则必须为 Amazon L CloudWatch ogs 创建 VPC 终端节点，以便工作人员可以写入日志。有关更多信息，请参阅使用进行监控 CloudWatch。
- 如果您使用任务附件，则必须为亚马逊简单存储服务 (Amazon S3) Simple Storage S3 创建 VPC 终端节点，以便工作人员可以访问附件。有关更多信息，请参阅中的 Job 附件 Deadline Cloud。

为创建终端节点 Deadline Cloud

您可以创建用于 Deadline Cloud 使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 的接口终端节点。有关更多信息，请参阅《AWS PrivateLink 指南》中的创建接口端点。

Deadline Cloud 使用以下服务名称创建管理和调度端点。*region*替换为已部署 AWS 区域 的位置 Deadline Cloud。

```
com.amazonaws.region.deadline.management
```

```
com.amazonaws.region.deadline.scheduling
```

Deadline Cloud 支持双堆栈端点。

如果您为接口终端节点启用私有 DNS，则 Deadline Cloud 可以使用其默认区域 DNS 名称向发出 API 请求。例如，`scheduling.deadline.us-east-1.amazonaws.com`用于工作人员操作或`management.deadline.us-east-1.amazonaws.com`所有其他操作。

如果您的客户管理的队列位于没有 Internet 连接的子网上，则必须使用以下服务名称创建 L CloudWatch ogs 端点：

```
com.amazonaws.region.logs
```

如果您使用任务附件传输文件，则必须使用以下服务名称创建 Amazon S3 终端节点：

```
com.amazonaws.region.s3
```

受限的网络环境

Deadline Cloud 提供艺术家或其他用户在本地工作站上使用的工具。这些工具需要访问 AWS API 和 Web 端点才能执行其功能。如果您使用网络内容过滤解决方案（例如下一代防火墙 (NGFW) 或安全

Web 网关 (SWG)）来过滤对特定 AWS 域或 URL 端点的访问，则必须将以下域或 URL 端点添加到您的网络内容过滤解决方案许可名单中。

AWS 允许列入许可名单的 API 端点

Deadline Cloud 客户端工具，例如监控器 AWS 管理控制台、CLI 和集成提交者，除了 Deadline Cloud 之外，还需要访问 AWS API。这些端点仅支持 IPv4。

- `scheduling.deadline.[Region].amazonaws.com`
- `management.deadline.[Region].amazonaws.com`
- `logs.[Region].amazonaws.com`
- `ec2.[Region].amazonaws.com`
- `s3.[Region].amazonaws.com`
- `sts.[Region].amazonaws.com`
- `identitystore.[Region].amazonaws.com`

要列入许可名单的 Web 域名

Deadline Cloud 监控器需要访问以下域才能运行。

有关允许列入许可名单的域名的更多信息 AWS Sign-In，请参阅《AWS Sign-In 用户指南》中的 “要添加到允许列表的域名”。

- `downloads.deadlinecloud.amazonaws.com`
- `d2ev1rdnjzhmnr.cloudfront.net`
- `prod.log.shortbread.aws.dev`
- `prod.tools.shortbread.aws.dev`
- `prod.log.shortbread.analytics.console.aws.a2z.com`
- `prod.tools.shortbread.analytics.console.aws.a2z.com`
- `global.help-panel.docs.aws.a2z.com`
- `[Region].signin.aws`
- `[Region].signin.aws.amazon.com`
- `sso.[Region].amazonaws.com`
- `portal.sso.[Region].amazonaws.com`

- oidc.*[Region]*.amazonaws.com
- assets.sso-portal.*[Region]*.amazonaws.com

Environment-specific 进入许可名单的终端节点

这些域名因截止日期云的具体配置而异。如果创建了其他 Deadline Cloud 监控器或队列，则需要将其他域名列入许可名单。

- *[Directory ID or alias]*.awsapps.com

  此域与 IAM Identity Center 设置相关联，并且使用相同的 IAM 身份中心实例的所有设置都应相同。企业管理员可以在 IAM Identity Center 控制台的 “设置” → “AWS 访问门户 URL” 下找到确切的值。

- *[Monitor alias]*.*[Region]*.deadlinecloud.amazonaws.com

  此域名用于 Deadline Cloud 中的监控器设置。艺术家将此链接输入他们的浏览器或 Deadline Cloud 监视器应用程序。如果将来在其他账户或地区设置了 Deadline Cloud，则此域名将发生变化。您可以在 Deadline Cloud 控制台的控制面板 → 监控器概述 → 监控器详细信息 → URL 中找到此值。

- *[Bucket name]*.*[Region]*.s3.amazonaws.com

  这是 Deadline Cloud 队列使用的任务附件存储桶的域。每个队列都可以配置自己的任务附件存储桶。可以在 Deadline Cloud 控制台的 “队列” → “队列详情” → “Job 附件” 下找到确切的存储桶名称。有关作业附件的更多信息，请参阅队列文档。

截止日期云的安全最佳实践

AWS Deadline Cloud（Deadline Cloud）提供了许多安全功能，供您在制定和实施自己的安全策略时考虑。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环境或不满足环境要求，请将其视为有用的考虑因素而不是惯例。

> **Note**
>
> 有关许多安全主题的重要性的更多信息，请参阅责任共担模型。

数据保护

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) 设置个人账户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon Simple Storage Service (Amazon S3) 中的个人数据。
- 如果在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅美国联邦信息处理标准（FIPS）第 140-2 版。

我们强烈建议您切勿将敏感的可识别信息（例如您客户的账号）放入自由格式字段（例如名称字段）。此建议包括在您使用控制台、API 或 SD AWS K AWS 服务 使用 Deadline Cloud 或其他工具包时。AWS AWS CLI您输入到Deadline Cloud或其他服务中的任何数据都可能会被提取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。

AWS Identity and Access Management 权限

使用用户、 AWS Identity and Access Management (IAM) 角色并通过向用户授予最低权限来管理对 AWS 资源的访问权限。制定用于创建、分发、轮换和撤消 AWS 访问凭证的凭证管理策略和程序。有关更多信息，请参阅《IAM 用户指南》中的 IAM 最佳实操。

以用户和群组的身份运行作业

在 Deadline Cloud 中使用队列功能时，最佳做法是指定操作系统 (OS) 用户及其主组，以便操作系统用户对队列的作业拥有最低权限权限。

当您指定 “以用户身份运行”（和组）时，提交到队列的作业的所有进程都将使用该操作系统用户运行，并将继承该用户的关联操作系统权限。

队列和队列配置相结合，可以建立安全态势。在队列方面，可以指定 “Job 以用户身份运行” 和 IAM 角色来使用队列任务的操作系统和 AWS 权限。队列定义了基础架构（工作主机、网络、已安装的共享存储），当这些基础架构与特定队列关联时，将在队列中运行作业。工作服务器主机上的可用数据需要由

一个或多个关联队列中的作业访问。指定用户或组有助于保护作业中的数据免受其他队列、其他已安装的软件或其他有权访问工作主机的用户的侵害。当队列没有用户时，它会以代理用户身份运行，代理用户可以模仿 (sudo) 任何队列用户。这样，没有用户的队列可以将权限升级到另一个队列。

Networking

为防止流量被拦截或重定向，必须确保网络流量的路由方式和位置安全。

我们建议您通过以下方式保护您的网络环境：

- 保护亚马逊虚拟私有云（Amazon VPC）子网路由表，以控制 IP 层流量的路由方式。
- 如果您在服务器场或工作站设置中使用亚马逊 Route 53（Route 53）作为 DNS 提供商，请安全访问 Route 53 API。
- 如果您使用本地工作站或其他数据中心 AWS 等外部连接到 Deadline Cloud，请保护任何本地网络基础设施。这包括路由器、交换机和其他网络设备上的 DNS 服务器和路由表。

工作和工作数据

Deadline Cloud 作业在工作主机的会话中运行。每个会话在工作主机上运行一个或多个进程，这通常需要您输入数据才能生成输出。

为了保护这些数据，您可以为操作系统用户配置队列。工作器代理使用队列操作系统用户来运行会话子进程。这些子进程继承队列操作系统用户的权限。

我们建议您遵循最佳实践，以保护对这些子流程访问的数据的访问。有关更多信息，请参阅责任共担模式。

农场结构

您可以通过多种方式安排 Deadline Cloud 舰队和队列。但是，某些安排会涉及安全问题。

服务器场具有最安全的边界之一，因为它无法与其他服务器场共享 Deadline Cloud 资源，包括队列、队列和存储配置文件。但是，您可以在服务器场内共享外部 AWS 资源，这会影响安全边界。

您还可以使用适当的配置在同一服务器场内的队列之间建立安全边界。

按照以下最佳实践在同一个服务器场中创建安全队列：

- 仅将队列与相同安全边界内的队列关联。注意以下几点：

  - 在工作主机上运行作业后，数据可能会留在后面，例如在临时目录或队列用户的主目录中。
  - 无论您将任务提交到哪个队列，都由同一个操作系统用户在服务拥有的队列工作人员主机上运行所有作业。
  - 作业可能会使进程在工作主机上运行，从而使来自其他队列的作业可以观察其他正在运行的进程。
- 确保只有处于相同安全边界内的队列才能共享用于存放任务附件的 Amazon S3 存储桶。
- 确保只有相同安全边界内的队列共享操作系统用户。
- 将集成到服务器场中的任何其他 AWS 资源保护到边界。

Job 附件队列

Job 附件与队列相关联，该队列使用您的 Amazon S3 存储桶。

- Job 附件对 Amazon S3 存储桶中的根前缀进行写入和读取。您可以在 `CreateQueue` API 调用中指定此根前缀。
- 存储桶有一个对应的`Queue Role`，它指定了向队列用户授予存储桶访问权限的角色和根前缀。创建队列时，您可以在任务附件存储桶和根前缀旁边指定 A `Queue Role` mazon 资源名称 (ARN)。
- 对`AssumeQueueRoleForRead`、`AssumeQueueRoleForUser`和 `AssumeQueueRoleForWorker` API 操作的授权调用会返回一组临时安全证书`Queue Role`。

如果您创建队列并重复使用 Amazon S3 存储桶和根前缀，则存在信息被泄露给未授权方的风险。例如，queueA 和 queueB 共享相同的存储桶和根前缀。在安全的工作流程中，ArtistA 可以访问 QueueA，但不能访问 queueB。但是，当多个队列共享一个存储桶时，ArtistA 可以访问 QueueB 数据中的数据，因为它使用的存储桶和根前缀与 queueA 相同。

控制台设置的队列在默认情况下是安全的。除非队列属于共同安全边界，否则请确保队列具有 Amazon S3 存储桶和根前缀的独特组合。

要隔离队列，必须将配置`Queue Role`为仅允许队列访问存储桶和根前缀。在以下示例中，将每个示例替换*`placeholder`*为您的资源特定信息。

JSON

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

```
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::JOB_ATTACHMENTS_BUCKET_NAME",

 "arn:aws:s3:::JOB_ATTACHMENTS_BUCKET_NAME/JOB_ATTACHMENTS_ROOT_PREFIX/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "111122223333"
                }
            }
        },
        {
            "Action": [
                "logs:GetLogEvents"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:/aws/
deadline/FARM_ID/*"
        }
    ]
}
```

您还必须为该角色设置信任策略。在以下示例中，用您的资源特定信息替换*placeholder*文本。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "sts:AssumeRole"
            ],
            "Effect": "Allow",
            "Principal": {
```

```
                "Service": "deadline.amazonaws.com"
            },
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:deadline:us-
east-1:111122223333:farm/FARM_ID"
                }
            }
        },
        {
            "Action": [
                "sts:AssumeRole"
            ],
            "Effect": "Allow",
            "Principal": {
                "Service": "credentials.deadline.amazonaws.com"
            },
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:deadline:us-
east-1:111122223333:farm/FARM_ID"
                }
            }
        }
    ]
}
```

定制软件 Amazon S3 存储桶

您可以在中添加以下语句Queue Role以访问您的 Amazon S3 存储桶中的自定义软件。在以下示例中，*SOFTWARE_BUCKET_NAME*替换为您的 S3 存储桶的名称和*BUCKET_ACCOUNT_OWNER*拥有该存储桶的 AWS 账户 ID。

```
"Statement": [
    {
        "Action": [
```

```
            "s3:GetObject",
            "s3:ListBucket"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:s3:::SOFTWARE_BUCKET_NAME",
            "arn:aws:s3:::SOFTWARE_BUCKET_NAME/*"
        ],
        "Condition": {
         "StringEquals": {
            "aws:ResourceAccount": "BUCKET_ACCOUNT_OWNER"
         }
      }
    }
]
```

有关 Amazon S3 安全最佳实践的更多信息，请参阅《亚马逊简单存储服务用户指南》中的 Amazon S3 安全最佳实践。

工作人员主机

保护工作人员主机，以帮助确保每个用户只能为其分配的角色执行操作。

我们建议采用以下最佳做法来保护工作主机：

- 使用主机配置脚本可以更改工作人员的安全性和操作。不正确的配置可能会导致工作器不稳定或停止工作。您有责任调试此类故障。
- 除非提交给这些队列的任务在相同的安全边界内，否则不要对多个队列使用相同的`jobRunAsUser`值。
- 不要将队列设置`jobRunAsUser`为工作代理运行的操作系统用户的姓名。
- 向队列用户授予目标队列工作负载所需的最低权限操作系统权限。确保他们没有工作代理程序文件或其他共享软件的文件系统写入权限。
- 确保只`Administrator`有 root 用户开启Linux且拥有者账户Windows拥有并可以修改工作代理程序文件。
- 在Linux工作服务器主机上，可以考虑在中配置一个umask替代项`/etc/sudoers`，允许工作器代理用户以队列用户身份启动进程。此配置有助于确保其他用户无法访问写入队列的文件。
- 向受信任的个人授予对工作人员主机的最低权限访问权限。
- 将权限限制为本地 DNS 覆盖配置文件（`/etc/hosts`开启和开`C:\Windows\system32\etc\hosts`启Windows），以及工作站和工作主机操作系统上的路由表。Linux

- 限制工作站和工作主机操作系统上的 DNS 配置权限。
- 定期修补操作系统和所有已安装的软件。这种方法包括专门用于 Deadline Cloud 的软件，例如提交者、适配器、工作人员代理、OpenJD包等。
- 为Windows队列使用强密码`jobRunAsUser`。
- 定期轮换队列的密码`jobRunAsUser`。
- 确保对Windows密码密钥的访问权限最低，并删除未使用的密码。
- 不要向队列`jobRunAsUser`授予将来运行的计划命令的权限：
  - 开启Linux，拒绝这些账户访问`cron`和`at`。
  - 开启Windows，拒绝这些账户访问Windows任务计划程序。

> **Note**
>
> 有关定期修补操作系统和已安装软件的重要性的更多信息，请参阅责任共担模型。

主机配置脚本

- 使用主机配置脚本可以更改工作人员的安全性和操作。不正确的配置可能会导致工作器不稳定或停止工作。您有责任调试此类故障。

工作站

保护能够访问 Deadline Cloud 的工作站非常重要。这种方法有助于确保你提交给 Deadline Cloud 的任何任务都无法运行向你 AWS 账户计费的任意工作负载。

我们建议采用以下最佳做法来保护艺术家工作站的安全。有关更多信息，请参阅 责任共担模式。

- 保护所有提供访问权限的永久凭证，包括 Deadlin AWS e Cloud。有关更多信息，请参阅《 IAM 用户指南》中的管理 IAM 用户的访问密钥。
- 仅安装可信、安全的软件。
- 要求用户与身份提供商联合使用临时证书 AWS 进行访问。
- 对 Deadline Cloud 提交者程序文件使用安全权限以防止篡改。
- 向受信任的个人授予访问艺术家工作站的最低权限。
- 仅使用您通过 Deadline Cloud Monitor 获得的提交者和适配器。

- 将权限限制为本地 DNS 覆盖配置文件（/etc/hosts开启和开C:\Windows\system32\etc\hosts启Windows），以及工作站和工作主机操作系统上的路由表。Linux macOS
- 将权限限制/etc/resolve.conf在工作站和工作主机操作系统上。
- 定期修补操作系统和所有已安装的软件。这种方法包括专门用于 Deadline Cloud 的软件，例如提交者、适配器、工作人员代理、OpenJD包等。

验证已下载软件的真实性

下载安装程序后，请验证软件的真实性，以防文件被篡改。此过程适用于Windows和Linux系统。

Windows

要验证您下载的文件的真实性，请完成以下步骤。

1. 在以下命令中，*file*替换为要验证的文件。例如 ***C:\PATH\TO\MY\DeadlineCloudSubmitter-windows-x64-installer.exe*** 。另外，请*signtool-sdk-version*替换为已安装的 SignTool SDK 版本。例如 **10.0.22000.0**。

   ```
   "C:\Program Files (x86)\Windows Kits\10\bin\signtool-sdk-version\x86\signtool.exe" verify /vfile
   ```

2. 例如，您可以通过运行以下命令来验证 Deadline Cloud 提交者安装程序文件：

   ```
   "C:\Program Files (x86)\Windows Kits\10\bin
   \10.0.22000.0\x86\signtool.exe" verify /v DeadlineCloudSubmitter-
   windows-x64-installer.exe
   ```

Linux

要验证下载文件的真实性，请使用gpg命令行工具。

1. 通过运行以下命令导入OpenPGP密钥：

   ```
    gpg --import --armor <<EOF
   -----BEGIN PGP PUBLIC KEY BLOCK-----

   mQINBGlANDUBEACg6zffjN43gqe5ryPhk+wQM10rEdvmItw4WPWaVsN+/at/OIJw
   MGCagSYXcgR+jKbsHQOQoEQdo5SrxxHjpKTEs3KQhGvf+ehrU1Ac7koXKIBWtes+
   BI9F0slRECz0nXTOy/cd/90RXjpF07mreTLIKNIbybULfad82nYykpITjFr5XRGj
   /shYkucxRQZdwkgkIYyV25pPICPd2RsX+Zua85jV8mCqVffDfRXvgcPe3+ofClj/
   ```

```
2CE8UfUIqO8Csua4YEkSqr3aeoTOEFT4kuQR5nFXVzorOEkQtO3gB35KNWKMlIOU
2vA+wyoL7nWSii4yfYtW3EZ+3gq6HxvnT9Zs8MC53uTOiOdamASXecYREwGmY/io
6n5XTEA/35LNbl4A756vSTZ7h4VFJAN5BpuqxstI1D7ou94skoSmcPoC/iniTvY9
kZylU5OCH/nifMAHM2a5jrQel80cW4oko9eyc8ENQpSy15JElFOKFf7D/4tcZJLF
F0VBTXbhfvq3dPfoq94IWt7p54Ovwj0S//CEu3jZYbNl2QC/3YiHE2H2XyGCQbq6
2MjcuxLnEapoRIqfbi8GPtCWVPzm28WGyKIDofWICczzeJFFJnvzrY3wRG64ibKJ
bR/uedwua1UuiC482V1FD5ffmzSSs8ktTp9hgj7RGDXlc9NTcF1jHxG9hwARAQAB
tCxBV1MgRGVhZGxpbmUgQ2xvdWQgPGF3cy1kZWFkbGluZUBhbWF6b24uY29tPokC
VwQTAQgAQRYhBJmXd7So2csyehiIYsg71N18bhtjBQJpQDQ1AhsvBQkDwmcABQsJ
CAcCAiICBhUKCQgLAgQWAgMBAh4HAheAAAoJEMg71N18bhtjk2UP/3h4KlEzZO/7
BxRmkbixuo1QuqOGvA6tXbSWaM8QH5jglcvL12PZLALklLT4v82uCsLR1lF8/Tch
cCl0SZEOFIS+XxAaw1Xfai6jlyLhabOwKF2ylq5eJlLcw1lh2nAArDRb4fLD0m1g
Dfqetq/XEpyXpOSkWxGRV4RlUdjQfytxrmcUnsT5/fk5f9VDdblu6K/lEmwfyYjB
lXv0uUCkqPot0SmbvOh3PY3Hi3n54ncy8NfTeV+TUvSe3C1s1zNl8aqHoTxJB/eU
kp+LFZ9m+igpSYnKeglKnytylH3KGCjTHglT/QXnI1wNTqmj1kFBVwtt/y1mtnA+
CPIUHP1CtbKsHaLtpp4llBm5TVtPN/Wqqicn5QLl4khg7R4K+V2aaA4ubY6p1tG9
0fFhN5tTnHDSKWMfmb83wfh5Zkcg85c3egjoit+wgGQRAQVqbznx7NqAHs9VoDIu
SPcAr+C329AOBzod4gyNGH7Ah5DkMITo4O4+axnAU9yhFOHcMJmTIask/fNg1Aum
OqYPMUwcgv1GZjLaTJyfGGC1xALsYR0KHnwIehD06MHR/Z98bGkcV8+Y0q8UPsd1
VN1fc1rjCJh/AT3w6owvG4DaEwspseSjzHv16mW4e2N6Uu23SPzgQsJ5qYN2g8D+
P7N9LGDfP8DaYc5JM9mlyFmYI2Q94ufL
=rY5l
-----END PGP PUBLIC KEY BLOCK-----
EOF
```

2. 确定是否信任0penPGP密钥。在决定是否信任上述密钥时需要考虑的一些因素包括：
   - 您用于从本网站获取 GPG 密钥的互联网连接是安全的。
   - 您访问本网站时使用的设备是安全的。
   - AWS 已采取措施保护本网站上0penPGP公钥的托管。
3. 如果您决定信任该OpenPGP密钥，请使用gpg类似于以下示例的方法编辑该密钥以使其可信：

```
$ gpg --edit-key 0xB840C08C29A90796A071FAA5F6CD3CE6B76F3CEF

    gpg (GnuPG) 2.0.22; Copyright (C) 2013 Free Software Foundation, Inc.
    This is free software: you are free to change and redistribute it.
    There is NO WARRANTY, to the extent permitted by law.


    pub  4096R/4BF0B8D2  created: 2023-06-23  expires: 2025-06-22  usage: SCEA
                         trust: unknown       validity: unknown
    [ unknown] (1). AWS Deadline Cloud example@example.com
```

```
   gpg> trust
   pub  4096R/4BF0B8D2  created: 2023-06-23  expires: 2025-06-22  usage: SCEA
                        trust: unknown       validity: unknown
   [ unknown] (1). AWS Deadline Cloud aws-deadline@amazon.com

   Please decide how far you trust this user to correctly verify other users'
keys
   (by looking at passports, checking fingerprints from different sources,
etc.)

     1 = I don't know or won't say
     2 = I do NOT trust
     3 = I trust marginally
     4 = I trust fully
     5 = I trust ultimately
     m = back to the main menu

   Your decision? 5
   Do you really want to set this key to ultimate trust? (y/N) y

   pub  4096R/4BF0B8D2  created: 2023-06-23  expires: 2025-06-22  usage: SCEA
                        trust: ultimate      validity: unknown
   [ unknown] (1). AWS Deadline Cloud aws-deadline@amazon.com
   Please note that the shown key validity is not necessarily correct
   unless you restart the program.

   gpg> quit
```

4. 验证 Deadline Cloud 提交者安装程序

   要验证 Deadline Cloud 提交者安装程序，请完成以下步骤：

   a. 下载 Deadline Cloud 提交者安装程序的签名文件。

      下载签名文件 (.sig)

   b. 运行以下命令验证 Deadline Cloud 提交者安装程序的签名：

      ```
      gpg --verify ./DeadlineCloudSubmitter-linux-x64-installer.run.sig ./
      DeadlineCloudSubmitter-linux-x64-installer.run
      ```

5. 验证截止日期云监视器

> **Note**
>
> 您可以使用签名文件或特定于平台的方法来验证 Deadline Cloud 监控器的下载。有关平台特定的方法，请参阅Linux (Debian)选项卡、Linux (RPM) 选项Linux (AppImage)卡或基于您下载的文件类型的选项卡。

要使用签名文件验证 Deadline Cloud 监控桌面应用程序，请完成以下步骤：

a. 为你的 Deadline Cloud 监视器安装程序下载相应的签名文件：

- 下载.deb 签名文件
- 下载.rpm 签名文件
- 下载。 AppImage 签名文件

b. 验证签名：

对于.deb：

```
gpg --verify ./deadline-cloud-monitor_amd64.deb.sig ./deadline-cloud-
monitor_amd64.deb
```

对于.rpm：

```
gpg --verify ./deadline-cloud-monitor.x86_64.rpm.sig ./deadline-cloud-
monitor.x86_64.rpm
```

对于。 AppImage:

```
gpg --verify ./deadline-cloud-monitor_amd64.AppImage.sig ./deadline-cloud-
monitor_amd64.AppImage
```

c. 确认输出类似于以下内容：

```
gpg: Signature made Mon Apr 1 21:10:14 2024 UTC

gpg: using RSA key B840C08C29A90796A071FAA5F6CD3CE6B7
```

如果输出包含短语Good signature from "AWS Deadline Cloud"，则表示签名已成功通过验证，您可以运行 Deadline Cloud 监视器安装脚本。

历史钥匙

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGX6GQsBEADduUtJgqSXI+q76O6fsFwEYKmbnlyL0xKvlq32EZuyv0otZo5L
le4m5Gg52AzrvPvDiUTLooAlvYeozaYyirIGsK08Ydz0Ftdjroiuh/mw9JSJDJRI
rnRn5yKet1JFezkjopA3pjsTBP6lW/mb1bDBDEwwwtH0x9lV7A03FJ9T7Uzu/qSh
qO/UYdkafro3cPASvkqgDt2tCvURfBcUCAjZVFcLZcVD5iwXacxvKsxxS/e7kuVV
I1+VGT8Hj8XzWYhjCZxOLZk/fvpYPMyEEujN0fYUp6RtMIXve0C9awwMCy5nBG2J
eE2Ol5DsCpTaBd4Fdr3LWcSs8JFA/YfP9auL3NczOozPoVJt+fw8CBlVIXO0J7l5
hvHDjcC+5v0wxqAlMG6+f/SX7CT8FXK+L3iOJ5gBYUNXqHSxUdv8kt76/KVmQa1B
Akl+MPKpMq+lhw++S3G/lXqwWaDNQbRRw7dSZHymQVXvPp1nsqc3hV7KlOM+6s6g
1g4mvFY4lf6DhptwZLWyQXU8rBQpojvQfiSmDFrFPWFi5BexesuVnkGIolQoklKx
AVUSdJPVEJCteyy7td4FPhBaSqT5vW3+ANbr9b/uoRYWJvn17dN0cc9HuRh/Ai+I
nkfECo2WUDLZ0fEKGjGyFX+todWvJXjvc5kmE9Ty5vJp+M9Vvb8jd6t+mwARAQAB
tCxBV1MgRGVhZGxpbmUgQ2xvdWQgPGF3cy1kZWFkbGluZUBhbWF6b24uY29tPokC
VwQTAQgAQRYhBLhAwIwpqQeWoHH6pfbNPOa3bzzvBQJl+hkLAxsvBAUJA8JnAAUL
CQgHAgIiAgYVCgkICwIDFgIBAh4HAheAAAoJEPbNPOa3bzzvKswQAJXzKSAY8sY8
F6Eas2oYwIDDdDurs8FiEnFghjUEO6MTt9AykF/jw+CQg2UzFtEyObHBymhgmhXE
3buVeom96tgM3ZDfZu+sxi5pGX6oAQnZ6riztN+VpkpQmLgwtMGpSMLl3KLwnv2k
WK8mrR/fPMkfdaewB7A6RIUYiW33GAL4KfMIs8/vIwIJw99NxHpZQVoU6dFpuDtE
1OuxGcCqGJ7mAmo6H/YawSNp2Ns80gyqIKYo7o3LJ+WRroIRlQyctq8gnR9JvYXX
42ASqLq5+OXKo4qh81blXKYqtc176BbbSNFjWnzIQgKDgNiHFZCdcOVgqDhwO15r
NICbqqwwNLj/Fr2kecYx180Ktpl0jOOw5IOyh3bf3MVGWnYRdjvA1v+/CO+55N4g
z0kf50Lcdu5RtqV10XBCifn28pecqPaSdYcssYSRl5DLiFktGbNzTGcZZwITTKQc
af8PPdTGtnnb6P+cdbW3bt9MVtN5/dgSHLThnS8MPEuNCtkTnpXshuVuBGgwBMdb
qUC+HjqvhZzbwns8dr5WI+6HWNBFgGANn6ageYl58vVp0UkuNP8wcWjRARciHXZx
ku6W2jPTHDWGNrBQO2Fx7fd2QYJheIPPAShHcfJO+xgWCof45D0vAxAJ8gGg9Eq+
gFWhsx4NSHn2gh1gDZ41Ou/4exJ1lwPM
=uVaX
-----END PGP PUBLIC KEY BLOCK-----
EOF
```

Linux (AppImage)

验证使用Linux. 的软件包 AppImage 二进制，首先完成Linux选项卡中的步骤 1-3，然后完成以下步骤。

1. 从中的 AppImageUpdate GitHub页面下载 validate-x86_64。 AppImage文件。
2. 下载文件后，要添加执行权限，请运行以下命令。

```
chmod a+x ./validate-x86_64.AppImage
```

3. 要添加执行权限，请运行以下命令。

```
chmod a+x ./deadline-cloud-monitor_<APP_VERSION>_amd64.AppImage
```

4. 要验证 Deadline Cloud 监视器签名，请运行以下命令。

```
./validate-x86_64.AppImage ./deadline-cloud-monitor_<APP_VERSION>_amd64.AppImage
```

如果输出包含短语`Validation successful`，则表示签名已成功通过验证，您可以安全地运行 Deadline Cloud 监视器安装脚本。

Linux (Debian)

要验证使用 Linux .deb 二进制文件的软件包，请先完成选项卡中的Linux步骤 1-3。

dpkg 是大多数debian基础Linux发行版中的核心软件包管理工具。您可以使用该工具验证.deb 文件。

1. 下载 Deadline Cloud monitor.deb 文件：

   下载截止日期云监视器 (.deb)
2. 验证.deb 文件：

```
dpkg-sig --verify deadline-cloud-monitor_amd64.deb
```

3. 输出将类似于：

```
Processing deadline-cloud-monitor_amd64.deb...
GOODSIG _gpgbuilder B840C08C29A90796A071FAA5F6CD3C 171200
```

4. 要验证.deb 文件，请确认输出中`GOODSIG`是否存在。

Linux (RPM)

要验证使用 Linux .rpm 二进制文件的软件包，请先完成Linux选项卡中的步骤 1-3。

1. 下载 Deadline 云监视器.rpm 文件：

   下载截止日期云监视器 (.rpm)

2. 验证.rpm 文件：

```
gpg --export --armor "Deadline Cloud" > key.pub
sudo rpm --import key.pub
rpm -K deadline-cloud-monitor.x86_64.rpm
```

3. 输出将类似于：

```
deadline-cloud-monitor.x86_64.rpm: digests signatures OK
```

4. 要验证.rpm 文件，请确认输出中`digests signatures OK`是否有该文件。

文档历史记录

有关 Deadline Cloud 更新的信息，请参阅 De adline Cloud 发行说明。AWS

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。