aws

用户指南

AWS 大型机现代化

AWS 大型机现代化: 用户指南

Table of Contents

什么是 AWS 大型机现代化？

AWS 大型机现代化可帮助您将大型机应用程序现代化为 AWS 托管运行时环境。它提供了工具和资源来帮助您规划和实施迁移与现代化。您可以分析现有的大型机应用程序，使用 COBOL 或 PL/I 对其进行开发或更新，并实施自动化管道以实现应用程序的持续集成和持续交付 (CI/CD)。您可以选择自动重构模式或更换平台模式，具体取决于您的客户需求。如果您是帮助客户迁移大型机工作负载的顾问，则可以在迁移和现代化之旅的所有阶段（从初始规划到迁移后的云运营）使用 AWS 大型机现代化工具。

您可以使用 AWS 大型机现代化来帮助您高效地创建和管理大型机应用程序 AWS 的运行时环境，以及管理和监控现代化的应用程序。

主题

> **Note**
>
> 在 AWS 大型机现代化项目中，您是否与大型机迁移能力合作伙伴或 AWS 专业服务部门合作？ 如果没有，我们强烈建议您为项目聘请专家。
>
> - AWS 大型机现代化能力合作伙伴
> - AWS Professional Services

AWS 大型机现代化的功能和用例支持演进式现代化方法，这种方法通过提高敏捷性来提供短期收益，并在以后提供大量优化和创新的机会。有关更多信息，请参阅现代化方法。

AWS 大型机现代化的特点

AWS 大型机现代化功能支持以下用例：

- 评估： AWS 大型机现代化的评估能力可以帮助您评估、确定和规划迁移和现代化项目。
- 重构：在 AWS Blu Age 的支持下，您可以使用重构来转换传统应用程序编程语言、创建宏服务或微服务，以及对用户界面 (UI) 和应用程序软件堆栈进行现代化改造。

  AWS Blu Insights 现在可 AWS Management Console 通过单点登录获得。您无需再管理单独的 AWS Blu Insights 凭证。你可以直接从中访问 AWS AWS Blu Age Codebase 和 Transformation Center 功能。 AWS Management Console
- 更换平台：在 Micro Focus Enterprise 解决方案的支持下，您可以移植应用程序，并且无需更改即可重新编译大部分应用程序源代码。
- 开发者 IDE： AWS 大型机现代化提供了按需集成开发环境 (IDE)，因此开发人员可以通过智能编辑和调试、即时代码编译和单元测试更快地编写代码。
- 托管运行时： AWS 大型机现代化托管执行环境持续监控您的集群，通过自我修复计算和自动扩展保持企业工作负载的运行。
- 持续集成和交付 (CI/CD)： AWS 大型机现代化的 CI/CD 功能可帮助应用程序开发团队更频繁、更可靠地交付代码更改，从而加快迁移速度，提高质量，并有助于减少 time-to-market 新业务功能的发布。
- 与其他 AWS 服务的集成： AWS 大型机现代化支持 AWS CloudFormation、 AWS PrivateLink、可重复部署以及 AWS Key Management Service 更高的安全性和合规性。
- 扩大可用性： AWS 大型机现代化现已在美国东部（俄亥俄州）、美国西部（加利福尼亚北部）、亚太地区（孟买）、亚太地区（首尔）、亚太地区（新加坡）、亚太地区（东京）、欧洲（伦敦）和欧洲（巴黎）推出。

有关 AWS 大型机现代化功能的更多信息，请参阅https://aws.amazon.com/mainframe-modernization/features/。

模式

由 AWS Blu Age 支持的自动重构模式侧重于通过将完整的传统应用程序堆栈及其数据层转换为基于 Java 的现代应用程序来加速现代化，同时保持功能等同性。在此自动转换过程中创建的多层应用程序，具有基于 Angular 的前端、支持 API 的 Java 后端和可访问现代数据存储的数据层。重构过程提供了与传统堆栈同等的功能，可提高项目自动化程度来提升速度和质量并降低成本，从而更快地实现业务收益。有关更多信息，请参阅 AWS Mainframe Modernization 自动重构。

由 Micro Focus Enterprise 套件提供支持的更换平台模式侧重于保留应用程序语言、代码和构件，从而最大限度地减少对应用程序资产和团队的影响。它可以帮助客户保持对应用程序知识和技能的使用。

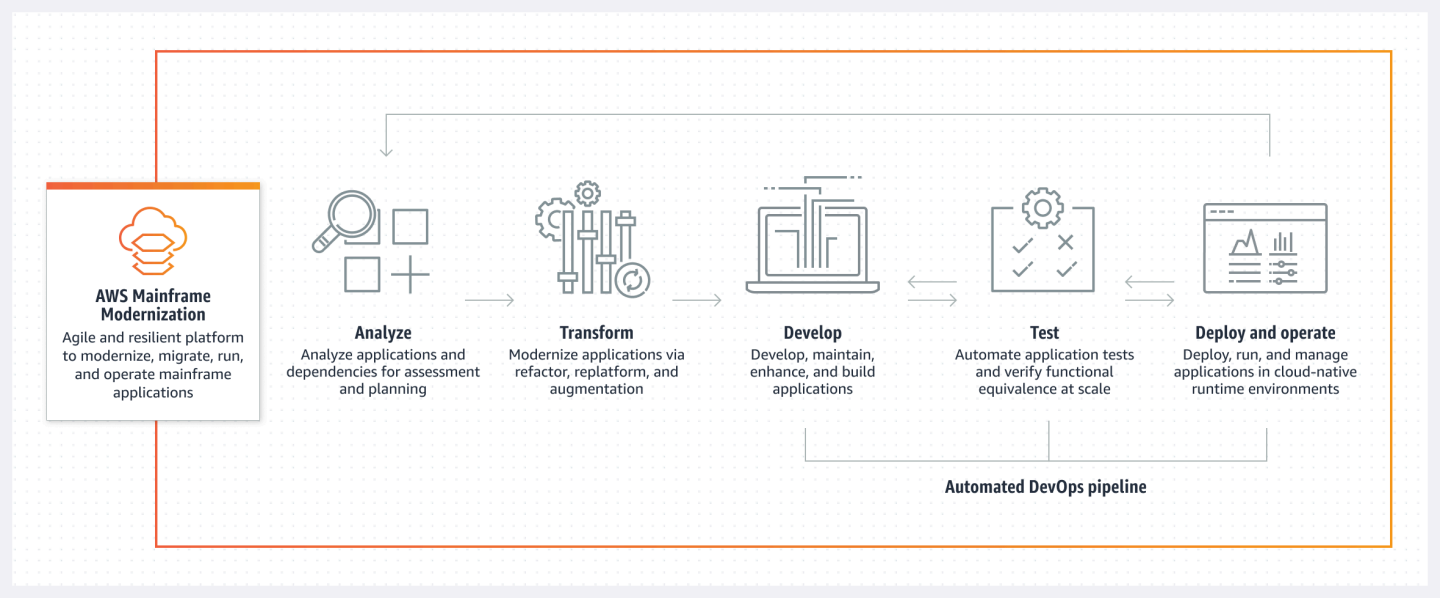
虽然应用程序的更改有限，但这种模式也有助于基础设施和流程的现代化。基础设施更改为基于云的现代托管服务，同时流程更改为遵循应用程序开发和 IT 运营的最佳实操。有关更多信息，请参阅 AWS Mainframe Modernization 更换平台

如何开始进行 AWS 大型机现代化

试试看！我们提供教程和示例应用程序，以帮助您了解 AWS 大型机现代化提供的功能。选择教程：AWS Blu Age 的托管运行时或查看教程：Micro Focus 的托管运行时完整的 step-by-step 教程。

如果您对自动重构感兴趣，请查看 AWS Blu Age 工具，网址为。BluInsights你也可以设置 AppStream 2.0 来访问 AWS Blu Age Developer IDE 或 Micro Focus 企业分析器和 Micro Focus 企业开发者工具。

教程和示例应用程序只能让你了解 AWS 大型机现代化所提供的内容。当您准备启动现代化项目时，请参阅现代化方法，了解有关现代化项目的各阶段和任务的详细信息。



相关服务

除了用于自动重构的 Blu Insights 之外，您还可以在 AWS 大型机现代化中使用以下 AWS 服务。

- Amazon RDS，用于托管您迁移的数据库。
- Amazon S3，用于存储应用程序二进制文件和定义文件。
- Amazon FSx 或 Amazon EFS，用于存储应用程序数据。
- 亚马逊 AppStream 可以访问 Micro Focus 企业分析器和 Micro Focus 企业开发者工具。

- AWS CloudFormation 用于自动 DevOps 管道，您可以使用它来为迁移的应用程序设置 CI/CD。
- AWS Migration Hub
- AWS DMS 用于迁移数据库。

访问 AWS 大型机现代化

目前，你可以通过控制台访问 AWS 大型机现代化，网址为 https://console.aws.amazon.com/m2/。有关提供 AWS 大型机现代化的区域列表，请参阅中的AWS 大型机现代化终端节点和配额。Amazon Web Services 一般参考

您是首次使用 AWS 大型机现代化改造的用户吗？

如果您是首次使用 AWS 大型机现代化的用户，我们建议您先阅读以下章节：

- 开始使用
- 设置

定价

AWS 大型机现代化对支持托管运行时环境的实例的使用收费。此外， AWS 大型机现代化还提供一些工具，无需支付额外费用。您应负责支付与 AWS 大型机现代化相关的其他 AWS 服务所产生的费用。AWS 将在使用 AWS 大型机现代化的任何定价变更生效前 30 天发出通知。有关更多信息，请参阅使用进行大型机现代化。 AWS

使用 AWS Blu Insights，您需要为转型中心的使用付费。有关更多信息，请参阅 AWS Mainframe Modernization 定价。

设置 AWS 大型机现代化

在开始使用 AWS 大型机现代化之前，您或您的管理员需要完成一些步骤。

主题

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

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

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请 为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建管理用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。

2. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建管理用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，为管理用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

作为管理用户登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

AWS 大型机现代化入门

要开始使用 AWS 大型机现代化，您可以按照向您介绍该服务和每个运行时引擎的教程进行操作。

主题

- 教程：AWS Blu Age 的托管运行时
- 教程：Micro Focus 的托管运行时

要继续学习，请参阅以下教程。

- 教程：为 BankDemo 示例应用程序设置 Micro Focus 构建
- 教程：设置 CI/CD 管道以与 Micro Focus Enterprise Developer 搭配使用

教程：AWS Blu Age 的托管运行时

本教程介绍如何将 AWS Blu Age 现代化应用程序部署到 AWS Mainframe Modernization 运行时环境中。

主题

- 先决条件
- 步骤 1：上传演示应用程序
- 步骤 2：创建应用程序定义
- 步骤 3：创建运行时环境
- 步骤 4：创建应用程序
- 步骤 5：部署应用程序
- 步骤 6：启动应用程序
- 步骤 7：访问应用程序
- 步骤 8：测试应用程序
- 清理资源

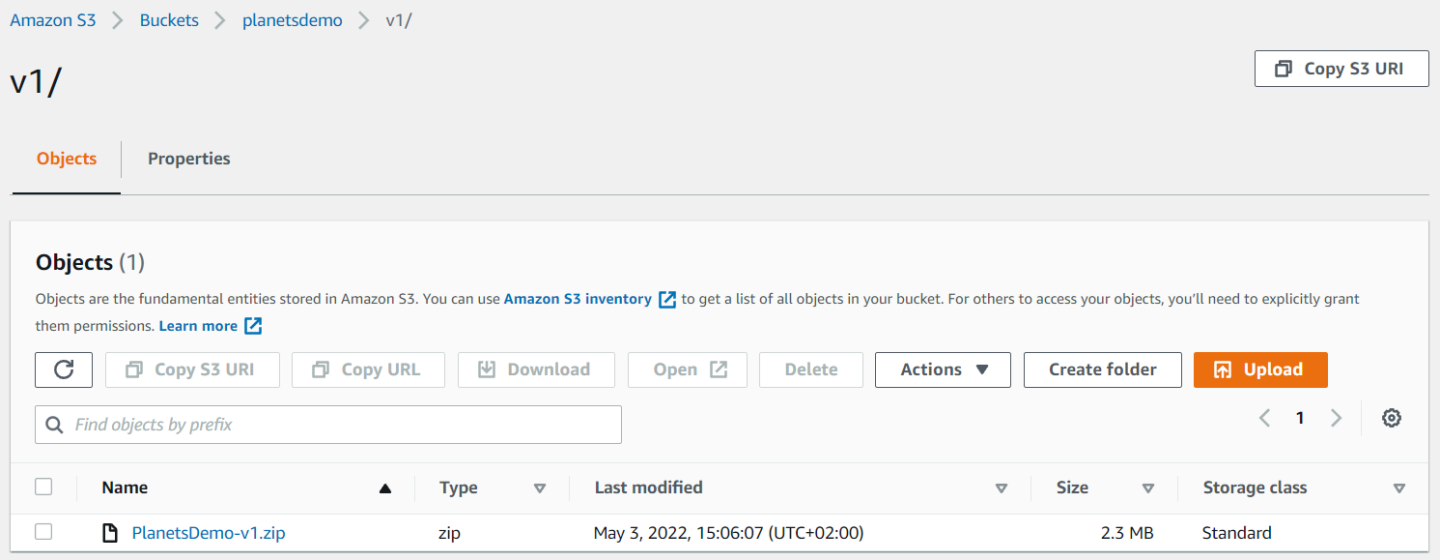
先决条件

要完成本教程，请下载演示应用程序档案 PlanetsDemo-v1.zip。

需要使用现代浏览器才能访问正在运行的演示应用程序。您是在桌面上还是从 Amazon Elastic Compute Cloud 实例（例如，在 VPC 内）运行此浏览器，决定了您的安全设置。

步骤 1：上传演示应用程序

将演示应用程序上传到 Amazon S3 存储桶。请确保此存储桶与您要部署的应用程序位于相同的 AWS 区域。以下示例显示了一个名为 planetsdemo 的存储桶，具有名为 v1 的键前缀或文件夹和名为 `planetsdemo-v1.zip` 的存档。



> **Note**
>
> 存储桶中的文件夹是必需的。

步骤 2：创建应用程序定义

要将应用程序部署到托管运行时，您需要一个 AWS Mainframe Modernization 应用程序定义。此定义是一个描述应用程序位置和设置的 JSON 文件。以下示例是演示应用程序的应用程序定义：

```
{
    "template-version": "2.0",
    "source-locations": [{
        "source-id": "s3-source",
        "source-type": "s3",
        "properties": {
            "s3-bucket": "planetsdemo",
```

```
            "s3-key-prefix": "v1"
        }
    }],
    "definition": {
        "listeners": [{
            "port": 8196,
            "type": "http"
        }],
        "ba-application": {
            "app-location": "${s3-source}/PlanetsDemo-v1.zip"
        }
    }
}
```

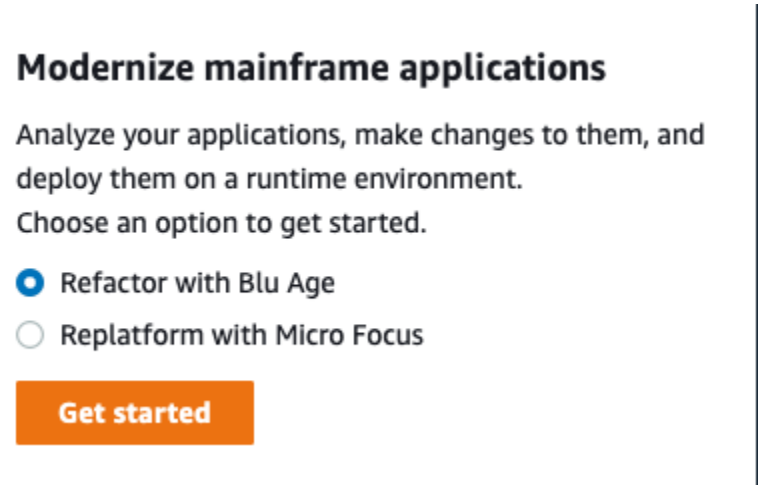
将 s3-bucket 条目更改为存储应用程序示例 zip 文件的存储桶的名称。

有关应用程序定义的更多信息，请参阅 AWS Blu Age 应用程序定义示例。

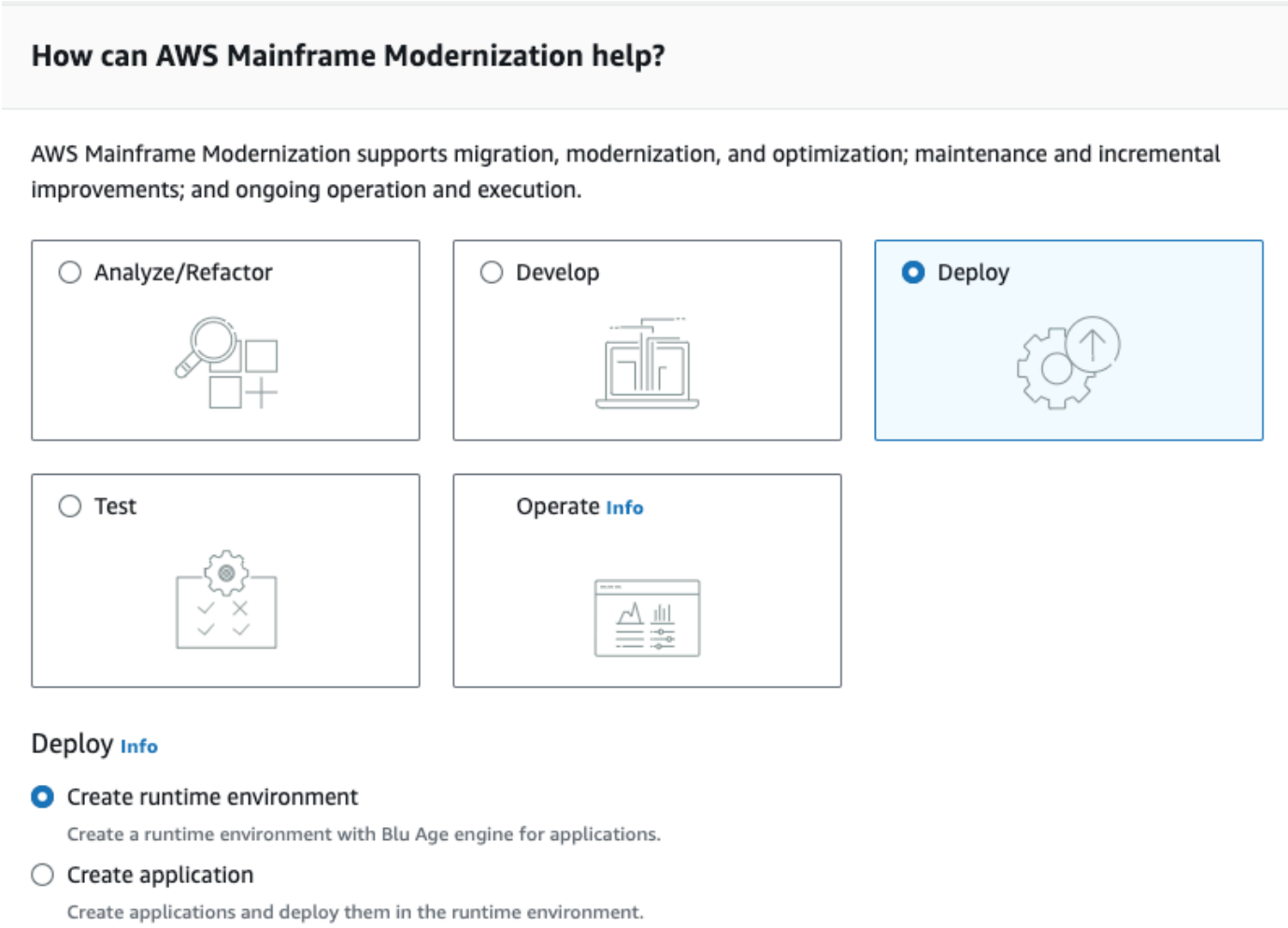
步骤 3：创建运行时环境

要创建 AWS Mainframe Modernization 运行时环境，请执行以下步骤：

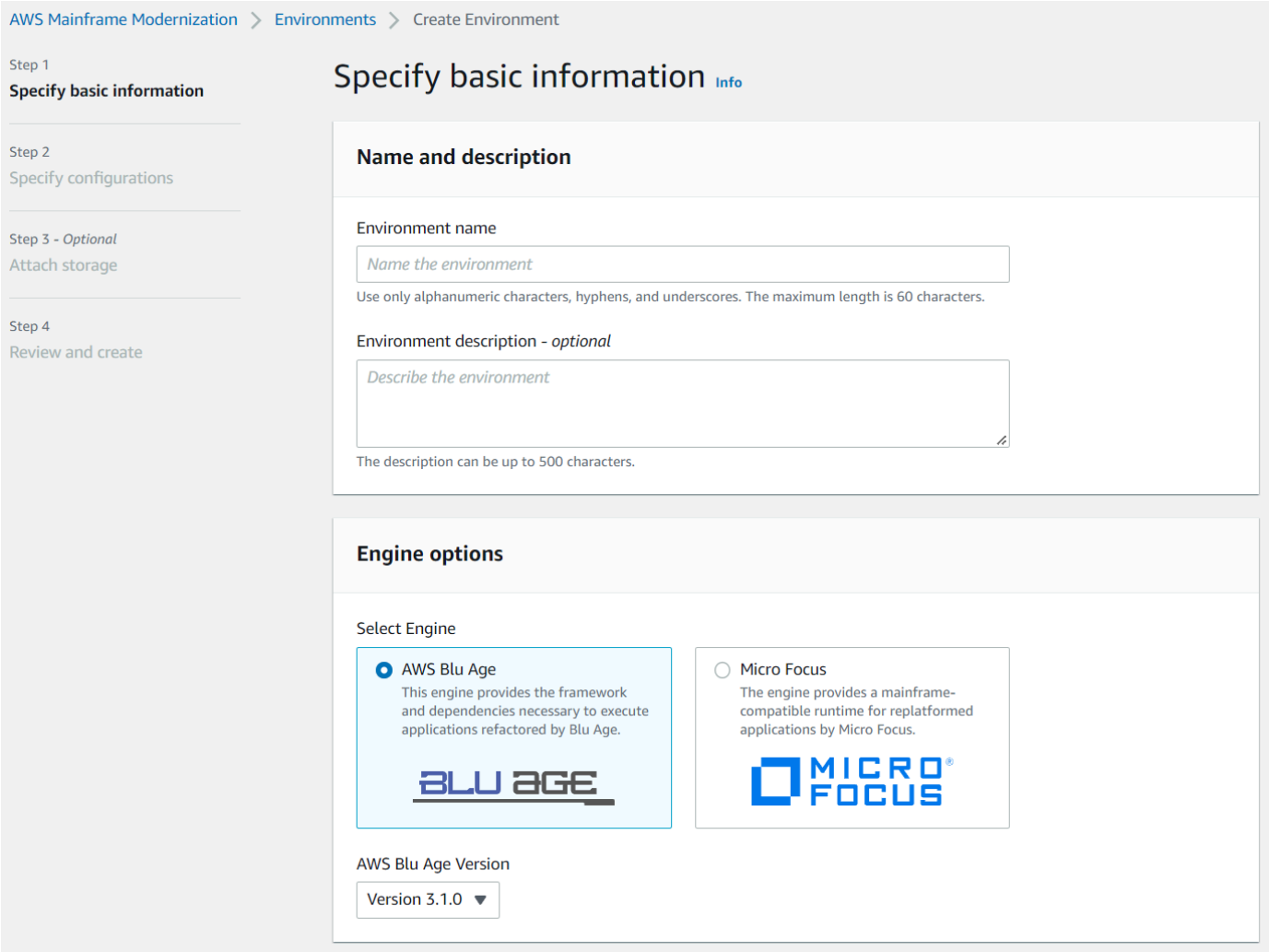
1. 使用 AWS Mainframe Modernization 控制台。
2. 在 AWS 区域选择器中，选择要创建环境的区域。该 AWS 区域必须与在您在步骤 1：上传演示应用程序中创建 S3 存储桶的区域匹配。
3. 在对大型机应用程序进行现代化下，选择使用 Blu Age 重构，然后选择开始。



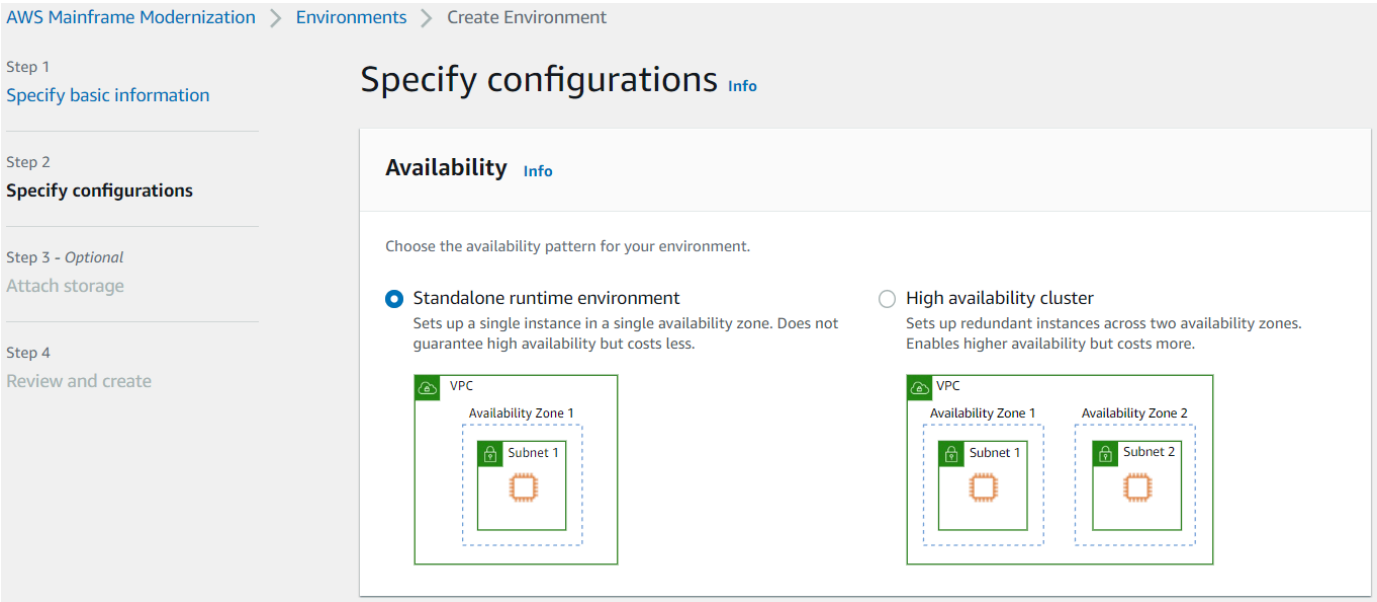
4. 在 AWS Mainframe Modernization 如何帮助您下，选择部署和创建运行时环境。



5. 在左侧导航中，选择环境，然后选择创建环境。在指定基本信息页面上，输入您的环境的名称和描述，然后确保选择 AWS Blu Age 引擎。您可以选择向创建的资源添加标签。然后选择下一步。

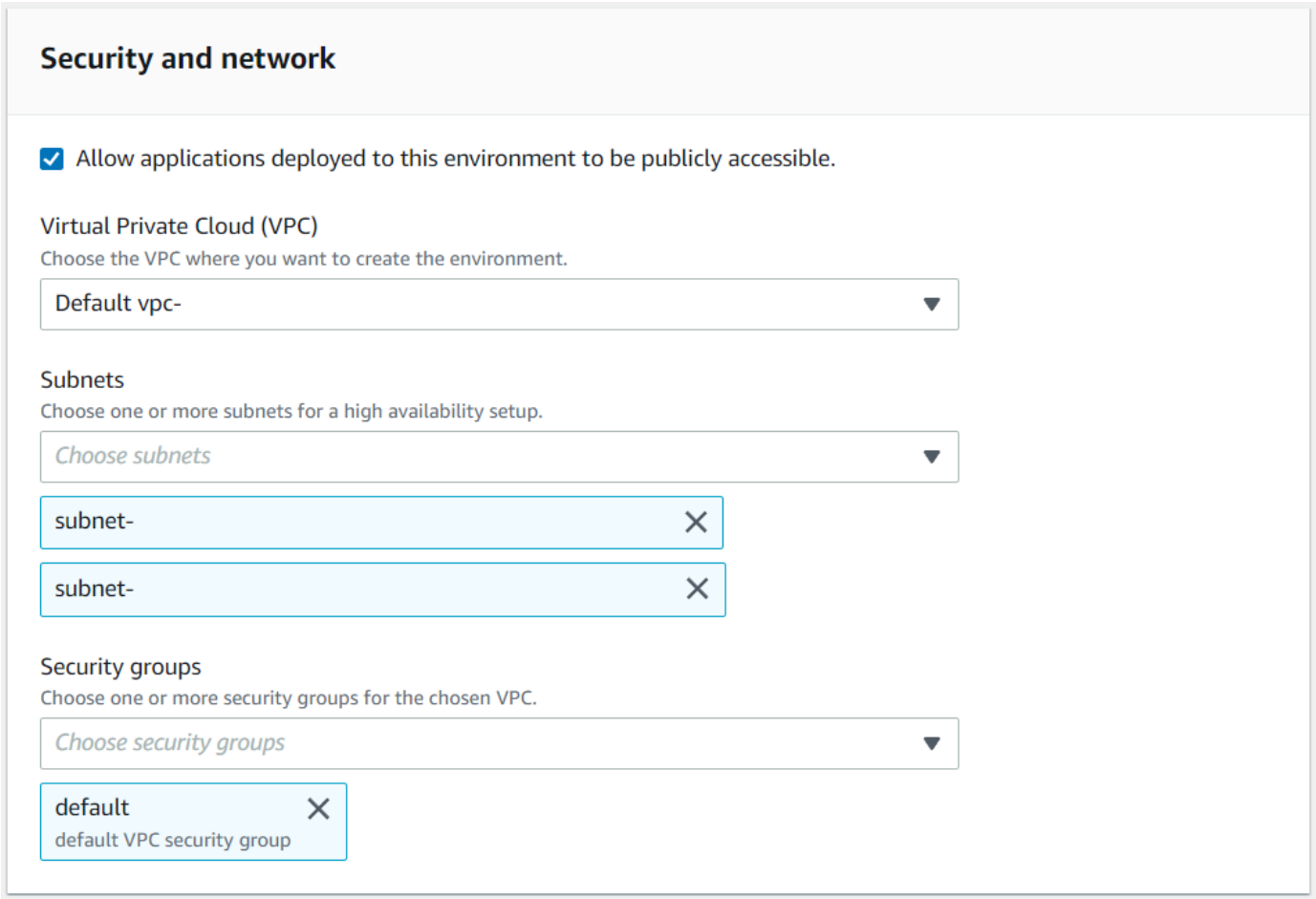


6. 在指定配置页面上，选择独立运行时环境。



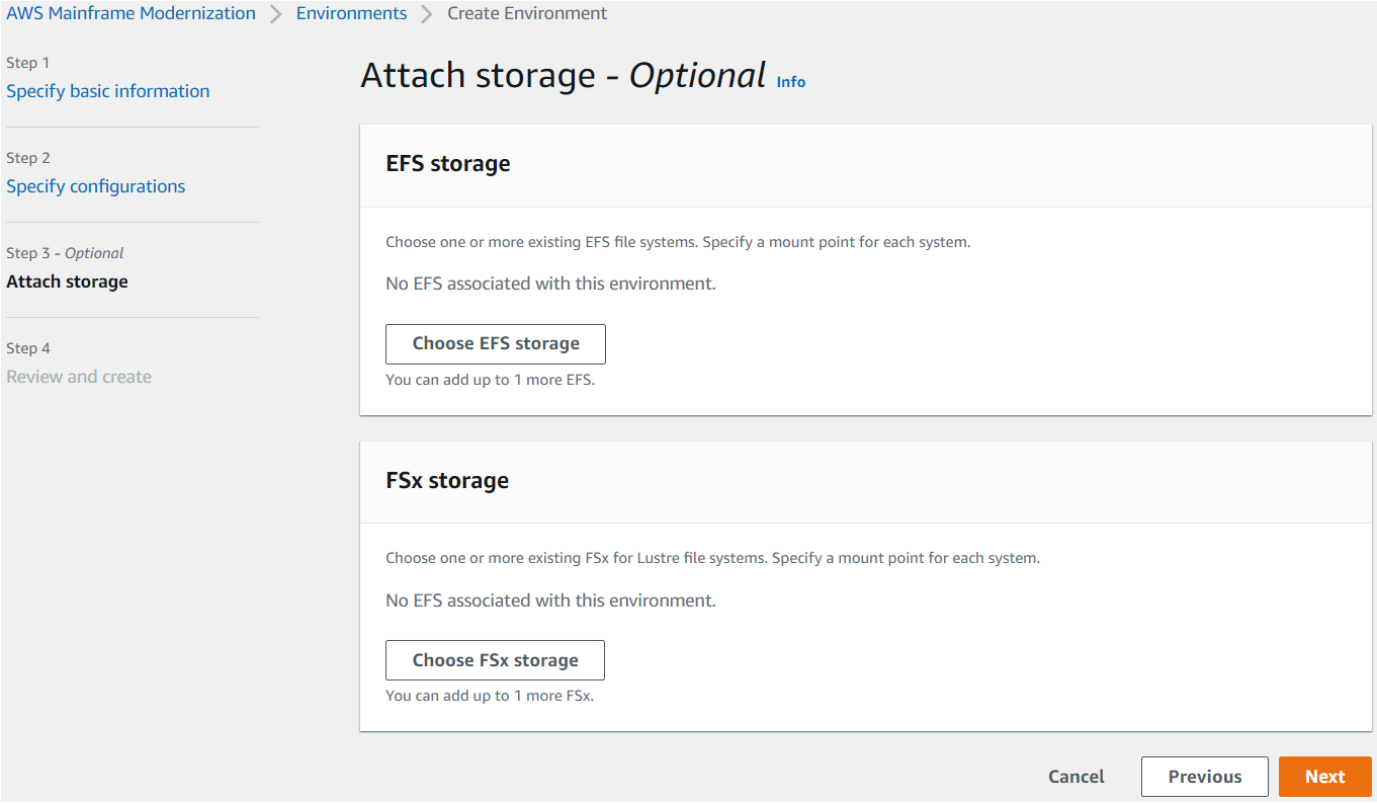
7. 在安全和网络下，进行以下更改：

   - 选择允许部署到此环境的应用程序可公开访问。此选项为应用程序分配一个公有 IP 地址，以便您可以从桌面访问它。
   - 选择 VPC。您可以使用默认值。
   - 选择两个子网。确保子网允许分配公有 IP 地址。
   - 选择安全组。您可以使用默认值。请确保您所选的安全组允许从浏览器 IP 地址访问您在应用程序定义的 `listener` 属性中指定的端口。有关更多信息，请参阅步骤 2：创建应用程序定义。



   如果要从所选 VPC 的外部访问应用程序，请确保已正确配置该 VPC 的入站规则。有关更多信息，请参阅无法访问应用程序的 URL。

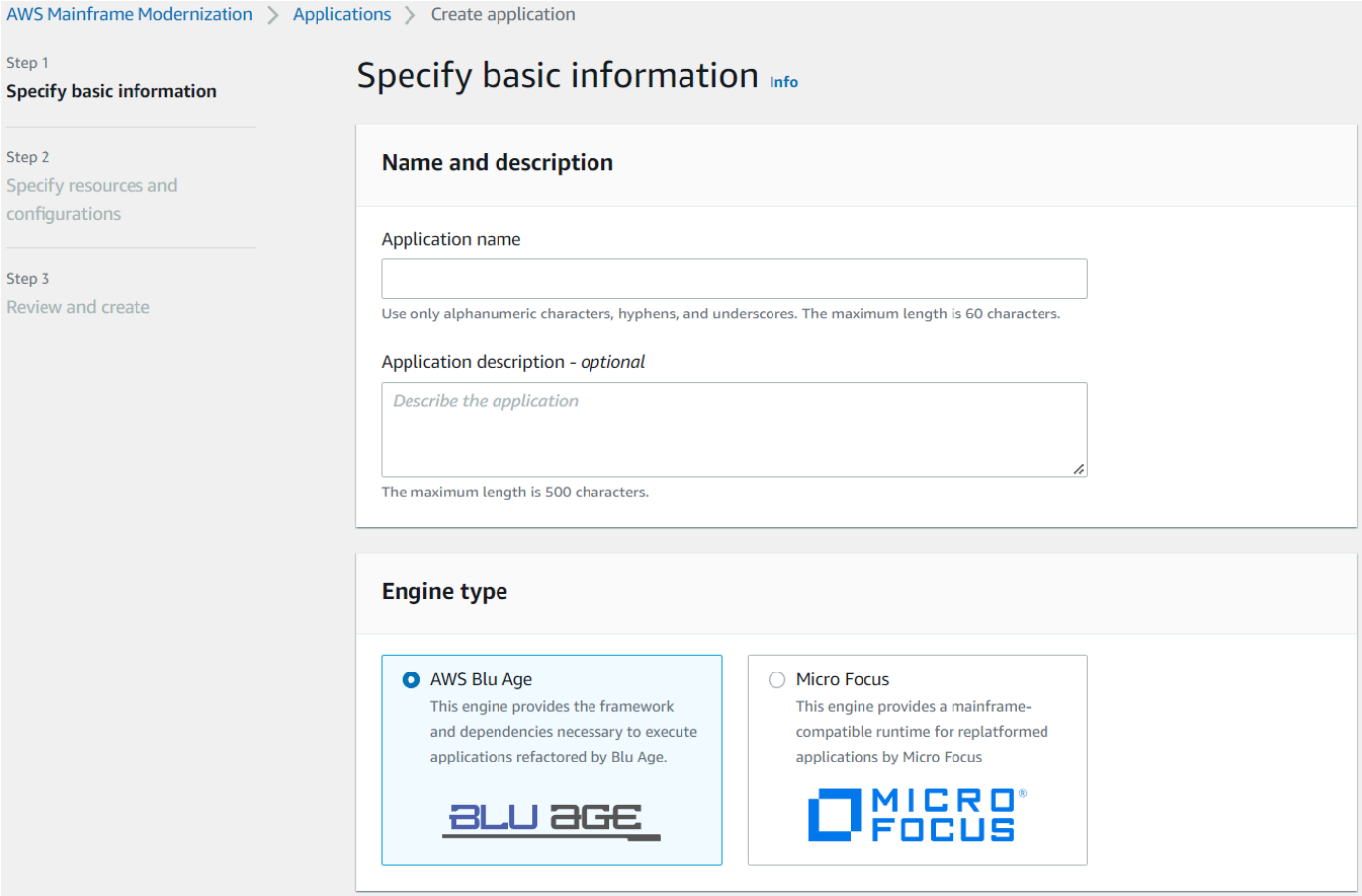
8. 选择 下一步。
9. 在附加存储 – 可选中，保留默认选项，然后选择下一步。



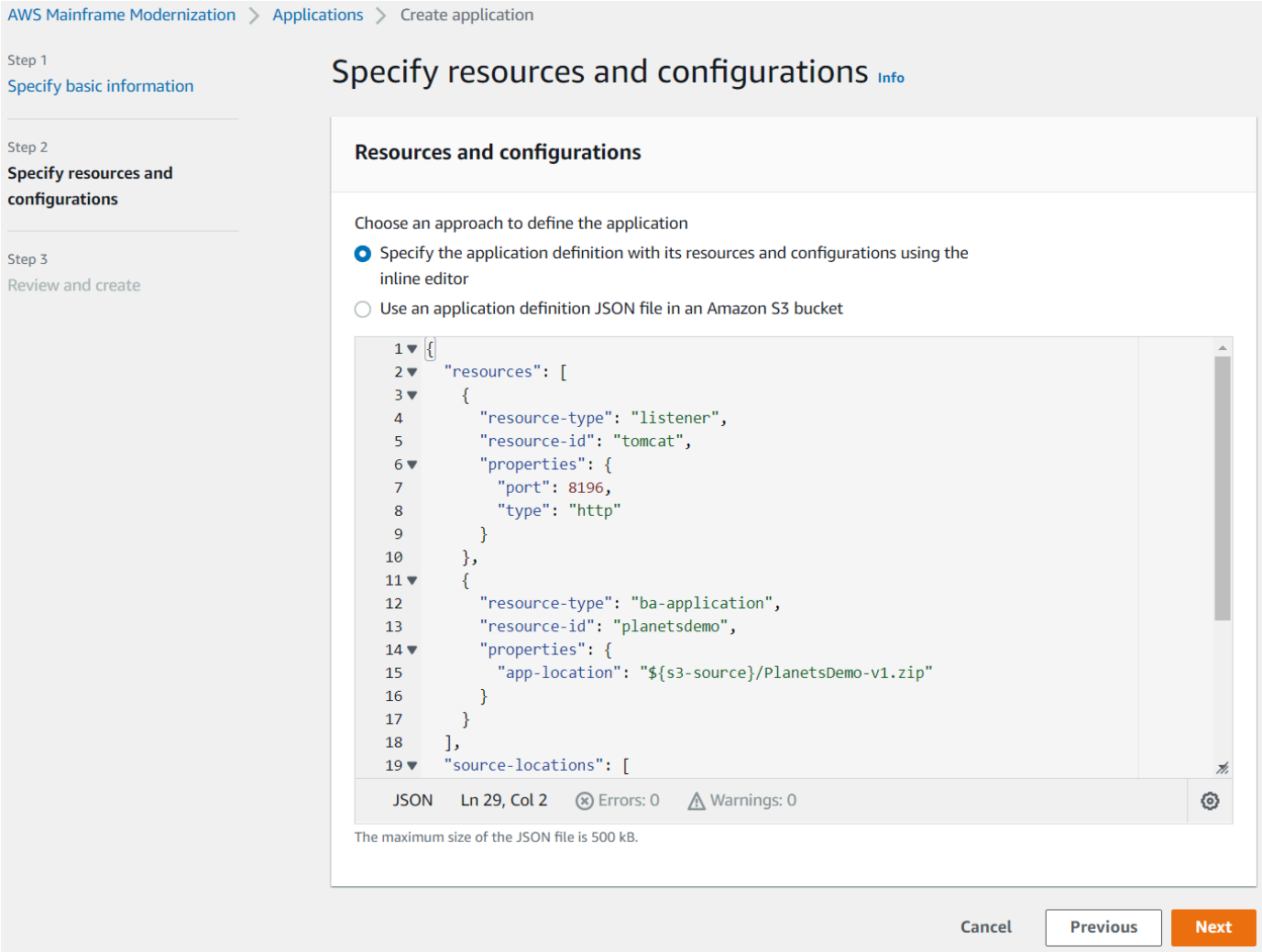
10. 在计划维护中，选择无首选项，然后选择下一步。
11. 在审核并创建中，审核信息，然后选择创建环境。

步骤 4：创建应用程序

1. 在 AWS Management Console 中导航到 AWS Mainframe Modernization。
2. 在导航窗格中，选择应用程序，然后选择创建应用程序。在指定基本信息页面上，输入应用程序的名称和描述，然后确保选择 AWS Blu Age 引擎。然后选择下一步。



3. 在指定资源和配置页面上，复制并粘贴在the section called “步骤 2：创建应用程序定义”中创建的更新的应用程序定义 JSON。

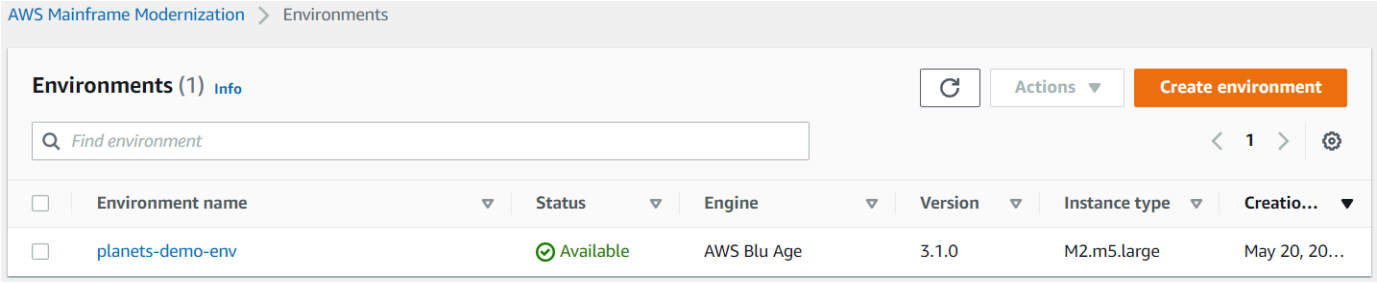


4. 在审核和创建中，审核您的选择，然后选择创建应用程序。

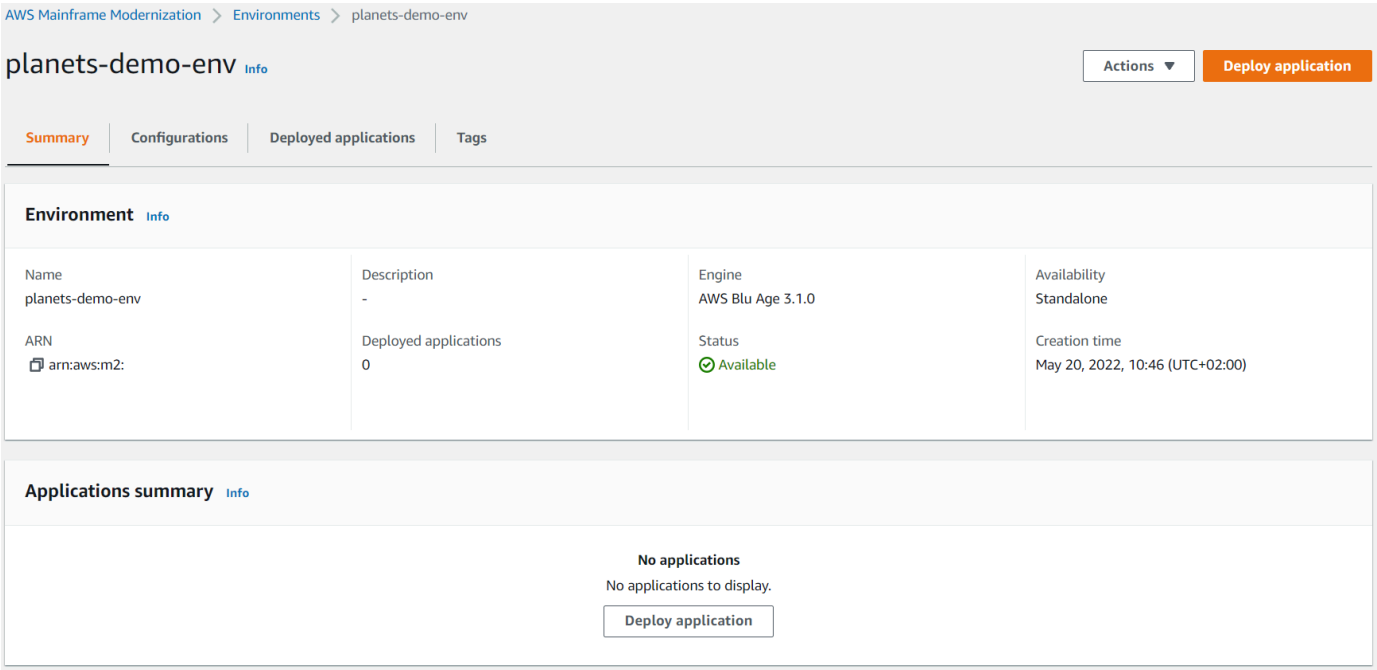
步骤 5：部署应用程序

成功创建 AWS Mainframe Modernization 运行时环境和应用程序，并且两者都处于可用状态后，可以将应用程序部署到运行时环境中。为此，请完成以下步骤：

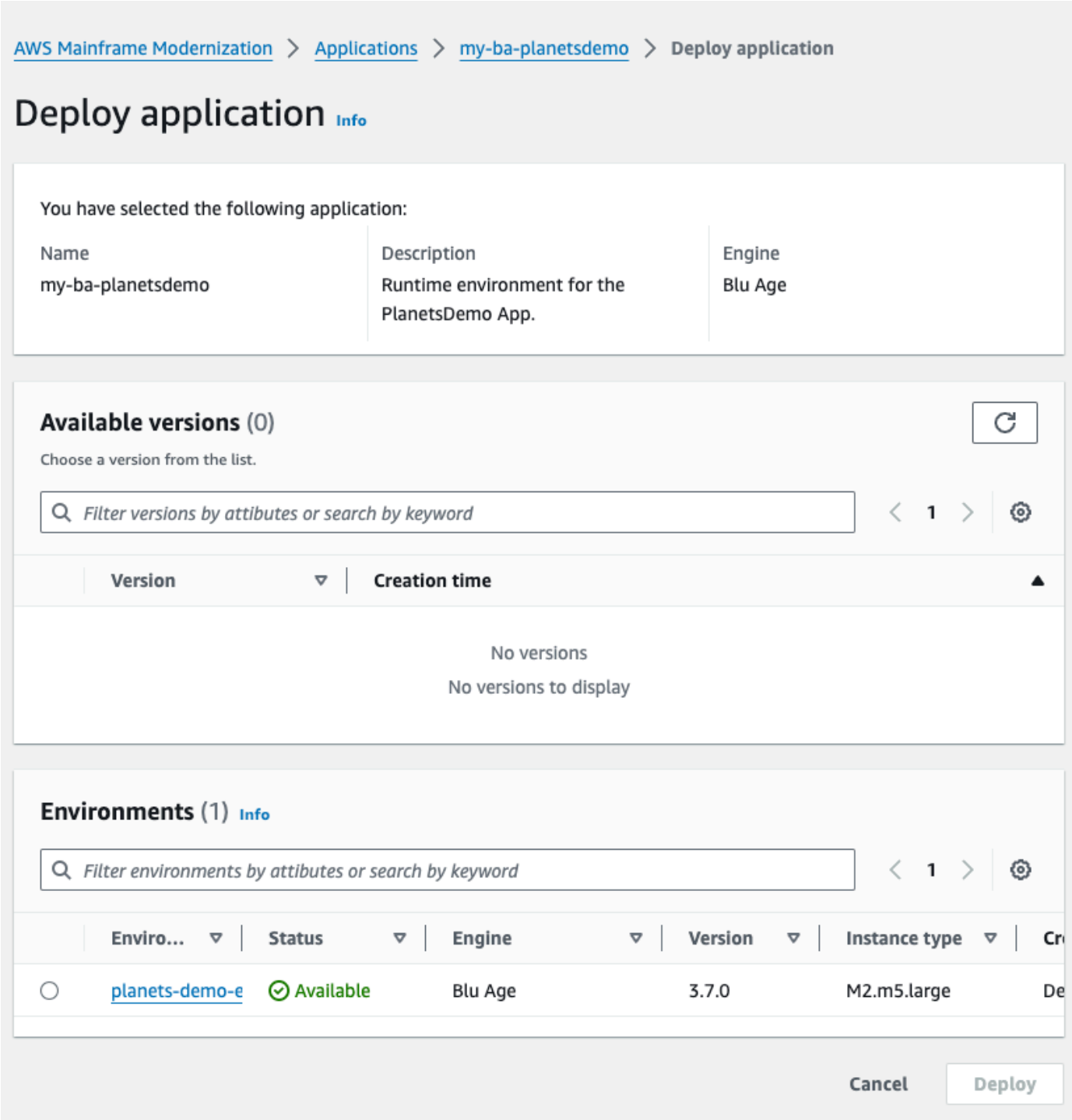
1. 在 AWS 管理控制台中，导航到 AWS Mainframe Modernization。在导航窗格中，选择环境。随后显示环境列表页面。



2. 选择之前创建的运行时环境。随后显示环境详细信息页面。
3. 选择部署应用程序。



4. 选择前面创建的应用程序，然后选择要将应用程序部署到的版本。然后选择部署。

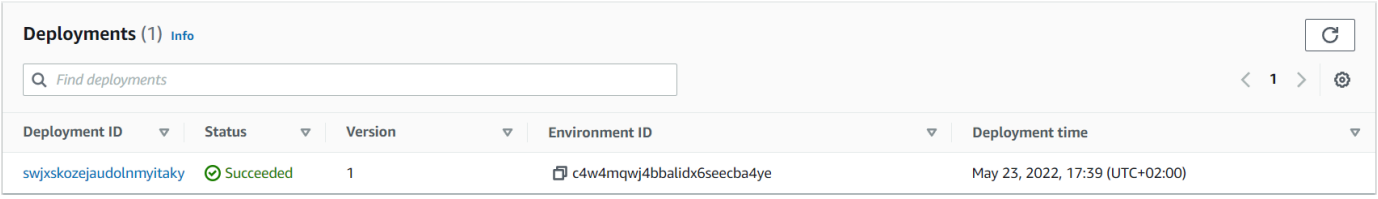


5. 等待直到应用程序完成部署。您将看到一条横幅，带有消息已成功部署应用程序。

步骤 6：启动应用程序

1. 在 AWS Management Console 中，导航到 AWS Mainframe Modernization，然后选择应用程序。

2. 选择您的应用程序，然后转到部署。应用程序的状态应为成功。



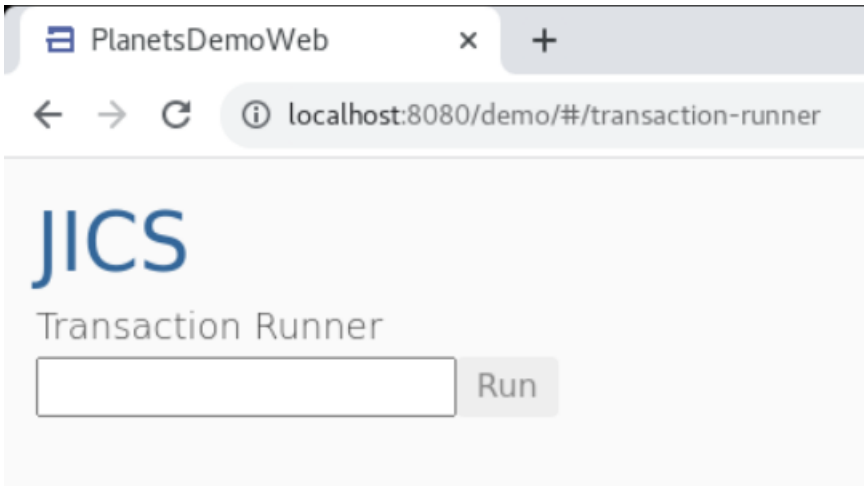
3. 选择操作，然后选择启动应用程序。

步骤 7：访问应用程序

1. 等待直到应用程序处于正在运行状态。您将看到一条横幅，带有消息已成功启动应用程序。
2. 复制应用程序 DNS 主机名。您可以在应用程序的应用程序信息部分中找到此主机名。
3. 在浏览器中，导航到 `http://{hostname}:{portname}/PlanetsDemo-web-1.0.0/`，其中：

   - `hostname` 是之前复制的 DNS 主机名。
   - `portname` 是您在步骤 2：创建应用程序定义中创建的应用程序定义中定义的 Tomcat 端口。

   随后显示 JICS 屏幕。



如果您无法访问应用程序，请参阅无法访问应用程序的 URL。

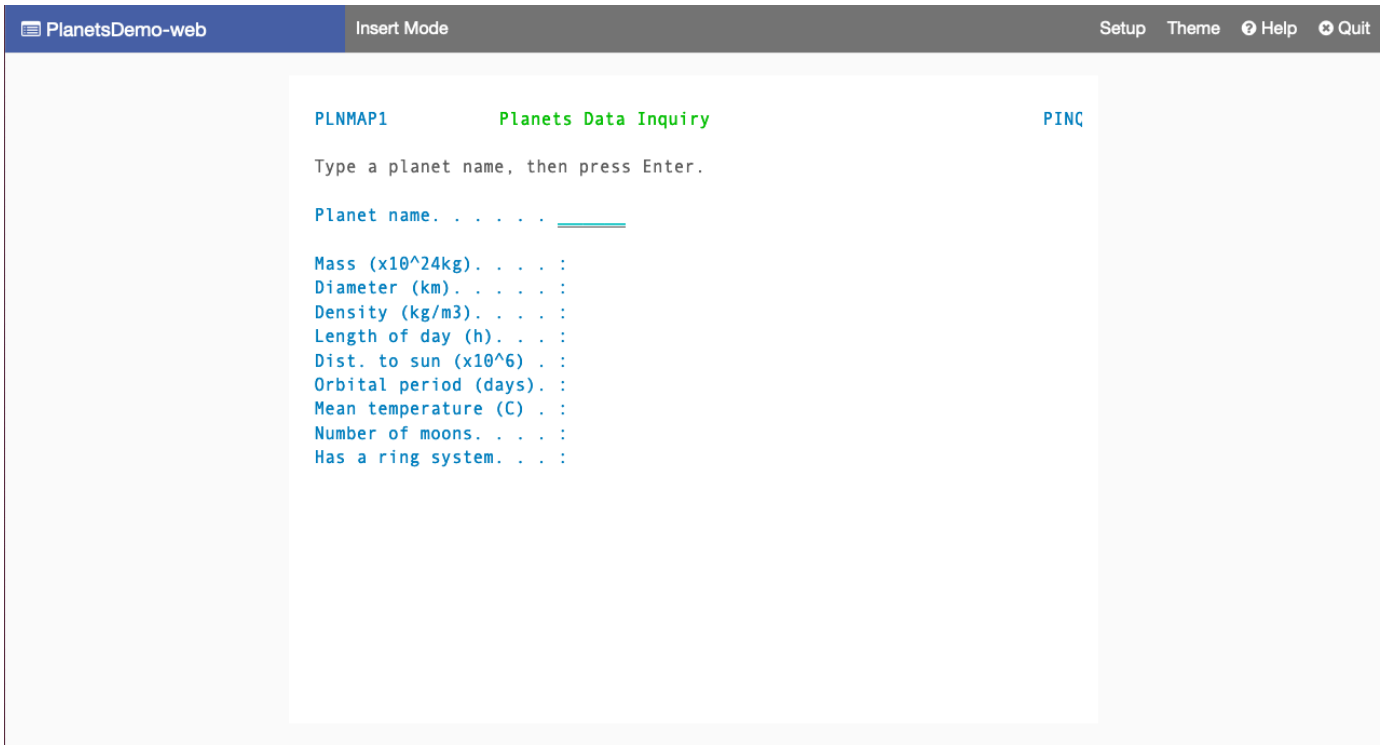
> **Note**
>
> 如果应用程序不可访问，并且端口 8196 上安全组的入站规则选择了“我的 IP”，则指定规则，以允许端口 8196 接受来自 LB i/p 的流量。

步骤 8：测试应用程序

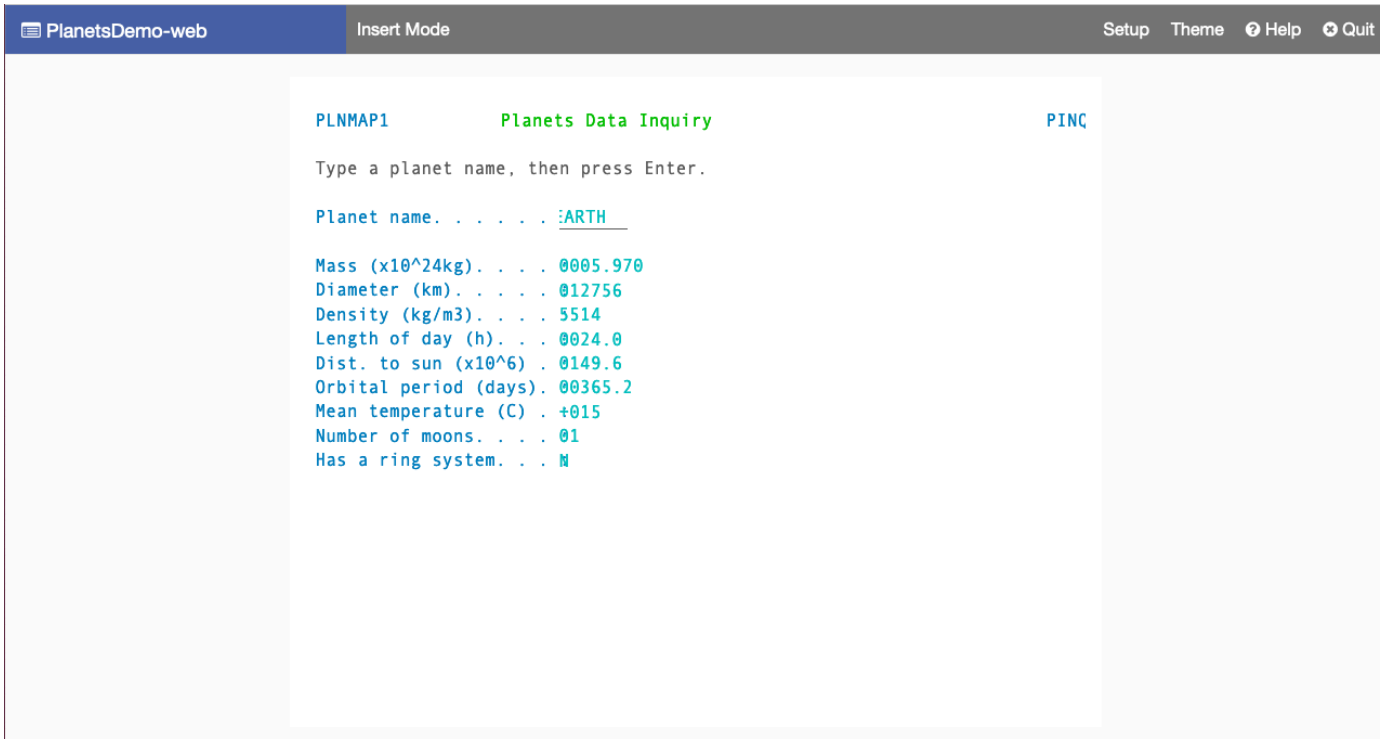
在此步骤中，在迁移的应用程序中运行事务。

1. 在 JICS 屏幕上的输入字段中输入 PINQ，然后选择运行（或按 Enter），以便启动应用程序事务。

   随后会显示演示应用程序屏幕。



2. 在相应的字段中键入行星名称，然后按 Enter。



您可以看到有关该行星的详细信息。

清理资源

如果您不再需要为本教程创建的资源，请将其删除，以免产生额外费用。为此，请完成以下步骤：

- 如果 AWS Mainframe Modernization 应用程序仍在运行，请将其停止。
- 删除应用程序。有关更多信息，请参阅删除 AWS Mainframe Modernization 应用程序。
- 删除运行时环境。有关更多信息，请参阅 删除 AWS Mainframe Modernization 运行时环境。

教程：Micro Focus 的托管运行时

本教程介绍如何使用 Micro Focus 运行时引擎在AWS大型机现代化托管运行时环境中部署和运行 CardDemo 示例应用程序。 CardDemo 示例应用程序是一个简化的信用卡应用程序，旨在测试AWS和展示大型机现代化用例的技术，并与之合作。

在本教程中，您将在其他 AWS 服务中创建资源。其中包括亚马逊简单存储服务、Amazon Relational Database Service 和AWS Secrets Manager。AWS Key Management Service

主题

先决条件

- 确保您可以访问 3270 仿真器以使用 CICS 连接。3270 模拟器可从第三方网站免费试用。或者，你可以启动AWS大型机现代化 AppStream 2.0 Micro Focus 实例并使用 Rumba 3270 模拟器（不免费提供）。

  有关 AppStream 2.0 的信息，请参阅the section called “教程：设置 AppStream 2.0 以与 Enterprise Analyzer 和 Enterprise Developer 搭配使用”。

  > **Note**
  >
  > 创建堆栈时，请选择企业开发人员 (ED) 选项，而不是企业分析器 (EA)。

- 下载CardDemo 示例应用程序并将下载的文件解压缩到任意本地目录。该目录将包含一个名`CardDemo`为的子目录。
- 在您的账户中确定一个 VPC，您可以在其中定义本教程中创建的资源。VPC 需要在至少两个可用区中设置子网。有关亚马逊 VPC 的更多信息，请参阅亚马逊 VPC 的工作原理。

步骤 1：创建并加载 Amazon S3 存储桶

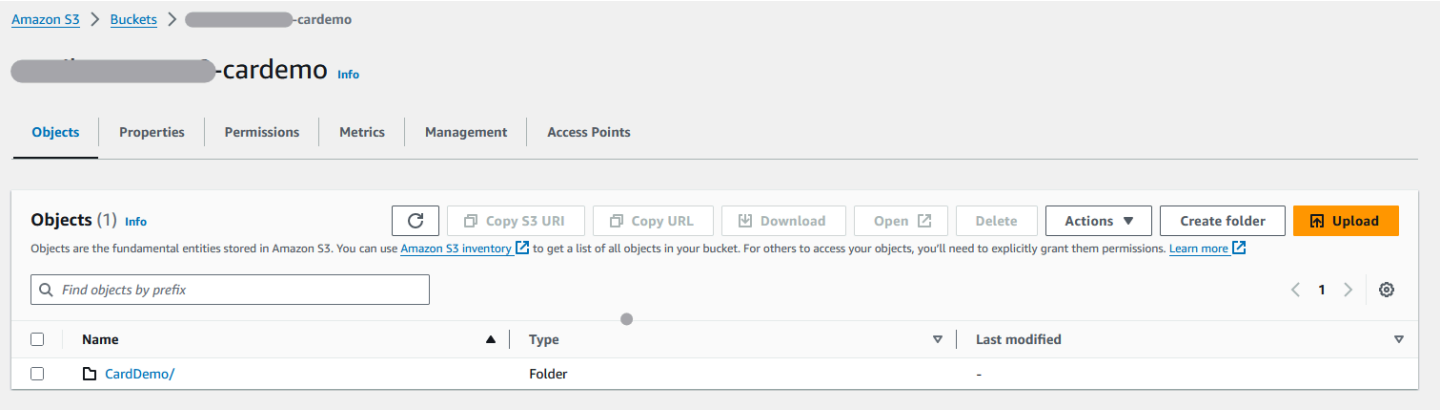
在此步骤中，您将创建一个 Amazon S3 存储桶并将 CardDemo 文件上传到该存储桶。在本教程的后面部分，您将使用这些文件在AWS大型机现代化 Micro Focus 托管运行时环境中部署和运行 CardDemo 示例应用程序。

> **Note**
>
> 您不必创建新的 S3 存储桶，但您选择的存储桶必须与本教程中使用的其他资源位于同一区域。

创建 Amazon S3 存储桶

1. 打开 Amazon S3 控制台，然后选择创建存储桶。
2. 在常规配置中，选择要在其中构建AWS大型机现代化 Micro Focus 托管运行时的 AWS 区域。
3. 输入存储桶名称，例如`yourname-aws-region-carddemo`。保留默认设置，然后选择创建存储桶。或者，您也可以从现有 Amazon S3 存储桶中复制设置，然后选择创建存储桶。
4. 选择您刚刚创建的存储桶，然后选择 Uploa d。
5. 在 “上传” 部分，选择 “添加文件夹”，然后从本地计算机浏览到该`CardDemo`目录。
6. 选择 “上传” 开始上传过程。上传时间因您的连接速度而异。
7. 上传完成后，确认所有文件均已成功上传，然后选择 “关闭”。

您的 Amazon S3 存储桶现在包含该`CardDemo`文件夹。



有关 S3 存储桶的信息，请参阅创建、配置和使用 Amazon S3 存储桶。

步骤 2：创建和配置数据库

在本步骤中，你将在亚马逊关系数据库服务 (Amazon RDS) 中创建一个 PostgreSQL 数据库。在本教程中，该数据库包含 CardDemo 示例应用程序用于客户执行信用卡交易任务的数据集。

在 Amazon RDS 中创建数据库

1. 打开 Amazon RDS 控制台。
2. 选择要在其中创建数据库实例的 AWS 区域。
3. 从导航窗格中选择 Databases（数据库）。
4. 选择 “创建数据库”，然后选择 “标准创建”。
5. 对于引擎类型，选择 PostgreSQL。
6. 选择 15 或更高版本的引擎。

> **Note**
>
> 保存引擎版本，因为本教程稍后需要它。

7. 在模板部分中，选择免费套餐。
8. 将数据库实例标识符更改为有意义的标识符，例如`MicroFocus-Tutorial`。
9. 不要在中管理主证书AWS Secrets Manager。而是输入主密码并进行确认。

> **Note**
>
> 保存您用于数据库的用户名和密码。在本教程的后续步骤中，您将安全地存储它们。

10. 在 “连接” 下，选择要在其中创建AWS大型机现代化托管运行时环境的 VPC。
11. 选择创建数据库。

在 Amazon RDS 中创建自定义参数组

1. 在 Amazon RDS 控制台导航窗格中，选择参数组，然后选择创建参数组。
2. 在 “创建参数组” 窗口中，为参数组系列选择与您的数据库版本相匹配的 Po stgres 选项。

> **Note**
>
> 某些 Postgres 版本需要类型。如果需要，选择数据库参数组。输入参数组的组名和描述。

3. 选择创建。

配置自定义参数组

1. 选择新创建的参数组。
2. 选择操作，然后选择编辑。
3. 筛选`max_prepared_transactions`并将参数值更改为 100。
4. 选择保存更改。

将自定义参数组与数据库关联

1. 在 Amazon RDS 控制台导航窗格中，选择数据库，然后选择要修改的数据库实例。
2. 选择修改。将显示修改数据库实例页面。

> **Note**
>
> 只有在数据库完成创建和备份（这可能需要几分钟时间）之后，“修改” 选项才可用。

3. 在修改数据库实例页面上，导航到其他配置，然后将数据库参数组更改为您的参数组。如果列表中没有您的参数组，请检查该参数组是否使用正确的数据库版本创建。
4. 选择 “继续”，然后查看修改摘要。
5. 选择 “立即应用” 可立即应用更改。
6. 选择修改数据库实例以保存更改。

有关参数组的更多信息，请参阅使用参数组。

> **Note**
>
> 您也可以将 Amazon Aurora PostgreSQL 数据库AWS与大型机现代化配合使用，但没有免费套餐选项。有关更多信息，请参阅使用亚马逊 Aurora PostgreSQL。

步骤 3：创建和配置 AWS KMS key

要安全地存储 Amazon RDS 实例的证书，请先创建一个AWS KMS key。

创建 AWS KMS key

1. 打开密钥管理服务控制台。
2. 选择 Create Key（创建密钥）。
3. 对于密钥类型，保留默认的 Symmetric，对于密钥的使用，保留加密和解密的默认值。
4. 选择下一步。
5. 为密钥指定一个别名，例如`MicroFocus-Tutorial-RDS-Key`和一个可选的描述。
6. 选择下一步。
7. 通过选中您的用户或角色旁边的复选框来分配密钥管理员。
8. 选择 “下一步”，然后再次选择 “下一步”。
9. 在查看屏幕上，编辑密钥策略，然后输入以下内容：

```
{
     "Sid" : "Allow access for Mainframe Modernization Service",
     "Effect" : "Allow",
        "Principal" : {
           "Service" : "m2.amazonaws.com"
                 },
      "Action" : "kms:Decrypt",
      "Resource" : "*"
},
```

   此策略使用此特定密钥策略授予AWS大型机现代化解密权限。
10. 选择 “完成” 以创建密钥。

有关更多信息，请参阅《AWS Key Management Service开发人员指南》中的创建密钥。

步骤 4：创建和配置AWS Secrets Manager数据库密钥

现在使用AWS Secrets Manager和安全地存储数据库凭据AWS KMS key。

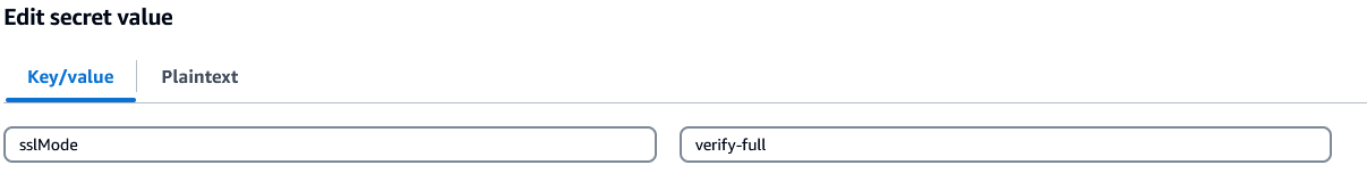
创建和配置AWS Secrets Manager数据库密钥

1. 打开 Secrets Manager 控制台。

2. 在导航窗格中，选择密钥。
3. 在 “密钥” 中，选择 “存储新密钥”。
4. 将密钥类型设置为 Amazon RDS 数据库的凭证。
5. 输入您在创建数据库时指定的凭据。
6. 在加密密钥下，选择您在步骤 3 中创建的密钥。
7. 在 “数据库” 部分，选择您为本教程创建的数据库，然后选择 “下一步”。
8. 在 “密钥名称” 下，输入名称（如）`MicroFocus-Tutorial-RDS-Secret`和可选描述。
9. 在资源权限部分，选择编辑权限，然后将内容替换为以下策略：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect" : "Allow",
            "Principal" : {
                "Service" : "m2.amazonaws.com"
            },
            "Action" : "secretsmanager:GetSecretValue",
            "Resource" : "*"
        }
    ]
}
```

10. 选择保存。
11. 在后续屏幕中选择 “下一步”，然后选择 “存储”。刷新密钥列表以查看新密钥。
12. 选择新创建的密钥并记下，`Secret ARN`因为你需要在本教程的后面部分使用它。
13. 在密钥的概述选项卡中，选择检索密钥值。
14. 选择 “编辑”，然后选择 “添加行”。
15. 为添加一个值**`sslMode`**为的密钥`verify-full`：

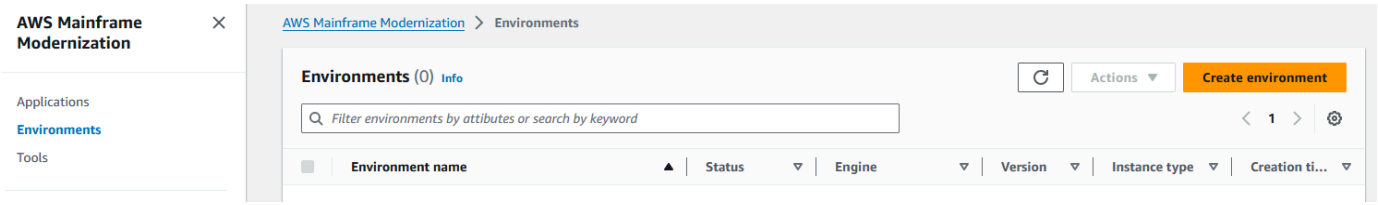


16. 选择保存。

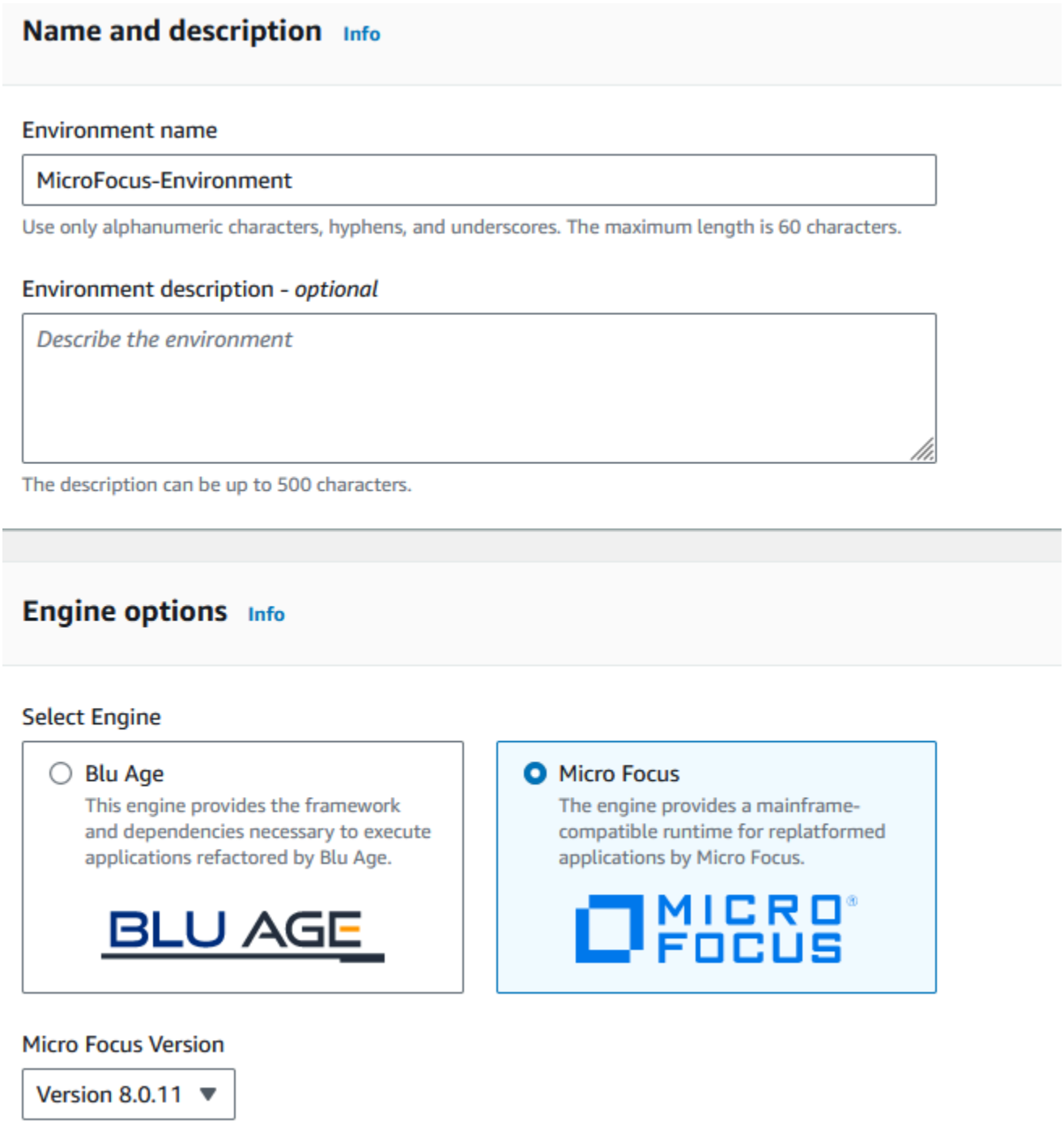
步骤 5：创建运行时环境

创建运行时环境

1. 使用 AWS Mainframe Modernization 控制台。
2. 在导航窗格中，选择环境。然后选择 “创建环境”。



3. 在 “指定基本信息” 下，
   a. 输入`MicroFocus-Environment`环境名称。
   b. 在引擎选项下，确保选择了 Micro Foc us。
   c. 选择最新的 Micro Focus 版本。
   d. 选择下一步。



4. 配置环境

   a. 在可用性下，选择高可用性集群。

   b. 在“资源”下，M2.m5.large为实例类型和所需的实例数量选择M2.c5.large或。最多指定两个实例。

c. 在 “安全和网络” 下，选择 “允许部署到此环境中的应用程序可供公众访问”，然后选择至少两个公有子网。

d. 选择下一步。

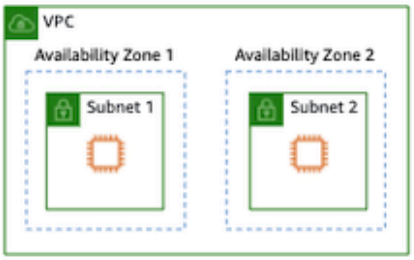
Specify configurations Info

Availability Info

Choose the availability pattern for your environment.

- Standalone runtime environment
  Sets up a single instance in a single availability zone. Does not guarantee high availability but costs less.

- High availability cluster
  Sets up redundant instances across two availability zones. Enables higher availability but costs more.



Resources

Instance type
Choose the instance type for your high availability cluster.

M2.m5.large

Desired capacity
Specify the desired number of instances.

2

Security and network

☑ Allow applications deployed to this environment to be publicly accessible.

Virtual Private Cloud (VPC)
Choose the VPC where you want to create the environment.

Default vpc-15

Subnets
Choose one or more subnets for a high availability setup.

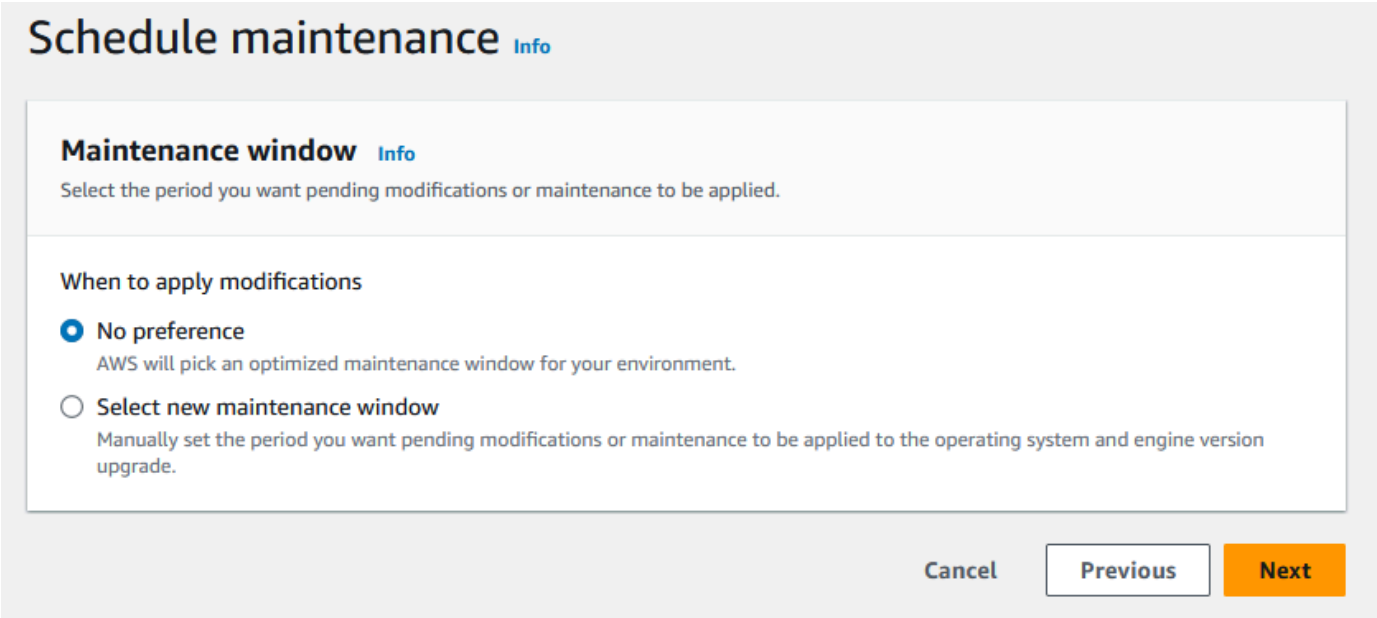
Choose subnets

subnet-56f1a | us-west-2a

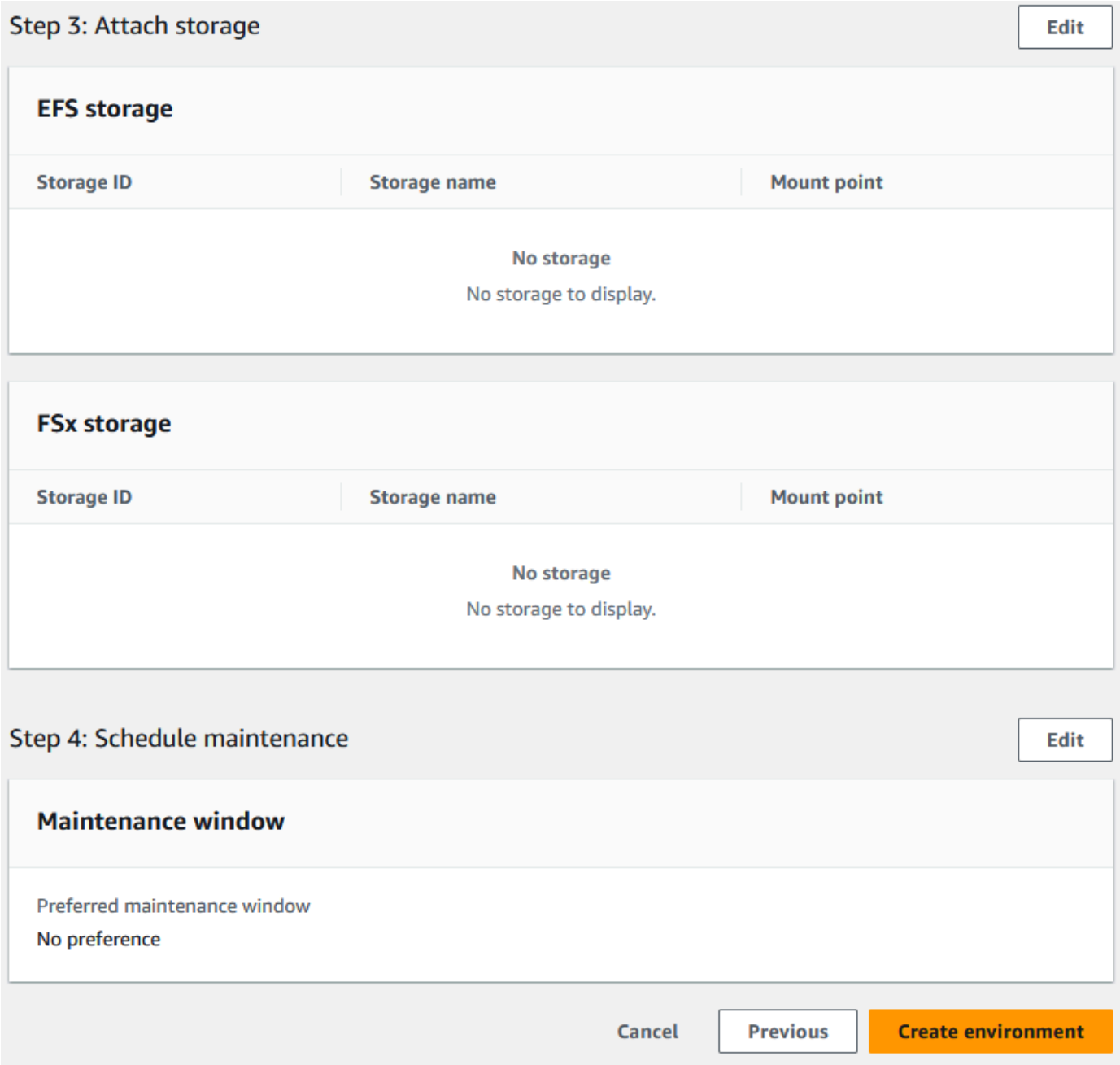
subnet-6685 | us-west-2b

Security groups
Choose one or more security groups for the chosen VPC.

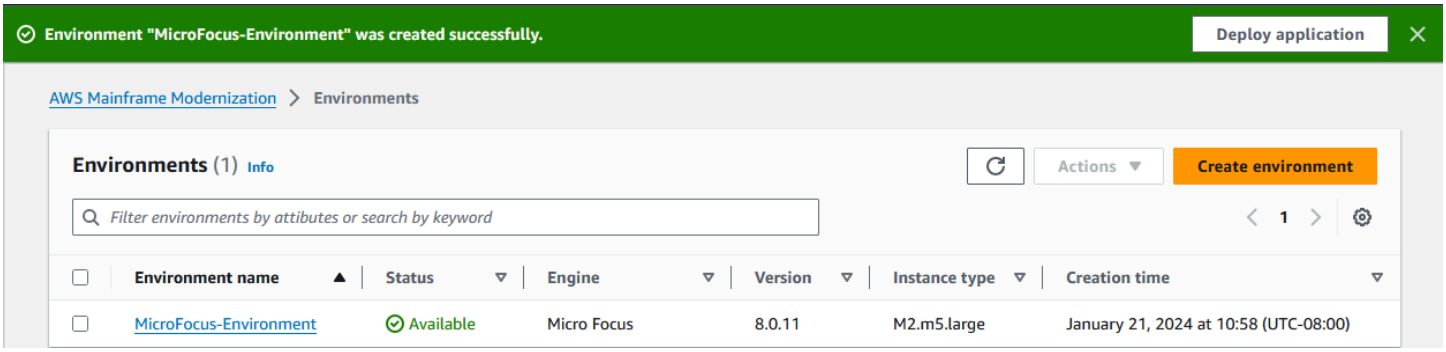
5. 在附加策略页面上，选择下一步。
6. 在 “计划维护” 页面上，选择 “无首选项”，然后选择 “下一步”。



7. 在查看并创建页面上，查看您为运行时环境提供的所有配置，然后选择创建环境。



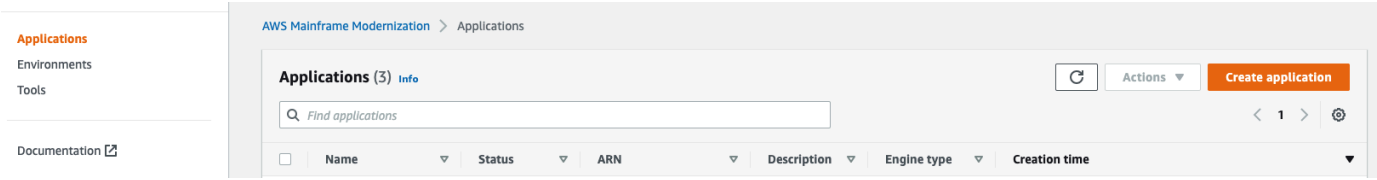
创建环境后，会出现一个横幅，显示`Environment *name* was created successfully`，并且状态字段更改为可用。环境创建过程需要几分钟，但您可以在环境运行时继续执行后续步骤。



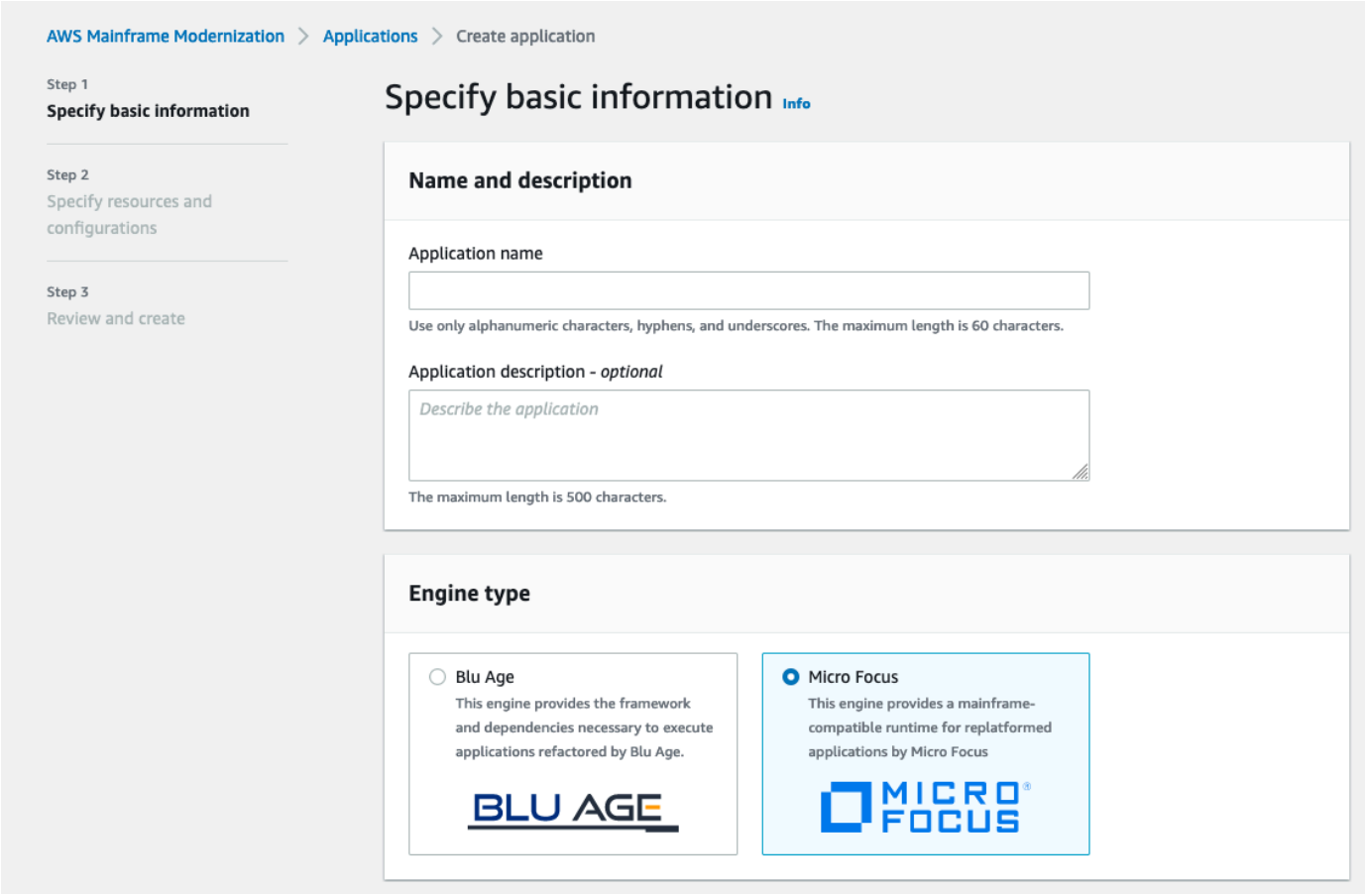
步骤 6：创建应用程序

创建应用程序

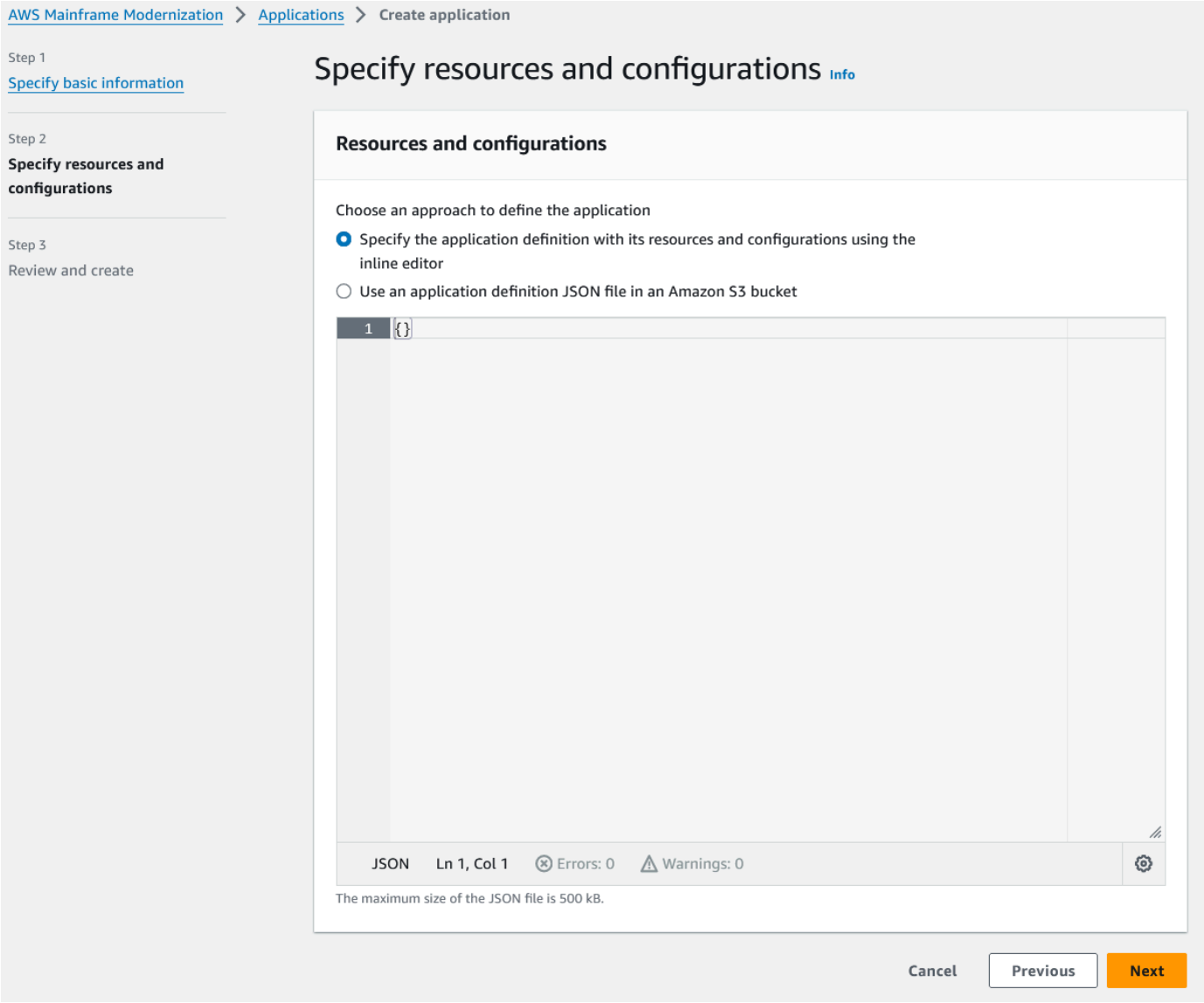
1. 在导航窗格中，选择应用程序。选择创建应用程序。



2. 在 “创建应用程序” 页面的 “指定基本信息” 下，输入`MicroFocus-CardDemo`应用程序名称，然后在 “引擎类型” 下确保选中 Micro Foc us。然后选择下一步。



3. 在“指定资源和配置”下，选择使用内联编辑器指定应用程序定义及其资源和配置的选项。



在编辑器中输入以下应用程序定义：

```
{
  "template-version": "2.0",
  "source-locations": [
    {
      "source-id": "s3-source",
      "source-type": "s3",
      "properties": {
        "s3-bucket": "yourname-aws-region-carddemo",
        "s3-key-prefix": "CardDemo"
      }
    }
```

```
  ],
  "definition": {
    "listeners": [
      {
        "port": 6000,
        "type": "tn3270"
      }
    ],
    "dataset-location": {
      "db-locations": [
        {
          "name": "Database1",
          "secret-manager-arn":
 "arn:aws:secretsmanager:Region:123456789012:secret:MicroFocus-Tutorial-RDS-Secret-
xxxxxx"
        }
      ]
    },
    "batch-settings": {
      "initiators": [
        {
          "classes": [
               "A",
               "B"
            ],
          "description": "initiator_AB...."
        },
        {
          "classes": [
                "C",
                "D"
             ],
          "description": "initiator_CD...."
        }
      ],
      "jcl-file-location": "${s3-source}/catalog/jcl"
    },
    "cics-settings": {
      "binary-file-location": "${s3-source}/loadlib",
      "csd-file-location": "${s3-source}/rdef",
      "system-initialization-table": "CARDSIT"
    },
    "xa-resources": [
      {
```

```
        "name": "XASQL",
        "secret-manager-arn":
 "arn:aws:secretsmanager:Region:123456789012:secret:MicroFocus-Tutorial-RDS-Secret-
xxxxxx",
        "module": "${s3-source}/xa/ESPGSQLXA64.so"
      }
    ]
  }
}
```

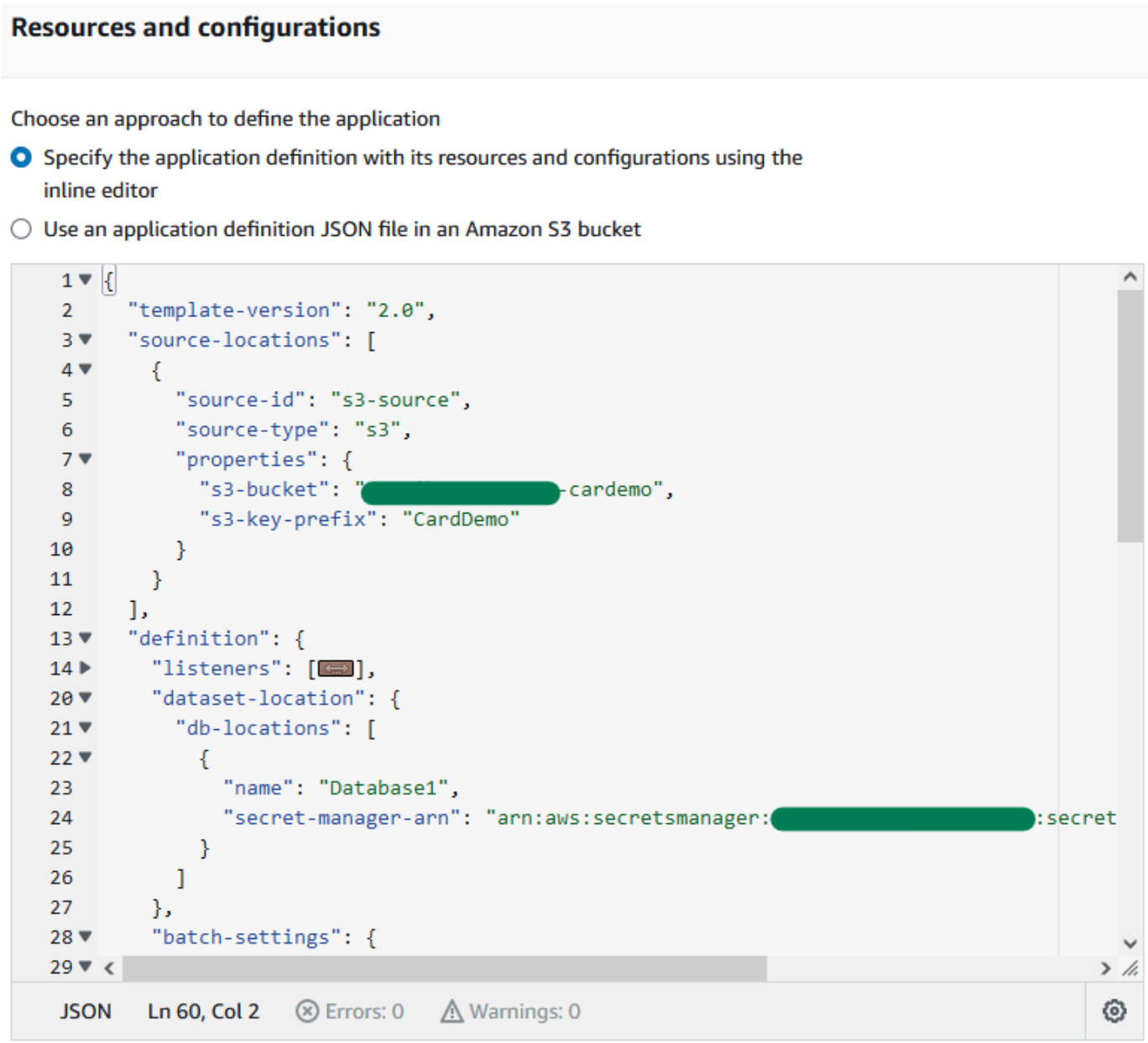
> **Note**
>
> 此文件可能随时更改。

4. 在源位置的属性对象中编辑应用程序 JSON，如下所示：

   a. 将的值替换为您在步骤 1 中创建的 Amazon S3 存储桶的名称。`s3_bucket`

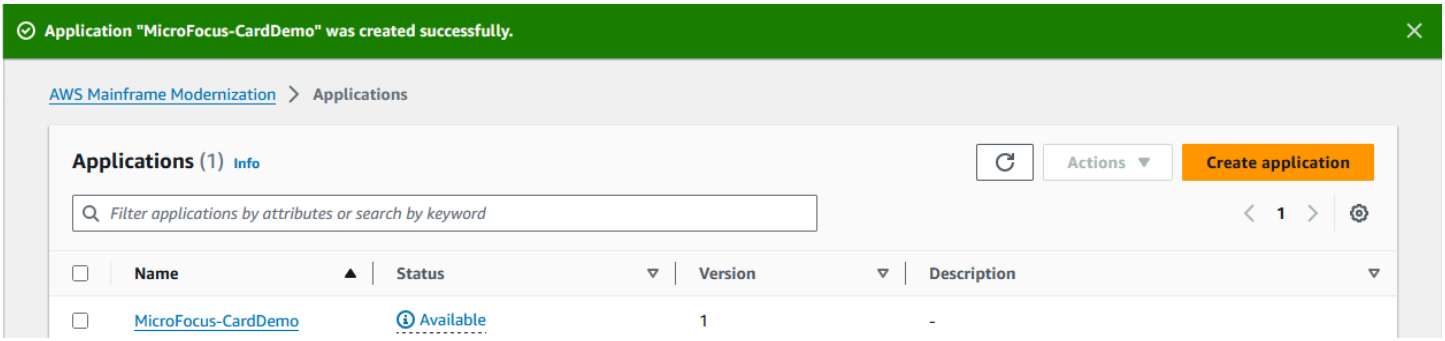
   b. 将的值替换为您上传 CardDemo 示例文件的文件夹（key prefix）。`s3-key-prefix`如果您将`CardDemo`目录直接上传到 Amazon S3 存储桶，则`s3-key-prefix`无需更改。

   c. 将这两个`secret-manager-arn`值替换为您在步骤 4 中创建的数据库密钥的 ARN。



有关应用程序定义的更多信息，请参阅 Micro Focus 应用程序定义。

5. 选择下一步以继续。
6. 在查看并创建页面上，查看您提供的信息，然后选择创建应用程序。

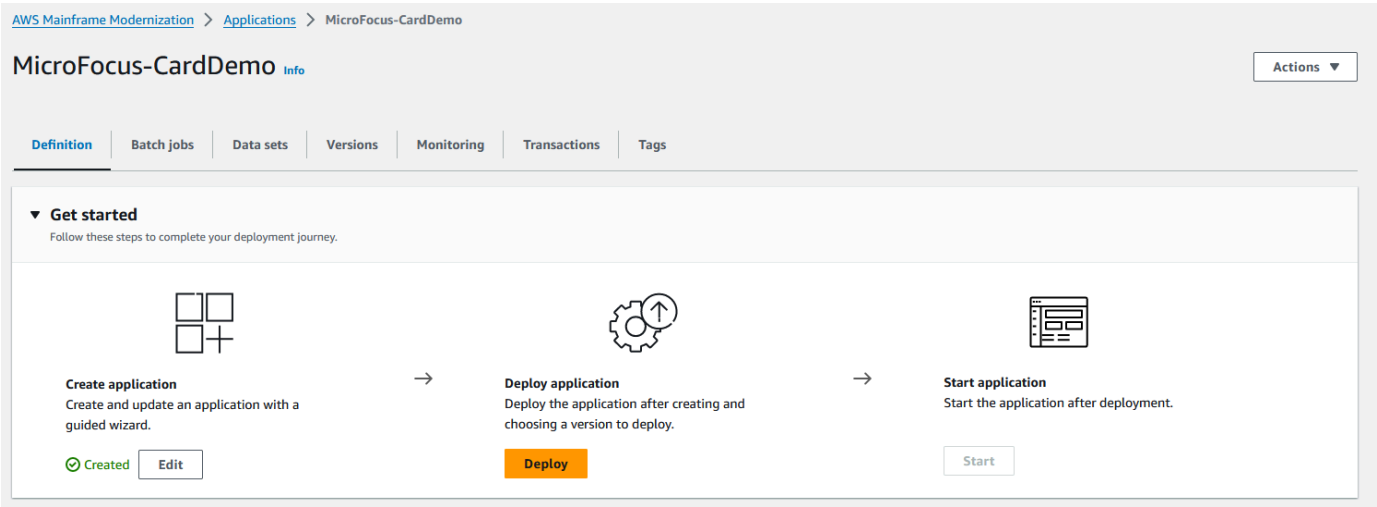


创建应用程序后，会出现一个横幅，上面写着Application *name* was created successfully。并且 “状态” 字段更改为 “可用”。

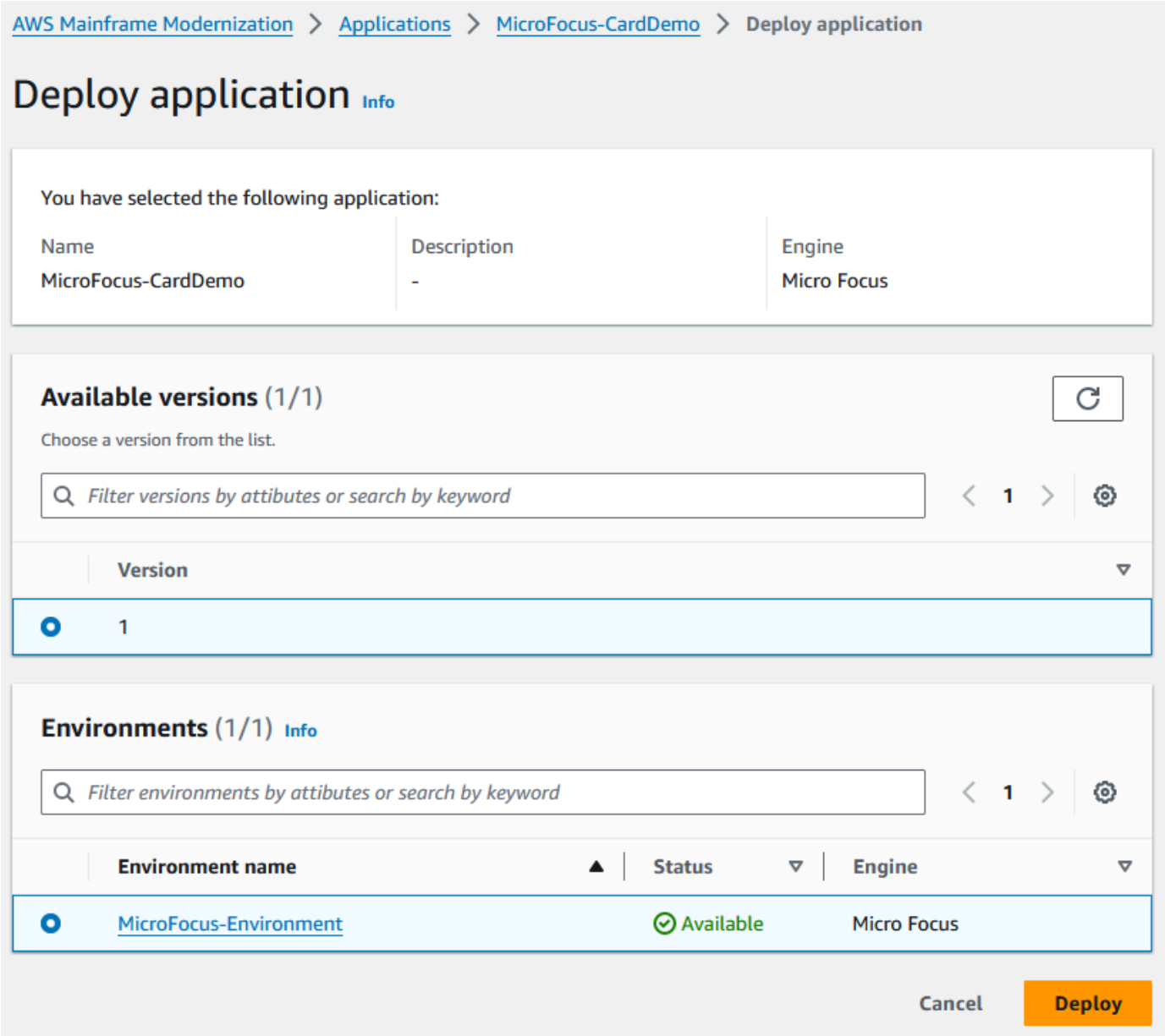
步骤 7：部署应用程序

部署应用程序

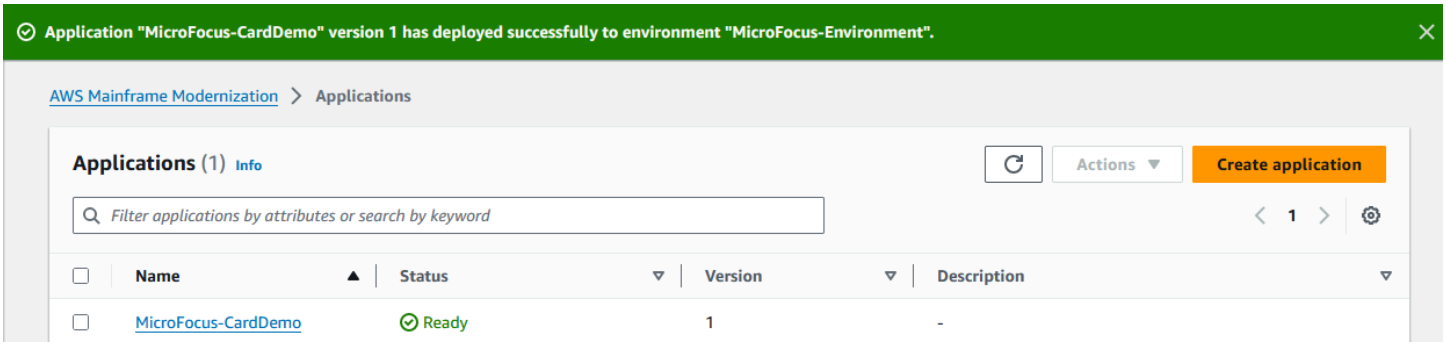
1. 在导航窗格中，选择 “应用程序”，然后选择MicroFocus-CardDemo。
2. 在 “部署应用程序” 下，选择 “部署”。



3. 选择您之前创建的应用程序和环境的最新版本，然后选择 Deploy。



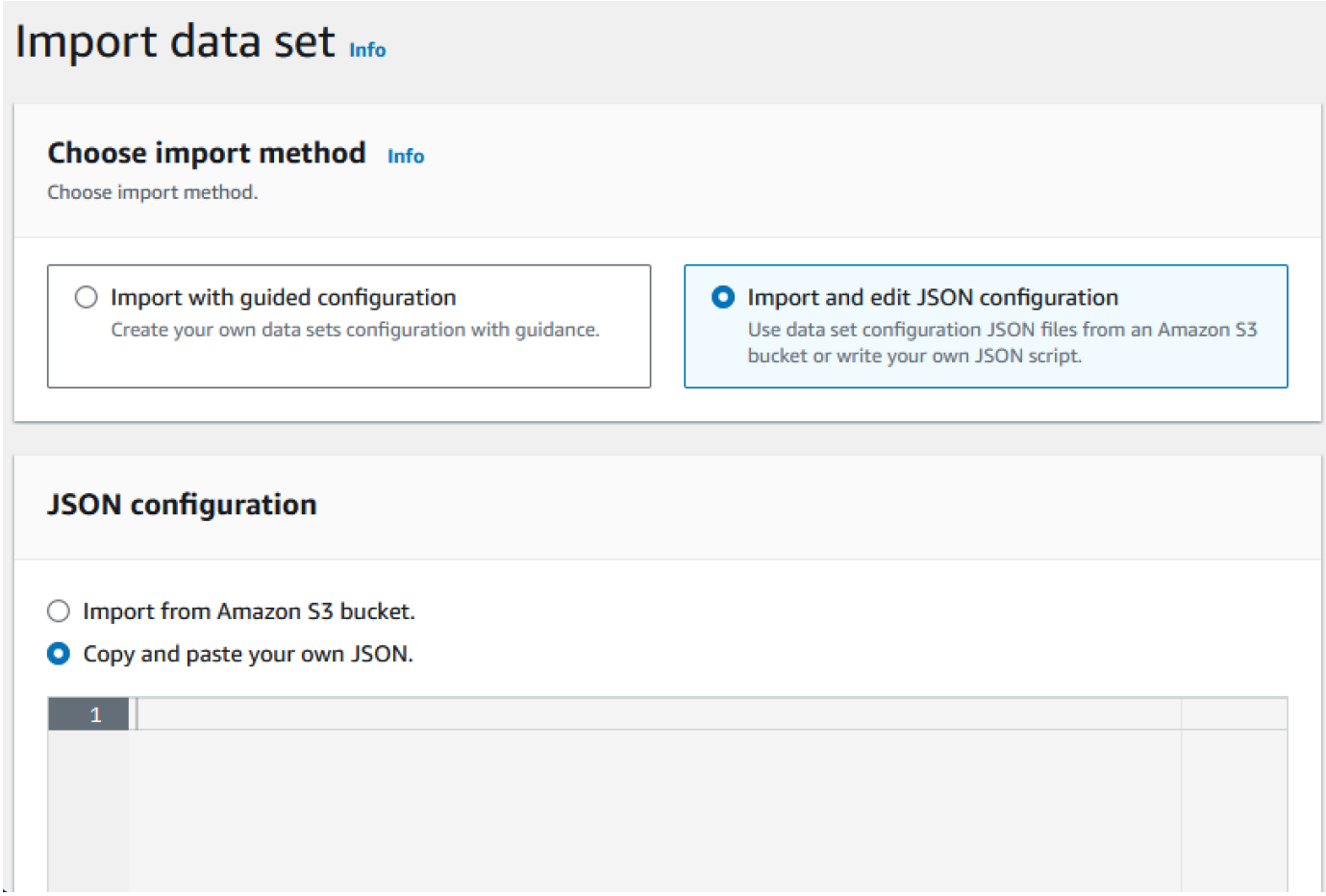
成功部署 CardDemo 应用程序后，状态将更改为 “就绪”。



步骤 8：导入数据集

导入数据集

1. 在导航窗格中，选择应用程序，然后选择应用程序。
2. 选择数据集选项卡。然后选择导入。
3. 选择 “导入和编辑 JSON 配置”，然后选择 “复制并粘贴您自己的 JSON” 选项。



4. 复制并粘贴以下 JSON，但暂时不要选择 “提交”。此 JSON 包含演示应用程序所需的所有数据集，但需要您的 Amazon S3 存储桶详细信息。

```
{
    "dataSets": [
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
```

```
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 11,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 300,
                    "max": 300
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX.PATH",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 11,
                            "offset": 16
                        }
                    }
                },
                "recordLength": {
                    "min": 150,
                    "max": 150
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS.DAT"
            }
        },
```

```
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 16,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 150,
                    "max": 150
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 16,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 50,
                    "max": 50
                }
```

```
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 9,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 500,
                    "max": 500
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 11,
                            "offset": 25
                        }
```

```
                    }
                },
                "recordLength": {
                    "min": 50,
                    "max": 50
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 16,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 350,
                    "max": 350
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
```

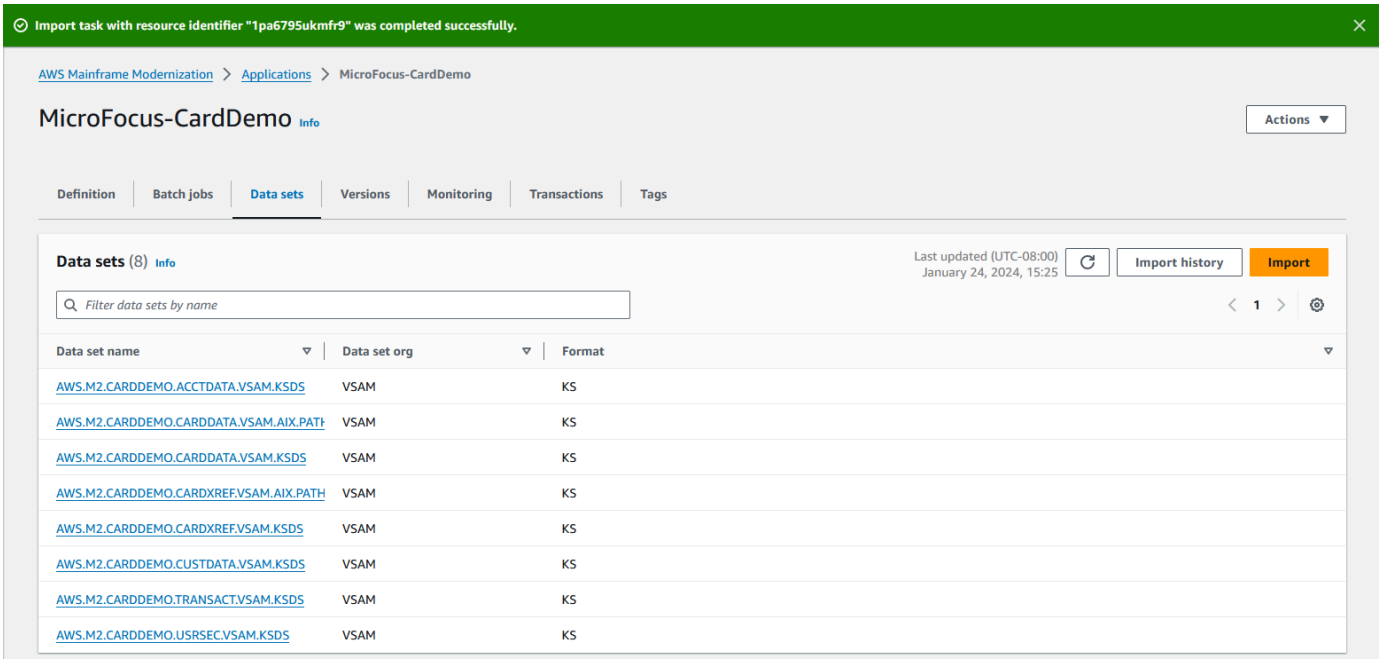
```
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 8,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 80,
                    "max": 80
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS.DAT"
            }
        }
    ]
}
```

5. 将每次出现的<s3-bucket-name>（有八个）替换为包含该 CardDemo 文件夹的 Amazon S3 存储桶的名称，例如your-name-aws-region-carddemo。

> **Note**
>
> 要复制亚马逊 S3 中该文件夹的 Amazon S3 URI，请选择该文件夹，然后选择 “复制亚马逊 S3 URI”。

6. 选择提交。

导入完成后，将出现一条横幅，上面写着以下消息：Import task with resource identifier *name* was completed successfully.将显示导入的数据集列表。

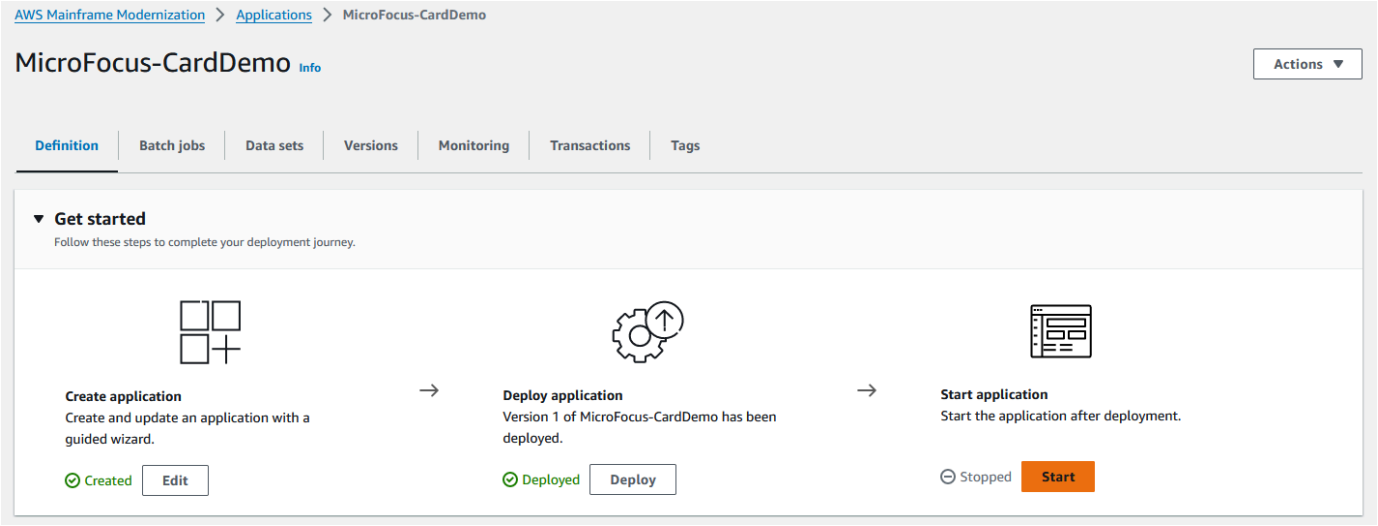


您还可以通过选择数据集选项卡上的导入历史记录来查看所有数据集导入的状态。

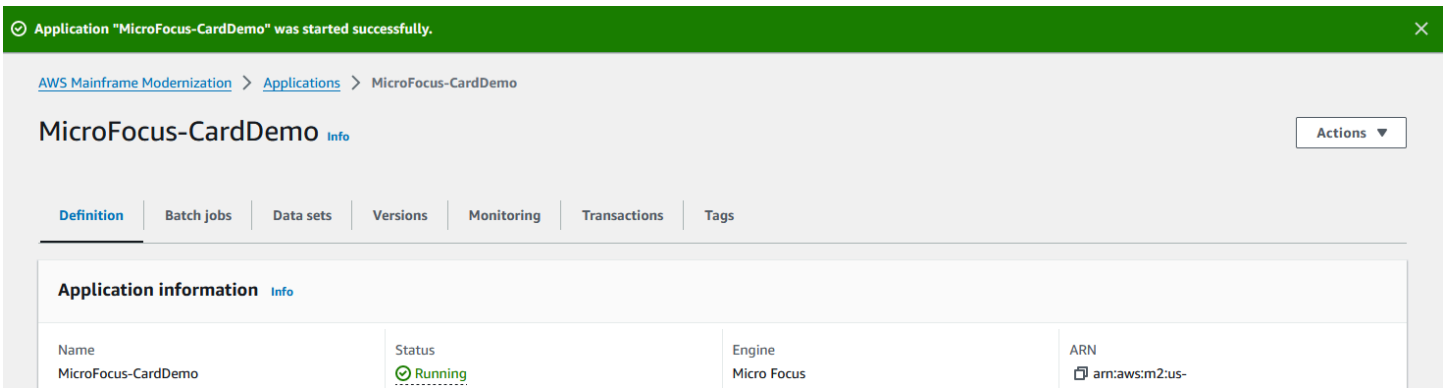
步骤 9：启动应用程序

启动应用程序

1. 在导航窗格中，选择应用程序，然后选择应用程序。
2. 选择 “启动应用程序”。



当 CardDemo 应用程序开始成功运行时，会出现一条横幅，上面写着以下消息：`Application name was started successfully`。“状态” 字段更改为 “正在运行”。



步骤 10：连接到 CardDemo CICS 应用程序

在连接之前，请确保您为应用程序指定的 VPC 和安全组与您为要连接的网络接口申请的 VPC 和安全组相同。

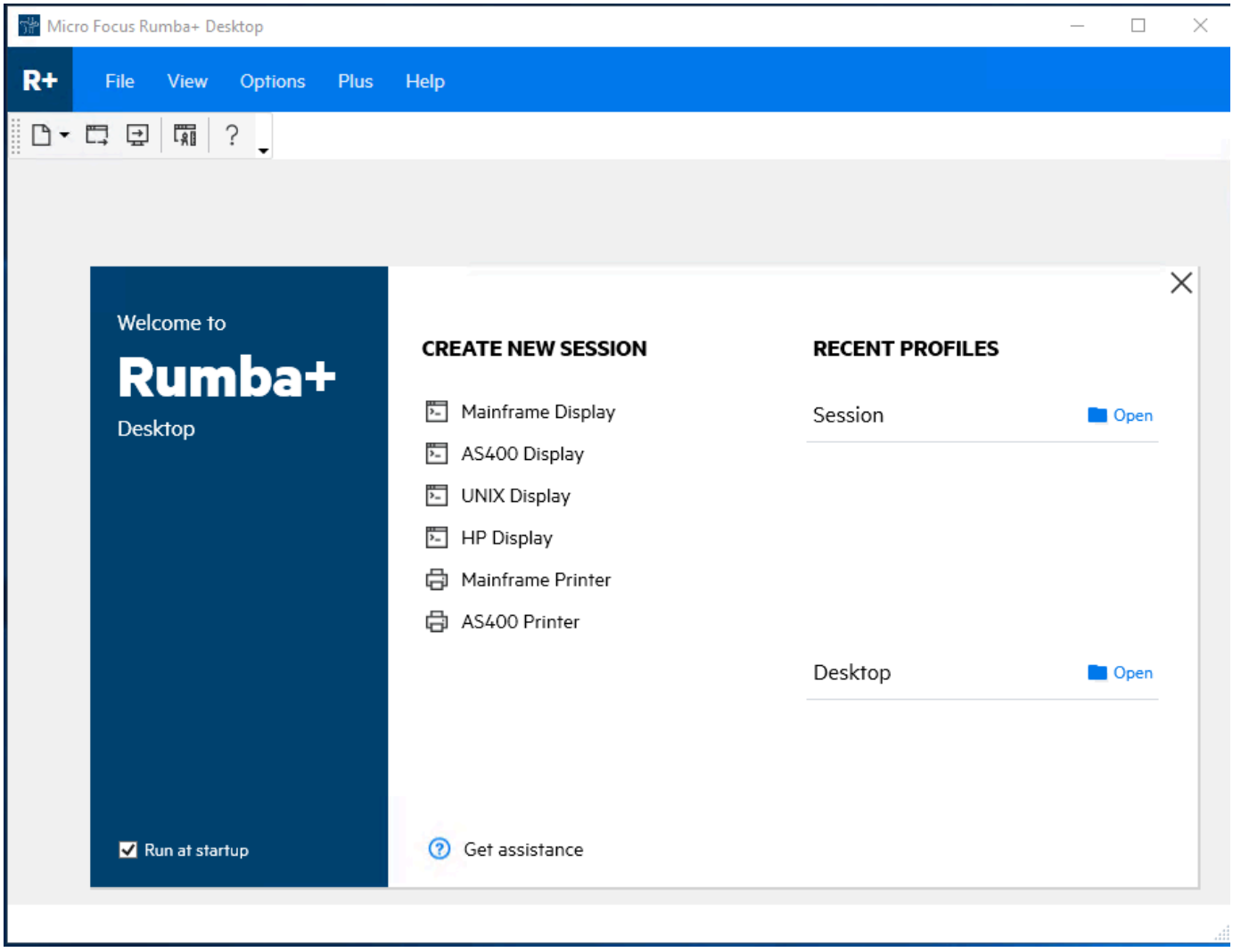
要配置 TN3270 连接，还需要应用程序的 DNS 主机名和端口。

使用终端仿真器配置应用程序并将其连接到大型机

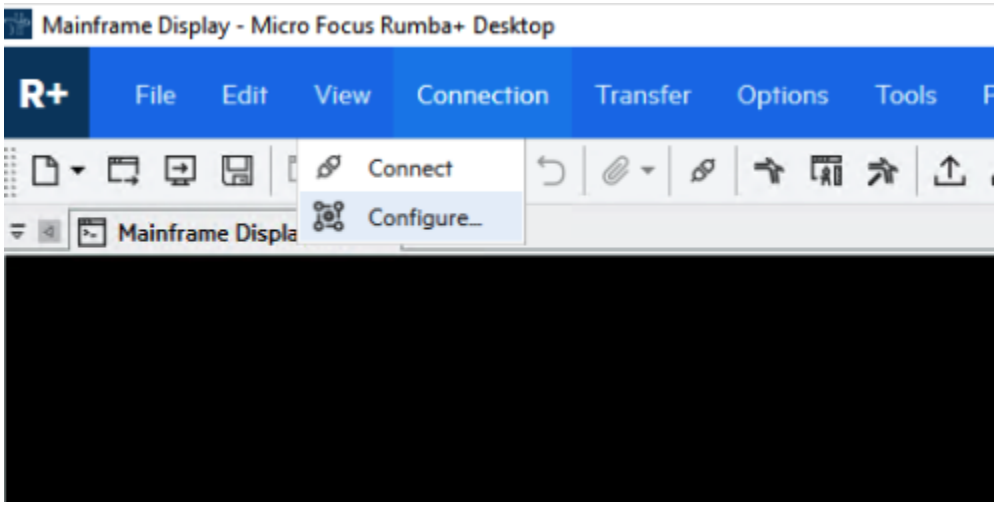
1. 打开AWS大型机现代化控制台并选择 “应用程序”，然后选择`MicroFocus-CardDemo`。
2. 选择复制图标复制 DNS 主机名。另外，请务必记下端口号。
3. 启动终端仿真器。本教程使用 Micro Focus Rumba+。

> **Note**
>
> 配置步骤因模拟器而异。

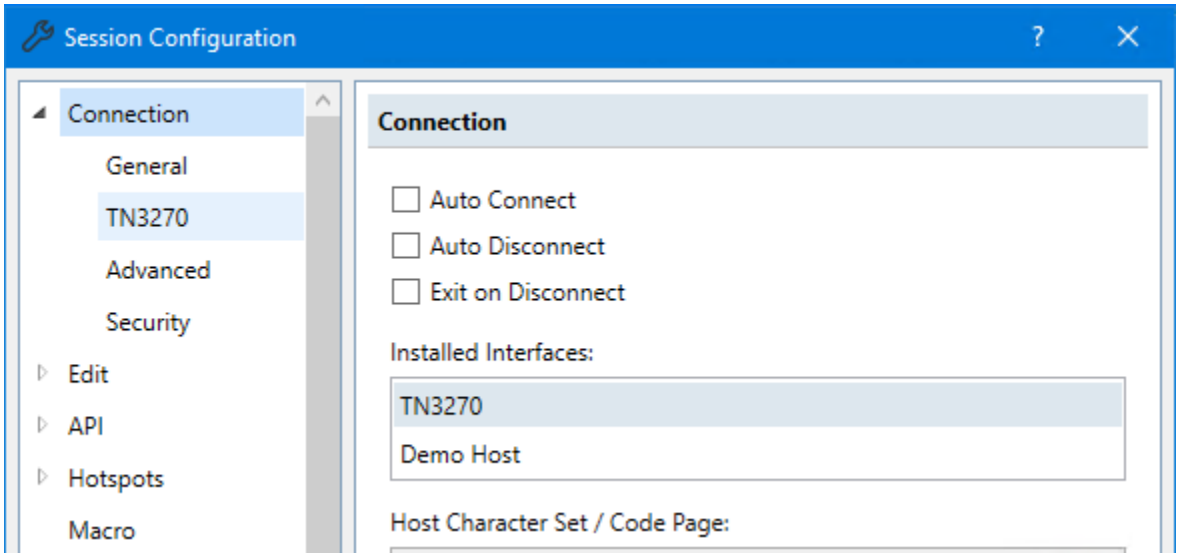
4. 选择 “大型机显示器”。



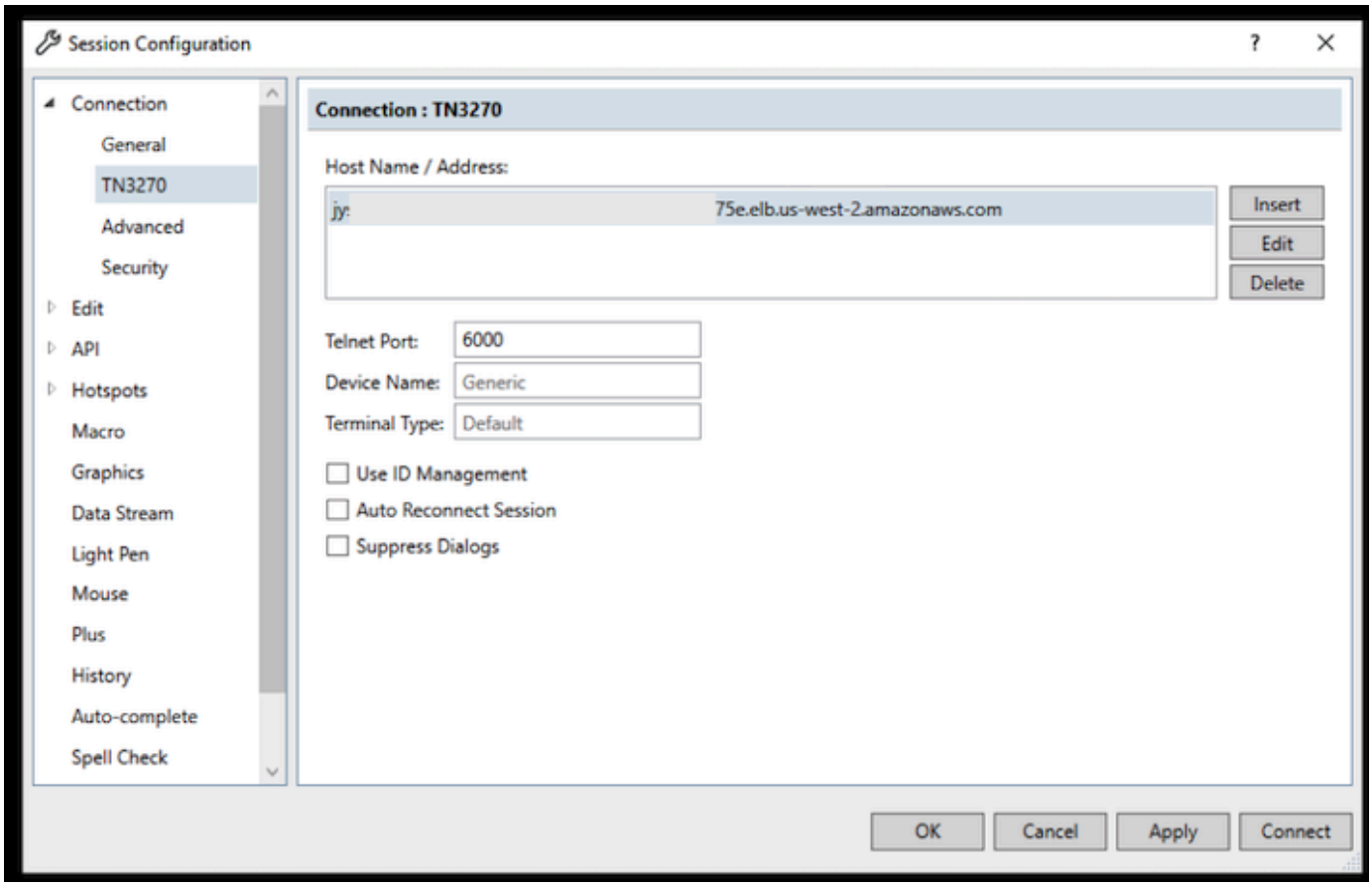
5. 选择 “连接”，然后选择 “配置”。



6. 在 “已安装的接口” 下TN3270，选择，然后在 “连接” 菜单下TN3270再次选择。



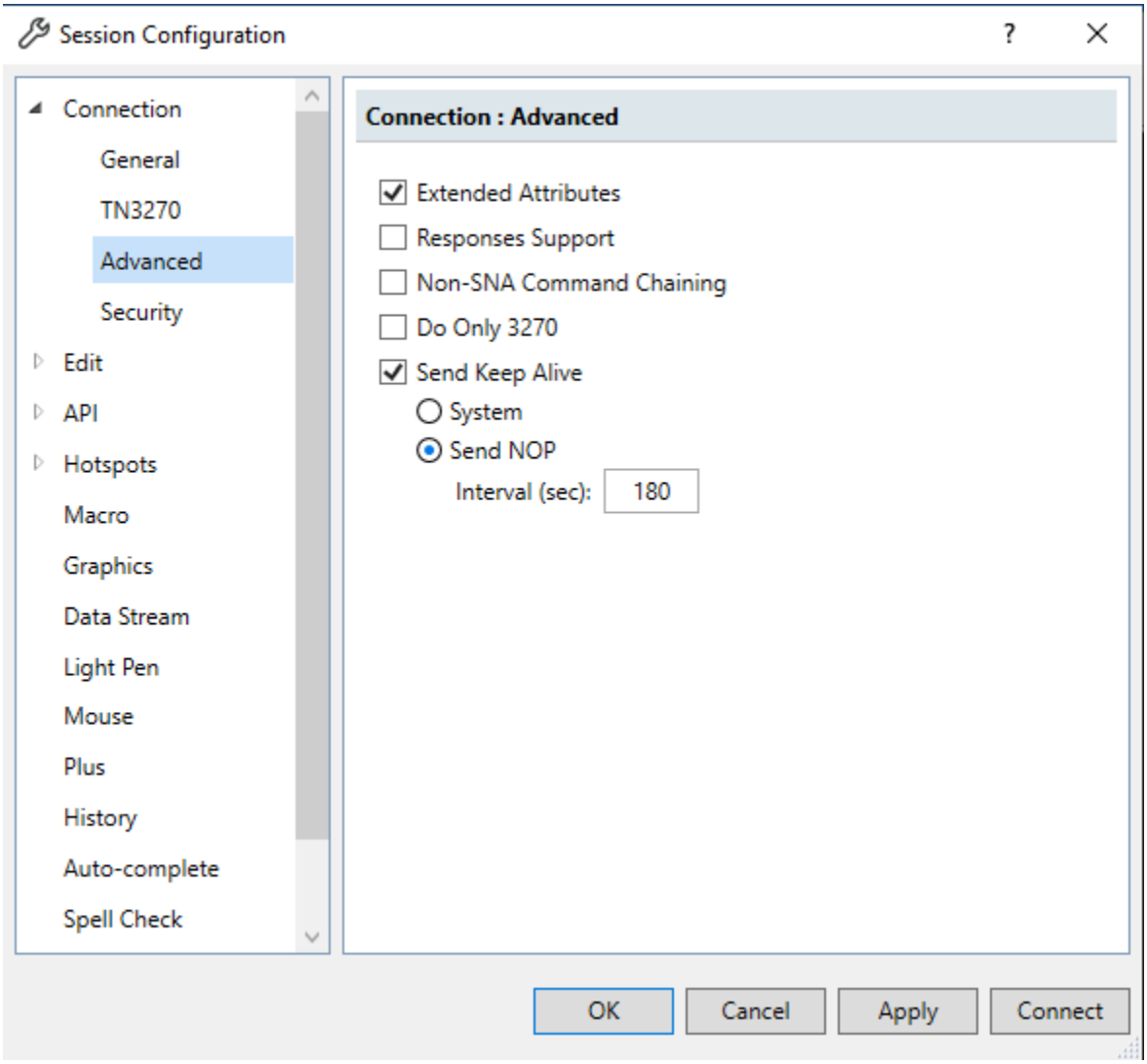
7. 选择“插入”，然后粘贴应用程序DNS Hostname的。6000为 Telnet 端口指定。



> **Note**
>
> 如果您在浏览器中使用 AWS AppStream 2.0，并且在粘贴值时遇到困难，请参阅 AppStream 2.0 用户问题疑难解答。

8. 在 “连接” 下，选择 “高级”，然后选择 “发送保持活动状态” 和 “发送 NOP”，然后输入 180 作为间隔。

> **Note**
>
> 将您的 TN3270 终端上的保持活动状态设置为至少 180 秒有助于确保网络负载均衡器不会中断您的连接。



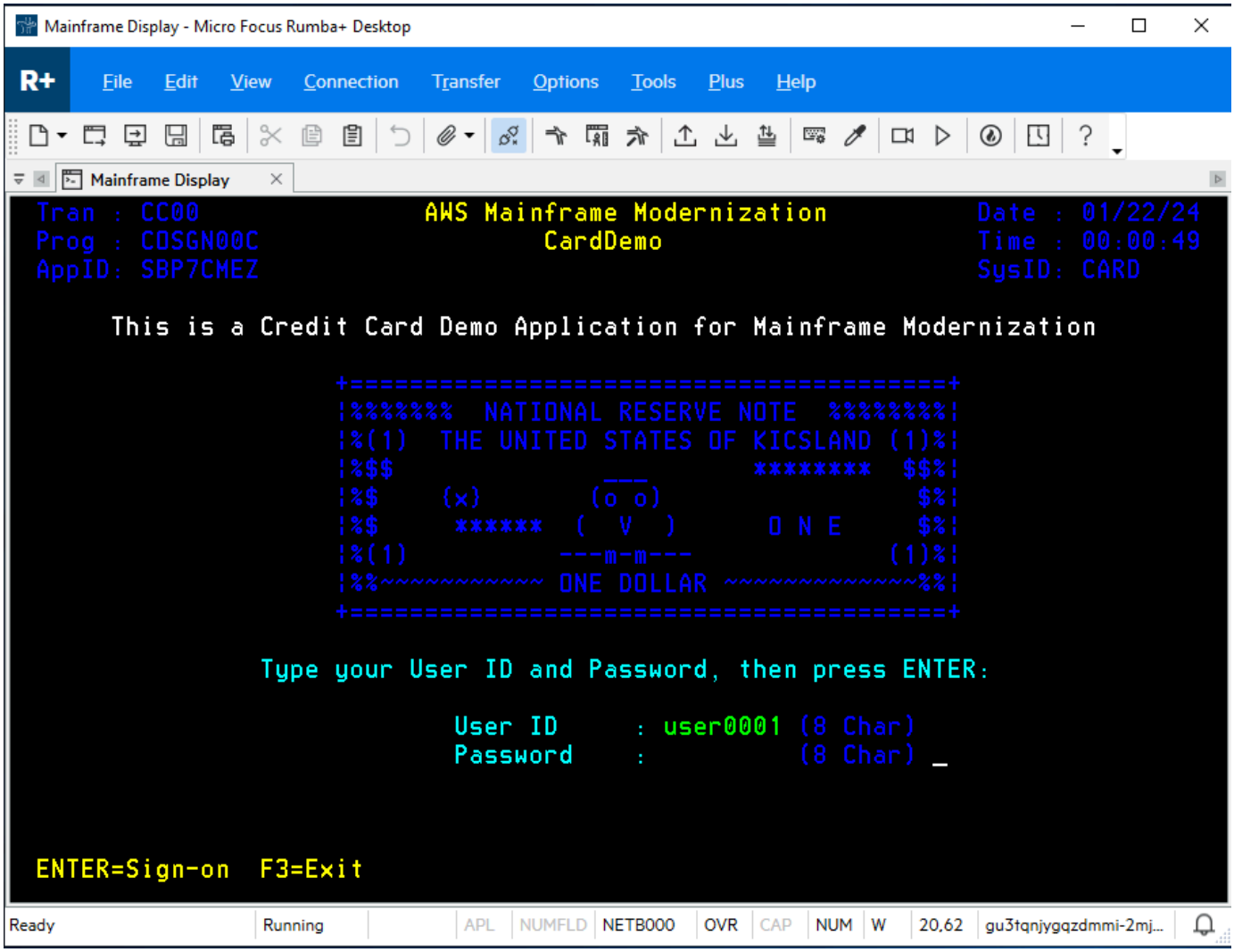
9. 选择连接。

> **Note**
>
> 如果连接失败：
>
> - 如果您使用的是 AppStream 2.0，请确认为应用程序环境指定的 VPC 和安全组与 AppStream 2.0 队列相同。
> - 使用 VPC Reachability Analyzer 分析连接。您可以通过控制台访问 Reachability Analyzer。
> - 作为诊断步骤，请尝试添加或更改应用程序的安全组入站规则，以允许来自任何地方的端口 6000 流量（即 CIDR Block 0.0.0/0）。如果您成功连接，则表明安全组阻止了您

的流量。将安全组来源更改为更具体的来源。有关安全组的更多信息，请参阅安全组基础知识。

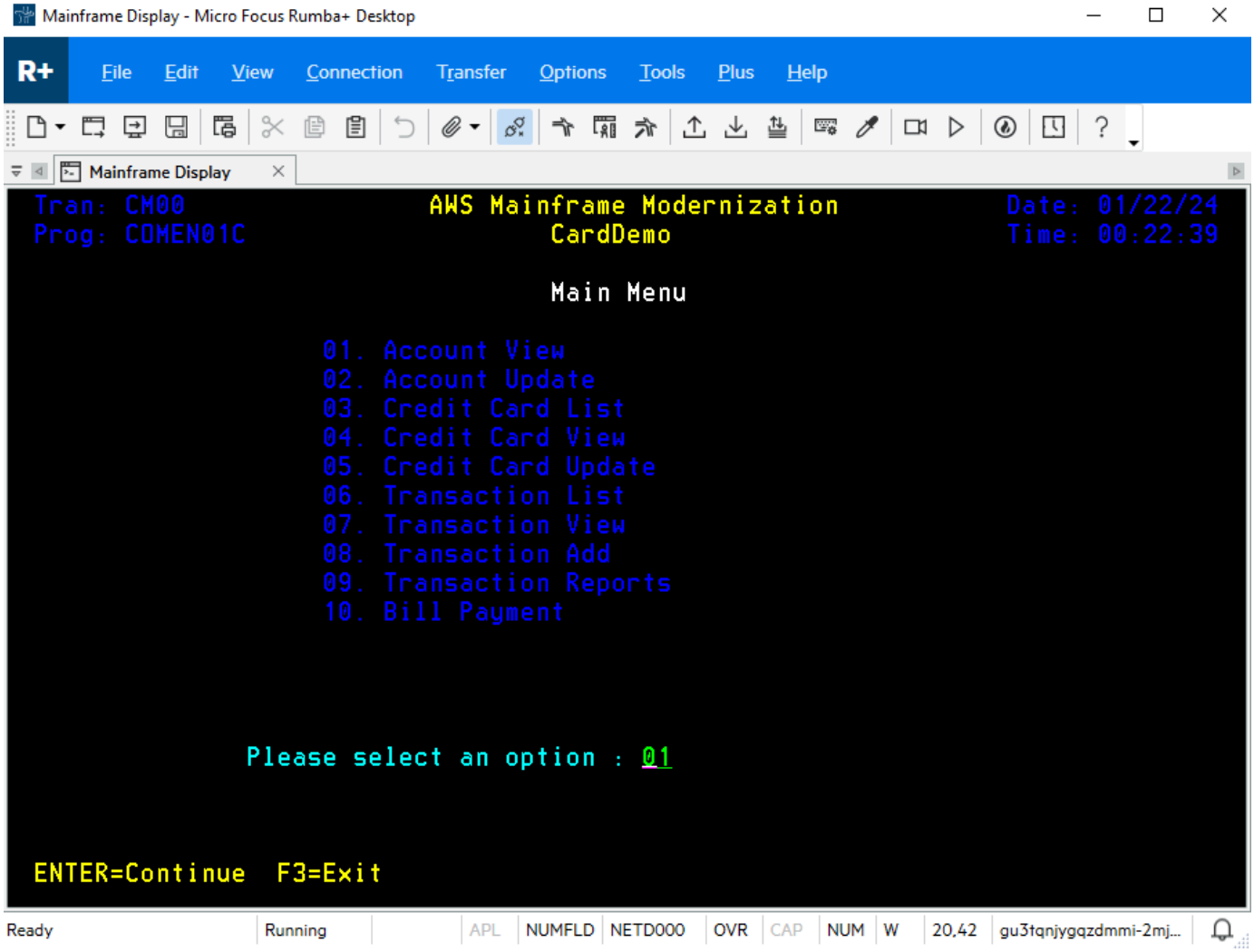
10. 输入USER0001用户名password和密码。

> **Note**
>
> 在 Rumba 中，“清除” 的默认值为 ctrl-shift-z，“重置” 的默认值为 ctrl-r。



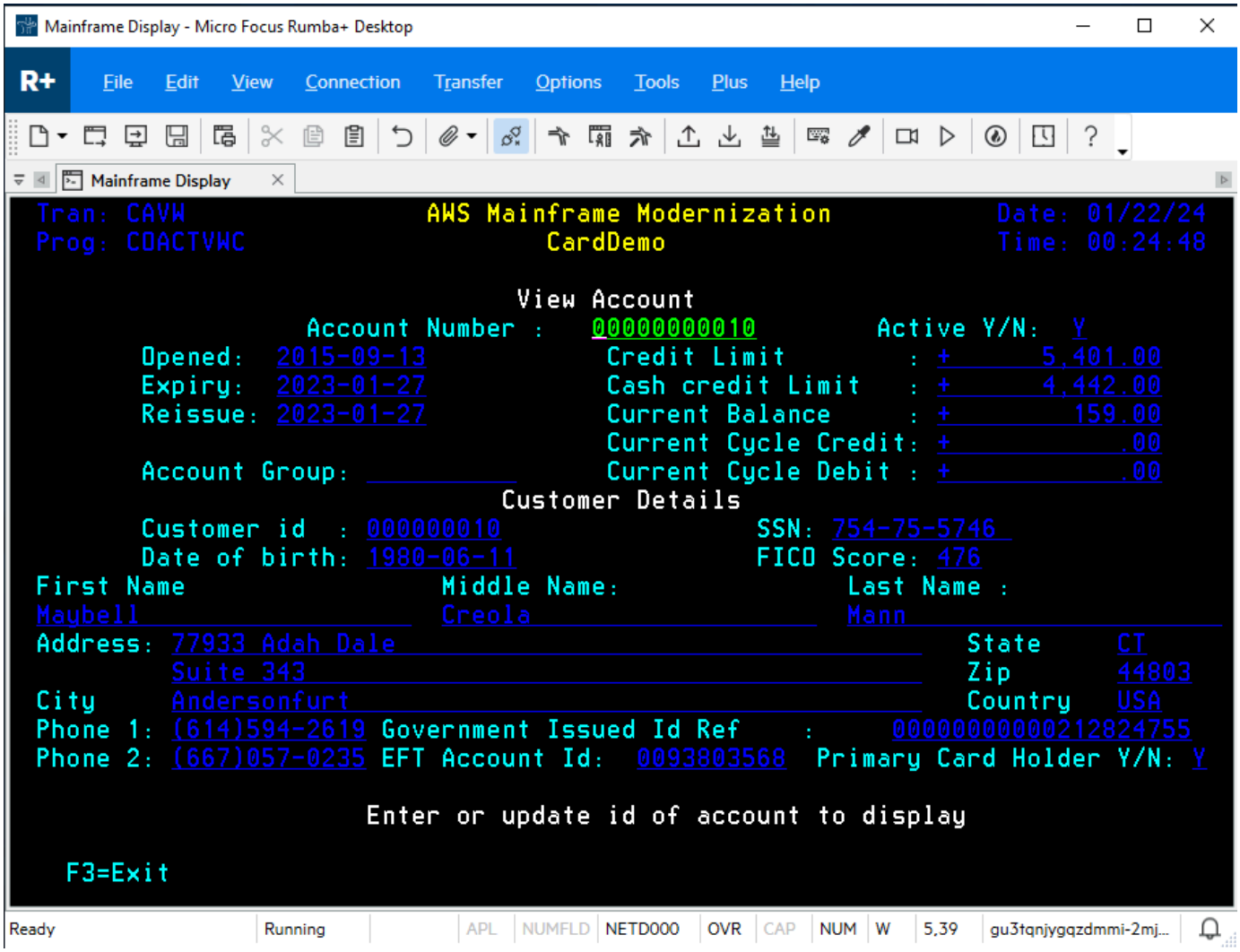
11. 成功登录后，您可以浏览 CardDemo应用程序。

12. 进入01账户视图。



13. 输入0000000010账号并按键盘上的 E nter。

> **Note**
>
> 其他有效账户为0000000011和00000000020。



14. F3按退出菜单，然后退F3出交易。

清理资源

如果您不再需要为本教程创建的资源，请将其删除，以免产生额外费用。为此，请完成以下步骤：

- 如有必要，请停止应用程序。
- 删除应用程序。有关更多信息，请参阅删除 AWS Mainframe Modernization 应用程序。
- 删除运行时环境。有关更多信息，请参阅删除 AWS Mainframe Modernization 运行时环境。
- 删除您为本教程创建的 Amazon S3 存储桶。有关更多信息，请参阅《Amazon S3 用户指南》中的删除存储桶。
- 删除您为本教程创建的AWS Secrets Manager密钥。有关更多信息，请参阅删除密钥。

- 删除您为本教程创建的 KMS 密钥。有关更多信息，请参阅删除 AWS KMS 密钥。
- 删除您为本教程创建的 Amazon RDS 数据库。有关更多信息，请参阅 A mazon RDS 用户指南中的删除 EC2 实例和数据库实例。
- 如果您为端口 6000 添加了安全组规则，请删除该规则。

后续步骤

要了解如何为现代化应用程序设置开发环境，请参阅教程：设置 AppStream 2.0 以与 Micro Focus Enterprise Analyzer 和 Micro Focus Enterprise Developer 配合使用。

现代化方法

迁移很复杂，并且设计多个变量。AWSMainframe Modernization 提供了一种先进的方法，不仅可以通过提高敏捷性来提供一些短期回馈，还提供大量后续进行优化和创新的机会。此外，AWS Mainframe Modernization 还有助于简化现代化过程，同时仍尊重客户公司和业务的细节。AWS Mainframe Modernization 支持的两种主要方法为自动重构和更换平台。选择哪种方法取决于您的客户的具体情况。

自动重构使用 AWS Blu Age 工具将代码、数据和依赖x自动转换为现代语言、数据存储和框架，同时保证功能等同于相同的业务功能。

更换平台使用 Micro Focus 工具将大型机工作负载转换为 AWS 上的敏捷服务。

您可以考虑分阶段实现现代化之旅。第一阶段包括三个阶段：评测、动员以及迁移与现代化。第二阶段包括运行和优化，您可以在其中发现更多的创新机会。

主题

评测阶段

概况来说，评测阶段着眼于您是否准备好迁移。您定义一个业务案例，然后通过 AWS 提供的研讨会和沉浸日（演示和实验室）来培训团队成员。研讨会和沉浸日涵盖不同的主题。这些任务是在 AWS Mainframe Modernization 外部执行的。

动员阶段

在动员阶段，首先启动项目，然后执行发现过程，从大型机应用程序中提取数据并将其摄取到迁移工具中。确定要迁移的应用程序，然后选择几个应用程序进行试验。完善业务案例，制定迁移计划，并决定如何处理安全性与合规性、帐户治理以及运营模式。与适合的团队成员合作设置一个云卓越中心。运行试点并记录您了解到的内容。完善迁移计划和业务案例。这些任务中有很多是在 AWS Mainframe Modernization 外部执行的。

迁移与现代化阶段

迁移与现代化阶段适用于每个应用程序，包括多项任务，如分配人员、进行深入发现、在 AWS 找出合适的应用程序架构、设置应用程序运行时环境、更换平台或重构代码、与其他系统集成，当然还有测试。在该阶段结束时，将更换平台或重构后的应用程序部署到生产环境中，并切换到 AWS 上的新系统。这些任务中的大多数或全部都是在 AWS Mainframe Modernization、其他 AWS 服务或 AWS Mainframe Modernization 提供访问权限的工具中执行的。

如果要使用自动重构，请参阅 Blu Insights。AWS现在，通过单点登录即可从 AWS Management Console 获取 Blu Insights。您无需再管理单独的 AWS Blu Insights 凭证，可以直接从 AWS Management Console 中访问 AWS AWS Blu Age Codebase 和 Transformation Center 特征。

要将数据从大型机迁移到 AWS，我们建议使用 AWS SCT 和 AWS Database Migration Service。有关更多信息，请参阅《AWS Schema Conversion Tool 用户指南》中的什么是 AWS Schema Conversion Tool？ 和《AWS Database Migration Service 用户指南》中的什么是 AWS Database Migration Service？

操作和优化阶段

在操作和优化阶段，您将专注于监控已部署的应用程序、管理资源以及确保安全性和合规性为最新状态。您还可以评测是否有机会优化迁移的工作负载。

概念

AWS 大型机现代化提供了工具和资源，可帮助您迁移、现代化和运行大型机工作负载。 AWS

主题

应用程序

大型机现代化中正在运行的大型 AWS 机工作负载。应用程序由一组批处理作业、交互式事务（CICS 或 IMS）或其他组件组成。您可以定义其范围。您必须定义和指定工作负载所需的任何组件或资源，例如 CICS 事务或批处理作业。

应用程序定义

大型机现代化中运行的应用程序（大型机工作负载）所需的组件和资源的定义或规范。 AWS 请务必将定义与应用程序本身分开，因为可以对由不同的运行时环境表示的多个阶段（预生产、生产）重复使用相同的定义。

批处理作业

配置为无需用户交互即可运行的计划程序。在 AWS Mainframe Modernization 中，您需要将批处理作业 JCL 文件和批处理作业二进制文件存储在 Amazon S3 存储桶中，并在应用程序定义文件中提供两者的位置。运行批处理作业时， AWS 大型机现代化会报告以下状态值：

正在提交

批处理作业正在提交。

暂停

批处理作业处于暂停状态。

正在分派

批处理作业正在分派。

Running

批处理作业当前正在运行。

正在取消

批处理作业正在被取消。

已取消

批处理作业已取消。

成功

批处理作业已成功完成运行。

失败

批处理作业失败。

成功但出现警告

批处理作业成功完成运行，但报告了一个小错误。作为 GetBatchJobExecution 响应的一部分返回的任务条件代码指出了错误的原因。

配置

环境或应用程序的特征。环境配置包括引擎类型、引擎版本、可用性模式、可选文件系统配置等。

应用程序配置可以是静态的，也可以是动态的。只有当您通过部署新版本来更新应用程序时，静态配置才会发生变化。动态配置（通常是一种操作活动，例如开启或关闭跟踪）会在更新后立即发生变化。

数据集

包含供应用程序使用的数据的文件。

环境

为托管一个或多个应用程序而创建的 AWS 计算资源、运行时引擎和配置详细信息的命名组合。

大型机现代化

将应用程序从传统大型机环境迁移到 AWS的过程。

迁移之旅

传统应用程序的迁移和现代化 end-to-end 过程，通常由以下几个阶段组成：评估、移动、迁移和现代化以及运营和优化。

挂载点

文件系统中的一个目录，用于访问存储在该系统中的文件。

自动重构

对传统应用程序构件进行现代化而使其在现代云环境中运行的过程。该过程可以包括代码和数据转换。有关更多信息，请参阅 AWS Mainframe Modernization 自动重构。

更换平台

将应用程序和应用程序构件从一个计算平台移动到另一个计算平台的过程。有关更多信息，请参阅 AWS Mainframe Modernization 更换平台

资源

计算机系统中的物理或虚拟组件。

运行时引擎

便于应用程序运行的软件。

使用 Blu Age 自动重构应用程序 AWS

AWS Blu Age 的自动重构为大型机 end-to-end 应用程序的迁移和现代化提供了解决方案。重构过程的步骤如下：

- 分析清单
- 分析依赖项
- 自动转换代码
- 捕获和管理测试场景

您可以在 Blu Insights 工具中完成前面的步骤，该工具可通过 AWS 大型机现代化控制台的单点登录获得。有关 Blu Insights 的更多信息，请参阅 Blu Insights 文档。

当你对转换后的源代码感到满意时，就该转到了 AWS，你将在那里完成以下步骤：

- 构建并部署重构的应用程序。
- 在 AWS 大型机现代化中部署和监控您的应用程序。

AWS Blu Age Runtime（非托管）是 AWS 大型机现代化服务与 AWS Blu Age托管一起提供的产品之一。通过管理 AWS Blu Age，您可以将现代化应用程序部署到 AWS托管环境中，从而简化您的体验，因此您无需管理运行现代化应用程序的底层基础架构。相比之下，借 AWS 助 Blu Age Runtime（非托管），您可以在自己的 AWS 账户中部署现代化应用程序，这样您就可以管理自己的基础架构。借 AWS 助 Blu Age Runtime（非托管），您可以灵活地操作以您想要的方式运行现代化应用程序所需的所有技术组件。

AWS Blu Age Runtime（非托管）可在亚马逊 EC2 和亚马逊 ECS 上部署。 AWS Fargate

主题

- AWS Blu Age 发行说明
- AWS 蓝光时代运行时概念
- AWS Blu Age 运行时配置和配置文件
- AWS 蓝光时代运行时 API
- AWS Blu Age 运行时（非托管）设置
- 使用 Blu Age Developer IDE 修改源代码

AWS Blu Age 发行说明

本节包含 3.5.0 版以后的 AWS Blu Age Runtime 和现代化工具的发行说明，最新版本在前，按版本号整理。

> **Note**
>
> 有关本文档之前的发行说明，请联系 AWS Blu Age 交付服务。有关 Blu Insights 最新功能的信息，请参阅 Blu Insights 发布版本。

主题

版本说明 3.10.0

此版本的 AWS Blu Age Runtime 和现代化工具侧重于整个产品的核心基准升级和改进，力求在所有转型和执行步骤中提高性能和稳健性。此版本中的一些关键特征和更改包括：

- 版本从 Java 8 升级到 Java 17，提高了安全性和性能，并允许客户部署和运行以更现代的语言实现的应用程序以及使用最新的第三方框架版本。
- 还支持管理用户或作业之间的大型共享内存空间，存储应用程序或实例重启后可重复使用的数据。
- 使用分页机制更快地访问 Blusam 中的大型数据集，从而可以增量检索记录子集。

有关此版本所包含更改的更多信息，请参阅以下部分。

运行时版本 3.10.0

此运行时基于 Java17、Spring2.7 和 Angular16。

zOS

新功能

- Blusam-通过分页机制增加了对大型数据集的支持，该机制使用页面存储和加载索引

改进

- 增强了 DataUtils .compare，可处理从字符串到数字的较低优先级转换
- 增加了通过 YML 属性 Data ByteRange Simplifier 检查是否使用不正确的值创建了否的支持。byteRangeBounds查看
- 增强了 removeOSI ()，支持初始化 GraphicAlphanumericType 带空字符的
- 增加了作业操作的稳健性和安全的 GDG 状态读取
- Blusam-增加了对通过名为.removeCache () 的新方法清除 Blusam 数据集的 Ehcache 的支持 CoreBluesamManager
- Blusam-改进了常规 Blusam 数据集的删除/重命名行为
- Redis-增强了对解锁数据集和清除记录锁定的支持
- JICS-改进了失败请求的错误消息
- JCL-增加了对基于点字符的 ControlM 变量串联的支持
- JCL-为 GDG 文件添加了对 Write ADVANCE (ADV) 的支持

- JCL-删除所有 GDG 文件后增强了对当前世代号的支持
- JCL-增强了对创建数据集时从目录中读取 RDW/RecordSize 的支持
- JCL-添加了在打开文件时更新资源对象（从 AbstractSequentialFile）的支持，其大小相当于数据输出记录
- JCL-改进了 IDCAMS 的性能
- JCL-通过添加 “CHAR” 作为 “字符” 的别名，增强了对打印语句的支持
- SORT-增强了对从 Blusam 固定长度数据集复制到可变长度数据集的复制操作的支持
- SORT-增强了排序语法，可处理一些特定的语句

AS400

新功能

- 增加了对用户空间及其相关 API 的支持
- 增加了对 SNDPGMSG 的 TOMSGQ 参数的支持，并实现了消息队列
- CL-为 OVRPRTF 命令添加了对 FILE 和 SPLFNAME 参数的支持
- CL-增加了对使用 CPYF 命令处理相应分区表的库的支持
- CL-增加了对处理 CHGCURLIB 命令和构建查询时考虑当前库的支持
- CL-增加了对作为调用堆栈跟踪一部分处理 cl 命令的支持

改进

- 经过改进 MessageHandlingBuilder ，可以更好地处理调用堆栈跟踪条目
- 改进了 contextPreConstruct 功能的并行执行
- 改进了 SFLINZ 创建记录时的显示属性
- 改进了 SAVOBJ，允许处理多个输出文件
- 通过在从 Java 程序调用 programCallStack 时将其添加到 groovy 程序的处理方式来改进这些程序的处理
- 改进了帮助模式的顶部定位检测
- 当为 SNDPGMMSG 提供 tomsgQ 参数时，改进了 TopGMQ 的功能
- 改进了预定义消息的获取和消息加载器的功能
- 改进了 CPYTOIMPF 对内容中分隔符的处理
- 改进了 READ 记录的释放锁定

横向功能

新功能

- 在前端添加了系统消息的翻译
- 在中添加了一个用于 ExecutionContext 返回程序调用堆栈的新方法
- 无论实际环境如何，都要设置行分隔符（用于数据简化器）
- 增加了配置 SQL 模型 JSON 路径的可能性

改进

- 改进了比较方法 DataUtils。 compareAlphInt() 当涉及填充时
- 创建标志以允许在游标查询中自定义异常行为
- 改进了图形低值数据库转换

第三方

- 升级以缓解 CVE-2024-21634、CVE-2023-34055、CVE-2023-34462、IN1-JAVA-ORGSPRINGFRAMEWORKSECURITY-5905484、CVE-2023-46120、CVE-2023-6481、CVE-2023-6378

现代化工具版本 3.10.0

zOS

改进

- COBOL-增加了对 ABS 功能的支持
- JCL-增强型变量作用域：附加到 STEP 而不是 JOB
- 增强了低/高值的游标参数注入
- 改进了 CSD 解析，尤其适用于远程交易

AS400

改进

- 删除了控制水平指示器的空白支票

- 为导入/导出关键字添加了对外部名称的支持
- 在字段上添加了对 %LEN 的支持
- CL-增加了对 CLLE 语言新运算符的支持
- CL-增加了对嵌套 IF 的支持
- COBOL-改进了与多个按键一起使用时对 START 命令的处理
- DSPF-改进了使用记录号对光标位置的处理
- DSPF-改进了带符号数字、仅限数字的字段和大比例字段的格式
- DSPF-改进了屏幕常规帮助的标题确定
- DSPF-改进了对输入/输出规范的支持
- DSPF-改进了在验证数值字段期间对分组分隔符的处理
- 改进了映射输出/DDS 记录
- 改进了打印机文件 REFFLT 关键字解析引用字段的能力
- RPG-增强了对 “全部免费” 语句的支持
- RPG-改进了条件解析并增加了对处理不带结果 TAG 的 CABXX 的支持
- RPG-改进了对数值字段的输入规范处理
- RPG-改进了在 IF/ELSEIF/WHEN 条件下对过程调用的处理
- RPG-改进了在 dspf 文件上调用 READ 命令时的处理
- RPG-改进对引用不存在的 DDS 的文件的支持
- 改进传递物理记录格式名称时对 REFFLD 的处理
- 增加了使用 “返回” 作为数据库列名的支持

横向功能

新功能

- Oracle-允许定义用户而不是 SYS 来存储内置函数

改进

- 将 Java 版本从 v8 升级到 v17
- 使用群集列名改进了 SQL 条件

- 从视图中添加了对 ORDER BY 子句的支持

版本说明 3.9.0

此次发布的 AWS Blu Age Runtime 和 Moderization Tools 侧重于整个产品的多项横向增强，旨在提高高可用性架构的性能，以及将任务执行提升到更高水平的新功能。此版本中的一些关键功能和更改包括：

- 版本从 Angular 13 升级到 Angular 16，提高了安全性，并推出可提高客户在线应用程序性能的新功能。
- 在 AS400 中添加了对跨作业功能的支持，其主要亮点是作业可以在它们之间同步发送查询消息，从而在现代化作业中实现脱钩。
- Redis 的使用性能改进，包括连接池优化、连接的高安全性以及升级的数据集锁定机制。

有关此版本所包含更改的更多信息，请参阅以下部分。

运行时版本 3.9.0

zOS

新功能

- 排序程序：更新了固定长度的 VSAM 输入
- JHDB DB：添加了可配置的超时

改进

- 增强了对于流的行分隔符（用于文件串联）的支持
- 增强了对打开串联的顺序文件的支持。文件打开 DataSetIndex 后初始化
- 增强 a NumericEditedType 受数值影响时对虚拟小数分隔符的支持
- 增强了 NumericEditedType 对负值的支持
- IDCAMS：现在可以使用.yml 中定义的 “编码” 属性读取 SYSIN 卡 application-utility-pgm
- IDCAMS：更新了语法以支持 DEFINE CLUSTER 语句中的 FILE(..) 参数
- INFUTILB：增加了对 DFSIGDCB 参数的支持，以覆盖 DD SYSREC 的 DCB 参数
- INFUTIL：增强了对“DFSIGDCB YES”参数的支持

- 改进了 SPLICE 以处理庞大的输入文件
- DFSORT：改进了备注字段的处理
- DFSORT：增加了对（签名/未签名）自由形式数字格式 (SFF/UFF) 的支持
- SORT：增加了对 OPTION PRINT 和 OPTION ROUTE 语句的解析支持
- SORT/ICEMAN：增加了对封闭式除法运算的支持（带有 DIV 运算符的字段）
- 增强了对使用通用密钥的 CICS READ 的支持
- 函数 StringUtils .chargraphic 已修复，用于从图形类型中删除 SOSI
- 开启时增强性能 DataUtils。 isDoubleByte编码
- JCL：增强了对临时数据集的 KEEP 处置模式的支持。系统将处置改为 PASS
- JCL：动态处理 DCB 参数
- JCL：增强了错误值的 SUM FIELDS 输出
- JCL：CommonDDUtils::getContent 现在在目录中搜索 recordSize
- JCL：创建数据集时从目录中读取 rdw/recordSize 属性
- JCL：增加了对 DCB=.MYDD 的支持，以便将 DD 的 DCB 参数复制到同一作业步骤中的另一个 DD
- JCL：改进了记录大小继承系统
- JCL：新增 (Redis) 专属数据集锁
- Redis：为独立模式添加了 SSL 支持
- Redis：添加了带锁的同步 Redis 锁定计数
- Redis：支持 Redis 锁的池参数
- Redis：优化了 Redis 的元数据刷新
- Redis：改进了 redis 集群支持
- 改进了 IO 模式下打开的锁
- 改进了数据集锁定性能并清除未使用的锁
- 取消注册文件期间数据集的路径增强
- 改进了预取窗口缓存失效问题
- 增加了对使用线程安全实用程序数据源提供程序的支持
- 增强了 datasetState 无效性检查
- 增强了对不重新打开已打开的数据集的支持
- 增加了作业最终操作的稳健性

- 增强了对允许重复项的键的索引顺序的支持
- 增强了对跳过列表序列化顺序的支持
- 增加了对调试转储功能的支持，以帮助诊断索引顺序问题
- 增强了对元数据刷新的支持
- 增强了对 Blusam 批量读取的支持

AS400

新功能

- 创建应用程序上下文注册表
- 支持 DSPF 关键字 CLRL(NO) 支持记录锁定监控
- Support 支持 keyed DataQueue
- 支持批处理作业的 INQUIRY 消息
- 增加了对 AS400 COBOL 的程序描述的打印机文件的支持
- 处理 RMVJOBSCDE cl 命令
- 改进了 RUNSQL/DLYJOB
- CHKOBJ：引发参数 LIB 的遗留错误代码
- SNDPGMMSG：支持字符串参数
- RTVDTAARA：改进了 LDA 中的子字符串
- DSPFD：增加对特定文件名的 FILE 参数支持
- RUNQRY：支持 QRY PARAM 中的 sql 文件
- CRTDUPOB：支持在数据区域之间复制数据
- SBMJOB：将指令转换为使用 JobQueueManager
- OPNQRYF：增加了对 Qtemp 库的支持
- CRTDUPOBJ：改进了复制分区内容的逻辑
- CRTDUPOBJ：为视图添加了对 Qtemp 的支持
- RTVSYSVAL：支持 SYSVAL 值，CL 命令中的 QDATFMT
- CHKOBJ：增加了对 OUTQ 的支持
- RTVJOBA：支持 SWS 参数

- SNDPGMMSG 和 RCVMSG：增加了受支持的参数，包括 MSGF、MSGFLIB、MSGDTA、MSGTYPE、KEYVAR、MSGKEY、MSGID

改进

- 改进 WORKSTATION I/O 卡支持
- 改进了对叠加上一条消息的设置消息的处理
- 支持有关 array-messageline 的其他消息信息
- 改进了 EVAL、SortA、象征值中的独立数组包装器访问权限
- 改进了在线应用程序结束时的 DAO 清理
- 增加了对其他日期格式的支持，并改进了对字符串输入的处理
- 通过添加系统值帮助器类 Decode 和 CL 命令生成参数，改进了 SYSVAL 的 CVTDAT 处理 SbmJob
- 已从组件扫描中移除软件包 com.netfective.bluage.gapwalk.rt.blu4iv gapwalk-cl-command
- 改进了对消息队列 API 的预定义消息的支持
- 改进了对 retrieveSubfileRecord 用其他程序编写的记录的支持
- 改进了对消息队列 API 的即时消息的支持
- 改进了提交作业时对本地数据区域的处理
- 服务器启动时 JobQueues 自动启动
- 使用 applicationContext 配置来解码 SBMJOB 的参数
- 改进了系统提供的错误消息
- 允许 RTVMSG 在嵌套的子目录中搜索 .properties 文件
- 处理绑定到错误/无效指针的实体的重置
- 已改进 MessageHandlingBuilder ，可将 msgID 和 MsgFile 名称显示为 RCVMSG 的字符串
- 改进了消息队列 API 的 withMsgFile命名方法
- 改进了数据区域锁定机制
- RTVMBRD：支持参数 FILE 的小写和大写格式
- CRTDUPOBJ：改进了视图的处理
- CPYTOSTMF：改进了对连接的处理
- CPYF：改进了从平面文件中复制时的目录名称处理
- RCVF：正确处理 DEV/RCDFMT 参数以及 groovy 和 java 的 RCDFMT 的转换
- RCVF：处理后续调用并避免重置游标

- CPYF：增加了对从平面文件写入的支持
- CRTDUPOBJ：添加了使用 Qtemp 库对新对象的处理
- CHGDTAARA：将数据区域的最大长度从 256 增加到 2000
- SAVOBJ：确保保存的记录按插入顺序排列
- RTVDTAARA：已检索到值（未修剪）
- CHKOBJ：成员不存在时返回正确的监视器消息
- RTVDTAARA：增加了对 LDA 子字符串的支持
- RTVDTAARA：返回不超过 RTNVAR 参数中指定变量长度的空格
- RTVDTAARA：支持起始和长度的整数参数，并支持最新的转换格式
- CHGDTAARA：增加了对包含下限和上限的参数的支持
- CHKOBJ：处理参数对象类型的 VIEW 值
- CHKOBJ：无论视图是否存在，结果都设置为 true

横向功能

新功能

- 处理生成报告到 .txt 文件
- 为 secret manager 添加了 currentSchema XA 数据源属性
- 添加 database.cursor.raise.already.opened.error yml 属性，使框架能够在已打开的游标打开时引发 SQLCODE 错误 502

改进

- 在 Amazon EC2 包装上的 AWS Blu Age 中添加了 gapwalk poms
- 默认使用新的信号处理器范例
- 添加对处置为 MOD 或 OLD 时锁定的支持
- 添加了缓存以存储数据库日期时间模式
- 改进了的检查功能 PackedType
- 改进 DataUtils Records 的.setto 函数 VariableSizeArray
- 处理与运行单元有关的 MQ SYNCPOINT 选项
- 启用框架以在回滚事务中设置 SQLCODE

- 根据引擎密钥添加了自动驱动程序类别名称
- 程序/事务超时
- 访问游标时在回滚后恢复游标位置

第三方

- 升级 SnakeyAML、Redisson 和 Amazon SDK， YamlBeans 移除（缓解 CVE-2022-25857、CVE-2023-24621、CVE-2023-42809、CVE-2023-44487）

现代化工具版本 3.9.0

zOS

改进

- 增强了对 XML-TEXT 作为字符串类型目标的源的支持
- 增强了 STM 到 UML 的工作流程，支持 X/(Y/Z）分割模式
- JHDB DB：在任何数据库更新之前接受 ROLLBACK 调用
- JHDB DB：即使事务终止也接受 ROLLBACK (NOP)
- JCL：改进了步骤验证函数
- SORT：处理带有区域十进制负值的 SUM 函数
- COBOL：增加了对字符串文字中单/双引号转义的支持

AS400

改进

- 通过添加前导零，改进了内置函数 %editc 对编辑代码 X 的处理
- 改进了对仅限输入字段初始值的处理
- 为帮助对话框添加了操作键
- 显示在底部的动态表的页脚记录
- 为指定实际 RECORD-KEY 的文件处理了 START 命令，但没有 KEY PHASE
- 为 float 和 NumberUtils:: pow 类型添加了默认值
- 增加了使用 LIKE(IN）定义变量的支持

- 更新了 FOR 循环处理以支持省略可选元素
- 更新了 RPG 解析以将记录与 CTDATA 数组名称相关联
- 改进了 CABxx 语句指示器的处理
- 支持 COMMIT 关键字上的可选参数
- 改进了 LF 中的 FORMAT 关键字支持
- 管理包含高于等于（或低于等于）指示器的 LOOKUP 操作代码
- 处理了双引号中声明的 PF 键名称
- 改进了 EDTCDE X 的处理以不抑制前导零
- 改进了对打印机文件中不生成未命名标签的 MSGCON 的支持
- 字段 CONTENT 由多个数据结构共享
- 结合 SFLMSG/SFLMSGID 处理了 ERRSFL 参数
- 改进了完全免费 rpg 的 proc 声明范围之前的主代码
- 添加了解析条件控制规范
- 改进了 dataholdermapper 中对 setErrSfl () 方法的支持
- 改进了内部创建的变量的类型解析
- 改进了对 Z-ADD opcode 的支持
- 改进了对带有 DFT 值的常量字段的处理
- 改进了对程序状态 ds 中整数字段的支持
- 处理了 ENTRY 参数中的指示器分配
- 改进了通过 ref/reffield 关键字传播的关键字的过滤器
- 支持的未命名 DataArea 数据结构
- 改进了对指针数据类型的处理
- 处理了用于使用输出字段中的 LIKE 关键字支持数组访问定义变量的数组元素
- 改进了对带符号数字的支持，仅显示数字
- 支持 O 卡上的逻辑关系
- 字母数字上 %CHAR 的测试用例
- 支持控件规范关键字 main
- 带有打印机文件中两个参数的 EDTCDE
- 改进了 FullFree RPG 解析

- 增强了动态表格，确保页脚位置正确
- 增加了对使用 ALL 象征常量初始化数字类型的支持
- 改进了对引用相同物理文件的多个 RPG 逻辑文件的处理
- 改进了现代屏幕中的修改字段检测
- 与动态字段模态同步
- 改进了对仅输出有符号数值字段的处理
- 改进了 WORKSTATION I/O 卡支持

横向功能

新功能

- 数据迁移器工具：添加了 ebcdicFilesWith VarcharIn VB 属性，允许在读取字节时考虑 VARCHAR 2 字节的长度
- 实现了一个用于记录错误的通用 API
- 实现 BluAgeErrorDictionaryUtils 和使用通用 API 在 Cobol2Model、R CycleBuilder PG、Definitions2Model 和 FieldsProcessor
- 改进了 SQL 语法以支持不同的隔离子句定义

改进

- 将 Angular 版本升级到 v16
- Angular：将 ajv 版本从 6 升级到 8.9

第三方

- 将 Groovy 升级到版本 2.4.15

版本说明 3.8.0

此版本的 AWS Blu Age Runtime 和 Moderization Tools 侧重于对整个产品进行多项横向增强，以提高其质量和安全性，同时提高缓存性能和统一单个发行版中的命令支持。此版本中的一些关键特征和更改包括：

- 从 Spring 2.5 升级到 Spring 2.7 版本，提高了平台的维护支持、性能和安全性。

- 作为 over-the-counter发行版的一部分，统一了 82 多个 CL 命令支持，以促进以前使用 CL 脚本的现代化应用程序的使用和部署。
- 新的 API 可更好地与 BluSam 数据集搭配使用和交互，例如向托管服务集成导入以及列出数据集元数据信息功能。
- Redis 的性能改进且使用范围扩展，包括集群模式下的可用性、高可用性数据检索、密钥使用的标准化。

有关此版本所包含更改的更多信息，请参阅以下部分。

运行时版本 3.8.0

zOS

新功能

- 将密钥定义作为字符串处理 DynamicFileBuilder
- DFSORT：在 OUTFIL TRAILER1 + DFSORT 语法初始化中增加了对多个项目的支持
- CommonDDUtils 工具：处理入方向流数据中的记录大小
- 索引文件：处理 GENKEY 选项

改进

- 将单独的 jar 中的 BluSAM 加载服务进行了外部化
- 增加了支持设置存储临时文件的位置
- 改进了多节点情况下的共享缓存机制
- 共享缓存使用情况：IDCAMS 验证优化
- 改进了嵌入式选择的 ROWID 注入
- JCL：现在，每个流内作业过程是在不同的 Groovy 文件中生成的
- 确保 IDCAMS JCL 卡上有 card-demo-v 2 个覆盖范围
- BluSam：使用多个实例时避免重复预热
- 减少了缓存水化的内存占用
- Jedis 池配置支持
- 为流添加了行分隔符（用于文件串联时）

- 在 IDCAMS 实用程序中支持 EBCDIC 卡 + 屏蔽注释 (/* ... /)
- 数据库支持查询：在将 level49 转换为 SQL 时支持双字节字符串
- DFSORT 语法：实现 17 条控制语句 + 整合其中 2 条控制语句 (OMIT/INCLUDE)
- 增强 GRAPHIC 列获取 INFUTILB
- 支持读取带有可变大小表格的文件
- Support 支持 ZonedType 带有 nibble 符号的 nibble 符号，其中最后一个字节的第一位是 'E'
- 如果记录与任何 CHANGE 查找常量都不匹配，DFSORT/ICETOOL 会添加对 NOMATCH=(..) 参数的支持
- Redis 集群兼容性
- 根据 groovy 退出代码处理作业状态（失败）
- 改进了 CCICS SYNCPOINT ROLLBACK 支持。
- 预取窗口优化 Redis 缓存使用
- JCL/GROOVY：当 DISP=(,PASS) 时，从上一步的数据集中继承 isRDW 属性
- 处理具有可变大小数组的数据的部分副本

AS400

新功能

- 支持用于显示文件的 I/O 卡
- 支持 DSPF 关键词 ERRMSGID 和 CHKMSGID 的其他消息信息
- 支持在前端屏幕上显示多条错误消息
- 在 gapwalk-cl-command应用程序中添加或改进了对 82 个 CL 命令的支持

改进

- 改进了对提交控制下的删除和读取的支持
- ConvertDate 内置 %dec
- 强制使用 XSS 安全标头
- 提高了 STM 生成的稳健性和一致性（更好地处理：自由格式 rpg 中的连续行、小数部分的逗号、定义/声明中的自由形式块）
- 改进了 DataHolderMapper 生成

- 增加了稳健性并更改了范围 DataAreaFactory
- 改进了 Tab 键上的焦点移动
- 提高了 Jasper 报告生成的性能
- 改进了带有填充 0 的十进制显示
- 改进了 INFDS 中对 ROW/COL 字段的支持
- 改进了对从屏幕上修改字段的支持
- 为生成的报告名称和路径添加了 getter
- 改进了数据队列长度
- 改进了作业队列的自动配置，以便符合 Spring Boot 2.7 中的新标准
- 改进了多个并发会话的工作站更新

横向功能

新功能

- 对已打包数据支持“不容忍无效数据”
- 为列出数据集端点添加了分页/筛选功能

改进

- 增强了 ORACLE 查询转换策略，用于将列与空字符串进行比较
- 使用 DSNTEP 和 INFUTILB 实用程序处理 BLOB DB2。BLOB DB2 现已现代化为 BYTEA 类型的 postgres。
- 改进了对光标最后一项的删除
- 增强了对删除 RRDS 文件的支持
- 提高了 AWS Blusam 的秘密性能
- 改进了 SQL 框架中对数据库连接的处理
- 标准化的 AWS 多数据源密钥管理器密钥
- 性能回归修复
- 改进了以下各项的检查功能 PackedType
- 改进了 LOW-VALUE 的处理 PackedType
- 升级了用于 Cognito 连接的 spring 安全封装

- 未在 DB2 目标数据库上应用 codeshiftpoint 编码和解码

第三方

- Spring Boot 从版本 2.5 升级到 2.7

现代化工具版本 3.8.0

zOS

新功能

- JCL: 使用回车符“\r”处理流

改进

- 改进了日志记录以防止在对带有 ON SIZE ERROR 子句的 DIVIDE 进行现代化时除以零
- JCL：增强了对在过程中调用过程的支持
- 当存在模棱两可的字段时，支持在 FORMATTIME CICS 命令中使用 OF 关键字
- JCL：支持在变量中使用的 Â¥ 字符
- JCL：根据前面的步骤计算 RC
- 使用 PL1 SUBSTR 时比较字节而不是字符串
- 改进了单个源的多维数组的初始化
- 改进了 COBOL 在 IF 块中涉及单个 SQL 查询时的解析

AS400

新功能

- 支持 CL 中嵌套 IF 语句
- 改进了 RPG freeform 中对 ENDDO 语句的支持

改进

- 改进对调节控制级别的支持

- 改进了使用 LIKE 的原型返回值
- 改进了对处理函数 %months、%year、%days 的支持
- 支持整个屏幕的帮助特征
- 处理作为参数传递的象征性 BLANKS
- 使用 "" 运算符对表达式 EVAL 进行了改进
- 处理没有 KEY PHASE 的 START 命令
- 改进了对关键字 LIKEREC 的处理
- 改进了未命名的子字段
- 改进了返回无符号类型的过程
- 改进了对 RESET 操作（免费 RPG）、%CHAR 和 %DEC 内置函数的支持
- 改进了内置函数 %LOOKUPXX
- 改进了在没有原型的过程上对 LIKEDS 关键字的支持
- 处理 Dim 关键字数组类型（VAR、AUTO）
- 改进了对 XFOOT 的支持
- COBOL：改进了对 RENAMES 字段的支持
- CL：支持 while(true) 条件
- 使用 LIKE 关键字改进对独立数组的处理
- 改进了内置函数 %INT
- 改进了 RPG Full Free 解析
- 改进了对关联中数组的支持
- CL2GROOVY：支持 Select 语句
- 改进了 DSPF 关键字“ERRMSGID”
- 改进了使用前导零初始化字节的处理
- 改进了数值字段的 authorizedValues
- 处理 Free form EVAL 语句的头部说明符 H
- CL 到 Groovy：支持 LDA 子字符串
- 改进了对记录 RESET 支持
- 使用参考改进了对 EDTCDE 和 EDTWRD 的处理
- 使用 DDS 字段改进了输入字段映射

- 改进了对于 MOVEA 字符到 IN 数组的支持
- 使用 LIKEDS 关键字改进了原型
- 改进了对 DSPF 关键字 DSPATR 的支持
- 使用 +/- 改进了 D 卡的解析
- 提升了程序调用的稳健性
- 提升了字段解析过程的稳健性

横向功能

改进

- FrontEnd: 模拟 IME 输入的粘贴事件

第三方

- Spring Boot 从版本 2.5 升级到 2.7

版本说明 3.7.0

此版本的 AWS Blu Age 运行时和现代化工具主要包括用于更好地支持命令和实用程序的增强功能、与 AWS Secrets Manager 集成的功能以及新的监控功能。此版本中的一些关键更改包括：

- 现在，多个运行时组件可以使用 AWS Secrets Manager 来增强现代化应用程序的安全设置，这些应用程序主要与实用程序数据源、TS 队列的 Redis、 BluSam 缓存和锁有关。
- 监控允许检索有关资源使用优化和运营管理的事务、批处理和 JVM 指标的端点，例如状态、持续时间、数量等。
- 支持 RPG 中的 IBM MQ 调用的新功能，并增加了 JCL SORT 和 IDCAMS 转换覆盖范围。

有关此版本所包含更改的更多信息，请参阅以下部分。

运行时版本 3.7.0

主题

- zOS
- AS400

- 横向功能

zOS

新功能

- 使用类似 SQL 的语法，改进对实用程序应用程序中涉及的查询的解析。(V7-9401)
- 偏移时处理可变大小索引数组 (V7-9904)
- 支持以 24:00:00 小时格式将 SQL TIME 列插入到 DB2 中 (V7-10023)
- 支持从带有 FOR ROWS 和 ATOMIC 选项的数组中插入 SQL 查询 (V7-10105)
- JCL SORT-使用 IFTHE TranscodeTool N 增强功能以支持 OUTREC (V7-10124)
- JCL SORT – 在 OUTREC 命令中添加对 DATE 关键字的支持 (V7-10125)
- JCL – 增加对数据流过程的支持 (V7-10223)

改进

- 标有“PASS”处置的数据集应在所有作业步骤中可用 (V7-9504)
- 支持 JCL 属性 SCHENV (V7-9570)
- 支持带有 CTLCHAR 选项的 SEND (V7-9714)
- COBOL – 在 ACCEPT 语句中处理不同的行分隔符字符集 (V7-9875)
- 避免多次回滚 (V7-9958)
- 允许使用 MOD 处置附加在 GDG 文件末尾 (V7-10031)
- 优化：putAll 重构 (V7-10063)
- PutAll 重构：添加分页 (V7-10063)
- 使 Jedis 客户端读取超时可配置 (V7-10063)
- UseSsl 支持独立模式 (V7-10114)
- 成功打开文件后支持 EIBDS (V7-10147)
- 在收到文件控制请求后支持 EIBDS (V7-10147)
- 改进了 CICS SYNCPOINT 支持 (V7-10187)
- BluesamRedisSerializer: MetadataPersistensate 存在问题 (V7-10202)
- 支持 Redis AWS Secrets Manager 用于 TS 队列 (V7-10204)
- 在自定义 DD 名称大小时支持 JCLBCICS (V7-10224)

- 在 IDCAMS DELETE 语句中添加对绝对路径的支持 (V7-10308)

AS400

新功能

- 实现 AS400 屏幕帮助特征 (V7-9673)

改进

- INFDS 中的记录数量 (V7-9377)

横向功能

新功能

- 支持 EC2 上的 Runtime 以向亚马逊发送日志 CloudWatch (D87990246)
- 添加了新的端点，用于检索有关批处理、事务和 JVM 的指标 (D88393832)

改进

- 支持数据源 AWS Secrets Manager 用于实用程序 pgm (V7-9570)
- 添加了 Db2 对 DSNUTILB DISCARD 的支持 (V7-9798)
- 支持写入记录器而不是默认 SYSPRINT 和 SYSPUNCH 文件中的默认系统输出流 (V7-10098)
- 在 AWS Sec BluSam rets Manager 中支持 Redis 缓存和锁定连接属性 (V7-10238)
- 支持 Db2 XA AWS 密钥上的 SSL 连接 (V7-10258)
- 更新了 IDCAMS REPRO 和 VERIFY 的元数据 (V7-10281)
- 改进了 IDCAMS Abend 返回码管理 (V7-10307)

现代化工具版本 3.7.0

主题

zOS

新功能

- PLI – 改进了数组截面和二维数组的赋值 (V7-9830)

AS400

新功能

- 处理控制水平指示器 (V7-9227)
- 支持 EXTNAME 参数 *INPUT (V7-9897)
- 增强的 Goto 重写：支持位于 SELECT OTHER 语句中的标签 (V7-9973)
- 支持 REFSHIT DSPF 关键字 (V7-10049)

改进

- 改进了对文件描述关键字 EXTIND(*INUx) 的处理 (V7-7404)
- 改进了 SQLDDS 文件转换 (V7-7687)
- 不再为 AS400 文件生成文件对象 (V7-9062)
- 改进了对文件描述关键字 EXTDESC 的处理 (V7-9268)
- 改进了对内置 %CHAR 的处理 (V7-9311)
- 改进了对没有 SFLEND 的最后一条记录的向下翻页支持 (V7-9322)
- 改进了对前缀数据结构的支持 (V7-9436)
- 支持使用 %SIZE 定义的维度 (V7-9472)
- 支持对使用双引号声明的 PF 字段名进行处理 (V7-9557)
- 改进了文件操作，不区分大小写 (V7-9785)
- 支持初始化为 *USER 的字段 (V7-9806)
- 支持 AS400 中的 COMP 类型 (V7-9840)
- 改进了 COBOL400 解析开启（不是）InvalidKey (V7-9922)
- 改进了对 SCAN 操作的处理 (V7-9971)
- 改进了对 GOTO 操作码的支持 (V7-9973)
- 改进了对 EXCEPT 操作的处理 (V7-9977)

- 改进了对前缀的支持 (V7-10000)
- 支持 RPG 中的 MQ 调用 (V7-10007)
- 改进了内置 %LOOKUP（键控数组数据结构）(V7-10022)
- 支持 Close *All 操作 (V7-10036)
- 支持 UPDATE AS ROW CHANGE SQLDDS 语句 (V7-10051)
- 改进了对字面值类型 Long 的处理 (V7-10073)
- 改进了 RPG 语法（使用关键字 INZ 作为子例程的名称）(V7-10074)
- 改进了 RPG 语法，支持小数部分为空的数值 (V7-10077)
- 改进了对 CL 和外部文件之间共享的字段的支持 (V7-10081)
- 改进了对 DDS 条件指示器的支持 (V7-10084)
- 通过 COBOL 程序支持 DDS 二进制类型 (V7-10100)
- 改进了链接的名称冲突 (V7-10109)
- 支持混合主过程和导出过程 (V7-10112)
- 改进了对子过程 DataStructure 中的支持 (V7-10113)
- 改进了对 CLEAR 的支持 (V7-10126)
- 改进了对 DO 回路的支持 (V7-10134)
- 在 Full-Free RPG 中支持 SQLTYPE (V7-10151)
- 改进了 DDS 关键字条件的解析 (V7-10155)
- 改进了 DSL 生成 (V7-10163)
- 改进了条件为二进制表达式时的 processIndicators。(V7-10164)
- 改进了 Else 条件中使用的 GOTO (V7-10168)
- 支持 DSPF 中的时间和时间戳类型 (V7-10173)
- 改进了 DDS 连续行的解析 (V7-10183)
- COBOL 支持 RENAMES FLD OF RECORD (V7-10195)
- 改进了 DSPF 字段上的条件指示器解析 (V7-10221)
- 支持解析 DDS 关键字 NOALTSEQ (V7-10288)
- 支持“帮助”菜单和隐藏字段 (V7-10314)
- 改进了 DSPF 帮助关键字完整性检查 (V7-10328)
- 不再传播 Ref 字段上的所有关键字 (V7-10347)

横向功能

新功能

- 数据迁移器 – 处理 CLOB 数据 (V7-9665)

改进

- 通过 (V7-10225) 将 JCL 属性 SCHENV 从 JOB 传播到 PROC GROOVY 定义 JobContext
- FrontEnd -在没有边框的情况下调整窗口大小 (V7-10358)

版本说明 3.6.0

此版本的 AWS Blu Age 运行时和现代化工具为 zOS 和 AS400 传统迁移提供了新功能，主要面向扩展 CICS 支持机制、补充 JCL 功能、优化并发和高容量功能的性能以及添加功能。 multi-data-source 此版本中的一些关键更改包括：

- 增强 JCL 动态文件处理、扩展当前语句和连接数据集的管理、在单个块中执行多个语句以及将数据从批处理传输到程序。
- 增强了对多个 CICS 命令的支持，包括对多种 CICS 资源类型的查询。
- 使用 Blu Age 运行时实用程序时能够使用不同的数据库，非常适合业务数据分布在多个来源的场景。

有关此版本所包含更改的更多信息，请参阅以下部分。

运行时版本 3.6.0

主题

zOS

新功能

- JCL- DynamicFileBuilder -增强的文件句柄管理 (V7-9408)
- 增强了在调用 INFUTILB UNLOAD 实用程序时对某些内置 SQL DB2 函数的格式转换 (V7-9554)
- 增强了 PLI 多维数组赋值 (V7-9592)
- 将 sysout 重定向到文件 (V7-9992)

改进

- 为 DB2 RDBMS 添加存储过程触发 (V7-9155)
- SORT 处理结果转换为 PDF 格式 (V7-9286)
- JCL/GROOVY – 增强 REPRO 语句，以便支持虚拟数据集 (V7-9424)
- 改进了 CICS UNLOCK 支持 (V7-9606)
- 处理 Union 的默认值大小 (V7-9648)
- JCL/GROOVY 处理拼接数据集中不同的终止/处置 (V7-9653)
- 使 blusam 数据集的 pageSize 可配置 (V7-9680)
- DSNUTIL – 允许在 DB2LUW 中加载 24:00:00 作为有效时间 (V7-9697)
- 支持 NumberUtils .ne ()/ NumberUtils.eq () 中的高值 (0xff) 比较 (V7-9731)
- JCL/GROOVY – 支持在 IDCAMS IF-THEN-ELSE 子句中使用 DO... THEN 关键字，以便在单个块中执行多个语句 (V7-9750)
- 在 JHDB 之外调用的 JHDB 程序无效 (V7-9782BatchRunner)
- 支持 SORT OUTFIL 控制卡中的空格字符 (V7-9808)
- 改进了 CICS READ PREV 支持 (V7-9845)
- 改进了对数据集索引的并发访问 (V7-9864)
- 改进了 CICS REWRITE 支持 (V7-9873)
- COBOL – 支持在 ACCEPT 语句中使用多行 SYSIN 将数据从批处理 (JCL) 传递到程序 (COBOL) (V7-9875)
- Groovy-更好地处理文件创建步骤 (V7-9876) ConcatenatedFileConfiguration
- IDCAMS 实用程序 – 处理 DEFINE PATH 语句 (V7-9878)
- SORT BUILD – 调整 TRAN 选项并处理隐式空白 (V7-9925)
- 通过 GENERIC 选项支持改进 CICS DELETE (V7-9939)
- 改进了 CICS STARTBR 和 ENDBR 支持 (V7-9952)
- 提高了并发访问的关闭性能 (V7-9953)

- 改进了启动时的文件状态处理 (V7-9991)
- Groovy-允许在 (V7-10012) 上调用 getDisposition getNormalTermination ConcatenatedFileConfiguration ()/getAbnormalTermination()/()

AS400

新功能

- 支持 COMMIT 关键词上的外部指示器 (V7-6035)
- 写入 SFLCTL 后重置 ReadC 循环 (V7-8061)
- 支持 CALL 中的 LR 指示器 (V7-9250)
- 添加新类型的动态字段（拆分），以便处理多行上的输入字段 (V7-9370)
- 支持主文件/辅助文件 (V7-9390)
- 目前，提交作业时，本地数据区会传递给被调用的作业 (V7-9775)
- QTEMP 支持数据区域，支持数据区域值创建。(V7-9916)
- 提交控制 – 支持启用/禁用提交控制 (V7-9956)
- 支持 COMMIT 关键词上的外部指示器

改进

- 改进 0 值显示和 EDTWRD (V7-8933)
- 支持 DSPF 关键词“CHKMSGID”(V7-9125)
- 在批处理终止时提交 SQL 事务 (V7-9232)
- 改进对字段和数据结构的关键字 EXPORT 和 IMPORT 的支持 (V7-9265)
- Support 中支持小写 DateHelper 字母 (V7-9461)
- 支持将 *CYMD 转换为 *ISO（数字）(V7-9488)
- 改进内置 %len 针对变化字段（表达式的左侧和右侧）的处理 (V7-9733)
- 改善对内置函数 '%LOOKUPXX' XX（“LE”、“LT”、“GE”、“GT”）的支持 (V7-10064)

横向功能

新功能

- CICS – 改进选项状态的查询事务 (V7-9712)
- JCL – 使用系统输出文件改进 sysprint 的负载 (V7-9797)
- CICS – 改进 INQUEU TSQUEUE (V7-9823)
- CICS – 改进选项 userid 的查询终端 (V7-9906)

改进

- 改进与空白比较的处理方式 (V7-8047)
- 改进 Jics 和 BluSam (V7-8847) 的日志记录
- 支持 BMS 扩展属性 SOSI 和动态字段的编程符号 F8 (V7-8857)
- 处理程序参数中的缓冲区溢出 (V7-9138)
- 改进 Blusam 锁注册表的线程写入并发性 (V7-9505)
- 支持 Utility-pgm 的多个数据源配置 (V7-9570)
- Blusam 记录级别仅锁定模式 (V7-9626)
- 确保元数据在服务器重新启动后保持不变 (V7-9748)
- 改进发生异常时的 DAO 清理（浏览器关闭）(V7-9790)
- 支持 INFUT DummyFile ILB SYSPUNCH (V7-9799)
- 在 NumericEditedType (V7-9935) 上增强对负值的支持

现代化工具版本 3.6.0

主题

- zOS
- AS400
- 横向功能

zOS

新功能

- JCL – 增强过程结束时的日志记录 (V7-8509)
- PL1-增强数据类型的袋子生成 PakedLong (V7-8917)

- JCL – 增强在文件包含“结束”标记的情况下过程结束时的日志记录功能 // (V7-9509)
- PL1 – 增强对定点和 SYSIN 流的 GET EDIT 支持 (V7-9593)
- DB2 – 增强对 VARGRAPHIC DB2 类型的支持 (V7-9809)
- CICS – 提高选项 LOGMESSAGE 的命令查询安全性 (V7-9969)
- PL1 – 改善内置 Charg/Chargraphic 的包生成 (V7-9989)

改进

- PL1 – 增强对 INCLUDEX 关键字的支持 (V7-9588)
- PL/I – 将 CHARGRAPHIC 关键字作为任何方法调用的有效参数处理 (V7-9589)
- 改进使用特定字符 (@ # $ §) 命名时 PL1 主机变量的解析。(V7-9654)
- COBOL – 支持 C01...C12 和 S01...S05 关键字作为解析步骤 WRITE ADVANCING 语句的参数 (V7-9669)

AS400

新功能

- 支持分析器中的 SQL-DDS 转换 (V7-7687)
- 自动检测 SQL-DDS 文件 (V7-7687)
- 实现 SQL-DDS 预处理 (V7-7687)
- 支持 ALIGN 关键字 (V7-9254)
- 支持 ExtName DSPF 和多维阵列 (V7-9663)
- COBOL WRITE 上的 Support InvalidKey 声明 (V7-9793)

改进

- 改进了 TESTB 操作码 (V7-8865)
- 改进了焦点上对 DECFMT 的支持 (V7-8933)
- 处理 MOVE 生成的指示器 (V7-9224)
- 改进了对字段和数据结构的关键字 TEMPLATE 的支持 (V7-9278)
- 改进了 LIKEDS (使用 LIKEDS 定义的 DS 会自动满足要求) (V7-9302)
- COBOL – 改进指示器结构生成 (V7-9423)

- 原型中的常量参数不是只读的 (V7-9437)
- 使用编辑代码“Y”改进 EDTCDE 关键字 (V7-9443)
- 支持在 PSDS 和 INFDS 中生成 *ROUTINE 字段 (V7-9487)
- 将重写字段 XXX 改为独立字段（重写时默认值会丢失）(V7-9522)
- 改进了对 DSPF 关键词的支持 (V7-9658)
- 处理二进制的零默认值 (V7-9666)
- 支持隐式指针 (V7-9719)
- 使用一个参数改进对内置调用 %size 的处理 (V7-9730)
- 改进内置调用 (%ELEM) 中对数据结构引用的处理 (V7-9736)
- 改进定义规范中带有 LIKE 引用的字段的带符号长度的处理 (V7-9738)
- 改进 REWRITE (V7-9791)
- 改进从 DDS 文件生成索引的方法 (V7-9803)
- 提高具有无效数值的映射器的稳健性 (V7-9813)
- 改进 SQLModel 和 allIndexes 文件的生成 (V7-9818)
- 改进限定 DS 支持 (V7-9863)
- 改进对 LOOKUP 的支持（在参数中使用类似 DS 的独立字段）(V7-9961)
- 在指示器上改进 LIKE (V7-9985)
- 处理 MVR 上生成的指示器 (V7-9224)
- 支持带波浪号的字符 N (V7-10021)
- 改进从 SQLDDS 传统文件生成现代 DDL 文件 (V7-10067)

横向功能

新功能

- 使用 yml 属性自定义资源位置 (D88816105)
- COBOL – 支持 EXIT PERFORM 语句在不使用 GO TO/PERFORM ... THROUGH 的情况下退出内联 PERFORM (V7-9582)
- 指定要在全局元数据中考虑的默认传统编码。(V7-9883)

改进

- 改进掩码生成 (V7-9602)
- 改进上下文预热 (V7-9621)
- 确保 Charset CUSTOM930 线程安全。(V7-9674)
- 改进 MOVEA (V7-9773)

版本说明 3.5.0

此版本的 AWS Blu Age 运行时和现代化工具为 zOS 和 AS400 传统迁移提供了新功能，主要面向数据集和消息优化，以及扩展的 Java 功能作为转换过程的资产。此版本中的一些关键更改包括：

- 除了先前存在的 groovy 脚本特征外，还能够将 CL 程序迁移到 Java，可促进其与其他现代化程序的集成，并通过统一生成的编程语言来简化客户的学习曲线。
- 借助新的批量数据特征，缩短时间并优化 Redis 中数据集加载的性能。
- 能够在作业步骤中操作和传递数据集，实现传统数据集行为的现代化。
- 扩展了 SQL 迁移来支持 VB 输入文件，Java 11 简化了迁移。
- 多种新机制可更快地与 IBM MQ 集成，包括额外的标头、扩展的 GET/PUT 支持和队列元数据自动检索。
- REST 端点用于数据集元数据和从 S3 存储桶导入数据集。

有关此版本所包含更改的更多信息，请参阅以下部分。

运行时版本 3.5.0

主题

- zOS
- AS400
- 横向功能

zOS

新功能

- JCL SORT – 处理新的关键字叠加 (V7-9409)
- ZOS COBOL – 增强对浮动字符的支持 (V7-9404)

- RedisJicstsQueue 到 RedisTemplate & 的端口 ListOperations (V7-9212)
- ZOS JCL-如果通过文件目录定义，则使用文件目录增强临时目录的路径 UserDefinedParameters (V7-9012)
- 使用全部（所有数组项目）处理函数 ORD-MAX (V7-9366)
- 在 Redis 中存储 TS 队列时，使用人类可读的带前缀密钥 (V7-9212)
- 增加为 blusam API 获取数据集端点
- JCL – 添加对名称包含 # 等特殊字符的批处理作业的支持 (V7-9136)
- TSMode 提取现在可以按需稳健执行 (V7-9212)

改进

- LNK 文件中支持非版本控制 INCLUDE (V7-6022)
- MQ – 增强编码支持 (V7-9652)
- 改进对不同字符类型的双字节或混合字符集的支持 (V7-9596)
- JCL – 支持 IDCAMS delete NONVSAM 语句中的 filesDirectory 配置 (V7-9609)
- 支持采用批量模式从文件加载 ESDS 和 RRDS 数据集 (V7-8639)
- 在输入模式下处理空 ESDS 的打开。(V7-9287)
- 支持 ORD/UNORD 缩写，增强 DEFINE CLUSTER 语句 (V7-9451)
- BluSam Redis 锁性能改进 (V7-8639)
- 增强 DEFINE CLUSTER 语句，以便支持 DATA () 参数作用域中提供的 RECORDSIZE (V7-9337)
- 添加了 DEFINE CLUSTER 语句对 BUFFERSPACE/UNIQUE 属性的支持 (V7-9419)
- 改进了可变长度记录数据集的 BluSam 读取操作。(V7-9391)
- CICS 地址正确地将缺失的 CWA 表示为空 (V7-9491)
- 移除结尾锁定时不必要的写入 (V7-8639)
- 处理缓存中的 Redis 缓存模板注入 (V7-9510)
- 正确解码 BPXWDYN 参数 (V7-9417)
- 改进 LISTCAT 的导出消耗 (V7-9201)
- BluSam TS 队列名称中支持不可打印字符 (V7-9212)
- 处理地图集为空的字段的接收地图构建 (V7-9486)
- 改进动态访问模式的 BluesamRelativeFile 删除和重写操作。(V7-8989)

AS400

新功能

- 添加通过标准 DS/STM 旋转以 Java 程序生成 CL 文件的特征 (V7-9427)
- 支持 ADD 模式下的输入文件 (V7-9378)
- 改进了排序顺序和检索管理以支持 cl 命令 OPNQRYF（打开查询文件），并在中添加了对 SHARE 参数的支持。 OverrideItem(V7-9364)

改进

- 在 (V7-8061) 上支持 SFLNXTCHG UpdateSubfile
- 运行 CL 命令时修改 CL 上下文的范围 (V7-9624)
- 处理程序 BPXWDYN 的返回码 (V7-9417)
- 清除本地显示器。(V7-9624)
- 支持 DSPF 关键字 RTNCSRLOC (V7-9389)
- setOnGreaterOrEqual() 未设置等于 1 (V7-9342)
- 开启更新字段缓存 UpdateSubfileRecord (V7-9376)
- 改进 SFLNXTCHG 支持 (V7-8061)

横向功能

新功能

- 忽略文字图形字符串上的 G 前缀。(V7-9420)
- ZOS COBOL – 增强对某些特殊结构的 Fiedl.initialize() 支持 (V7-9485)
- 允许异步初始化上下文，以便改善程序启动性能 (V7-9446)
- SQL 明确释放打开的准备语句和 ResulSet。(V7-9422)
- 增强 JMS MQ – 支持 MQ PUT 使用 MQRFH2 (V7-7085) - 支持默认队列管理器 (V7-9400)
- SQL 管理：启用 SET 命令参数的 Lambda 转换 (V7-9492)
- ZOS MQ JMS – 添加对 MQCOMIT 和 MQBACK 的支持 (V7-9399)
- ZOS IBMMQ – 增强对 MQINQ 的支持 (V7-9544)
- 使用双字节编码时，使用字节而不是字符串处理 CONCAT 操作。(V7-8932)
- ZOS IBMMQ – 使用选项 SET_ALL_CONTEXT 增强对 PUT 命令的支持 (V7-9544)

改进

- 使用 $ 字符处理 gdg 文件名 (V7-9066)
- 当上一个 SQL 语句成功时，SQL 诊断将 1 作为 NUMBER 子句返回。(V7-9410)
- 为长度不为空的字段设置大纲 (V7-7536)
- 支持内置 PL1 图形功能 (V7-9245)
- MQ – 添加对 MQGMO 字段设置版本的支持 (V7-9500)
- JMS MQ GET – 消息返回的 dataLength 改进 (V7-9502)
- 在 ROWSET 上下文中使用提取的项目数设置 sqlerrd(3)。(V7-9371)

现代化工具版本 3.5.0

主题

- zOS
- AS400
- 横向功能

zOS

新功能

- ZOS PLI – 支持二进制表达式赋值中使用星号索引 (V7-9178)
- JCL to BatchScript -“//” 表示任务执行结束 (V7-9304)
- ZOS PLI – 增强对浮动字符和数字签名编辑类型的支持 (V7-8982)
- COBOL – 支持内置 SUM 函数 (V7-9367)
- JCL – 可选，在空语句之后注释死码 (//) (V7-9202)
- JCL – 支持在条件语句中使用运算符“|”(V7-9499)
- PL/I – 预处理步骤中用于防止解析异常的预编译指令的注释 (V7-9507)

改进

- 使用分隔符处理流定义 (V7-9615)
- 改进 LISTCAT 导出处理能力。(V7-9201)

- PL/I – 增强对隐式“空”参数的支持 (V7-9204)

AS400

新功能

- 支持 DDS 关键词 CONCAT (V7-9439)
- 重构为 DSPF 关键字生成的 Java 代码。(V7-7700)
- 支持在数据结构定义中的字段上使用不同的关键字 (V7-9029)

改进

- 改进逻辑关系 AND/OR 的解析 (V7-9352)
- COBOL 改进 vo 和 dsEntity 之间的映射 (V7-9449)
- 如果聚焦数字输入，则显示空值 (V7-9374)
- SQL 声明游标中的局部变量 (V7-9456)
- 空的 DS 的作用域问题 (V7-9466)
- 在解析之前截断 col 80 之后的行 (V7-9632)
- 改进定义规范中关键字（DIM、LIKE 等）中字段引用和内置调用的处理 (V7-9358)
- 支持 SQL 注释 (--) (V7-9632)
- FullFree 正在解析，键入日期/时间/时间戳 (V7-9542)
- 包括 FullFree 解析中的 SQLCA (V7-9333)
- 改进对控制等级的支持。(V7-9610)
- 处理 DS 与 *BLANKS 的比较 (V7-9668)
- 改进对 DDS 中多个指示器的支持 (V7-9318)
- 改进对多个 DSPF 程序的支持（V7-9657）
- 使用 LIKE（类似数据结构和数组中类似数据结构的情况）改进字段处理 (V7-9213)
- Free RPG，处理文本字符串的延续 (V7-9686)
- 改进对程序结束记录的支持 (V7-9452)
- 支持 CALL 语句中的 LINKINGE 短语。(V7-9685)
- CASXX 操作代码（不带 CASXX 组的 CASBB）(V7-9357)

- 改进 FullFree角色扮演游戏解析 (V7-9457)
- 内置 %LEN 不支持 DS 作为参数 (V7-9267)
- 改进因子 2 为 *ALL'X...' 时的 MOVEA (V7-9228)
- 支持使用 RENAME 字段进行分配 (V7-9385)

横向功能

新功能

- SQL Migrator 工具：在 ebcdic 加载步骤中为可变记录长度添加 OID 选项。(V7-9380)
- SQL Migrator 工具：在 OID 选项上支持 Java 11 (V7-9599)

改进

- 改进对嵌套数组的支持 (V7-9595)
- 将 Â¬ 字符替换为 ! (如果原始编码支持 Â¬) 。(V7-9465)
- JCL – 支持传递正常终止，以便在作业步骤之间共享数据集 (V7-9504)
- 处理 VARCHAR 和可为空的数据库列类型时，对 ORACLE 上的列定义应用 ON NULL。(V7-9681)
- 提高 Spring 注入合规性 (V7-9635)

AWS 蓝光时代运行时概念

了解 AWS Blu Age Runtime 的基本概念可以帮助您了解如何通过自动重构实现应用程序的现代化。

主题

- AWS Blu Age 运行时高级架构
- AWS 现代化应用程序的 Blu 时代结构
- 数据简化器

AWS Blu Age 运行时高级架构

作为将传统程序现代化为 Java 的 AWS Blu Age 解决方案的一部分， AWS Blu Age Runtime 通过提供遗留结构和程序代码组织标准化的库，为现代化应用程序提供了统一的、基于 REST 的入口点，并为此类应用程序提供了执行框架。

这种现代化的应用程序是 AWS Blu Age 自动重构过程的结果，该过程用于将大型机和中端程序（在以下文档中称为 “旧版”）现代化为基于 Web 的架构。

AWS Blu Age Runtime 的目标是再现遗留程序的行为（同功能性）、性能（在程序执行时间和资源消耗方面），以及便于Java开发人员维护现代化程序，尽管使用熟悉的环境和习语，例如 tomcat、Spring、getters/setter、fluent API...

主题

AWS 蓝光时代运行时组件

AWS Blu Age 运行时环境由两种组件组成：

- 一组 java 库（jar 文件），通常称为“共享文件夹”，提供传统结构和语句。
- 一组 Web 应用程序（war 文件），其中包含基于 Spring 的 Web 应用程序，为现代化程序提供一组通用的框架和服务。

以下部分详细介绍了这两个组件的作用。

AWS 蓝光时代图书馆

AWS Blu Age 库是一组 jar 文件，存储在添加到标准 tomcat 类路径的`shared/`子文件夹中，以便所有现代化的 Java 程序都可以使用它们。这类库旨在提供在传统开发环境中常见但在 Java 编程环境中非原生且不容易实现的特征。这些特征尽可能以 Java 开发人员熟悉的方式公开（getters/setter、基于类、fluent API）。一个重要的例子是数据简化器库，该库为 Java 程序提供了传统的内存布局和操作结构（在 COBOL、PL1 或 RPG 语言中出现）。这些 jar 是传统程序生成的现代化 Java 代码的核心依赖项。有关数据简化器的更多信息，请参阅数据简化器。

Web 应用程序

Web 应用程序存档 (WAR) 是将代码和应用程序部署到 tomcat 应用程序服务器的标准方法。作为 AWS Blu Age 运行时的一部分提供的执行框架旨在提供一组执行框架，再现传统环境和事务监视器（JCL batch、CICS、IMS...）以及相关的必需服务。

最重要的是 gapwalk-application（通常缩写为 “Gapwalk”），它提供了一组基于 REST 的统一入口点来触发和控制事务、程序和批处理执行。有关更多信息，请参阅 AWS 蓝光时代运行时 API。

此 Web 应用程序分配 Java 执行线程和资源，以便在现代化程序所面向的环境中运行这些程序。以下部分通过举例详细介绍了此类重现环境。

其他 Web 应用程序向执行环境（更确切地说，向下文描述的“程序注册表”）中添加了仿真传统程序可用的和可从旧程序调用的程序的程序。其中重要的两类为：

- 操作系统提供的程序仿真：JCL 驱动的批处理特别需要能够调用各种文件和数据库操作程序作为其标准环境的一部分。示例包括 SORT/DFSORT 或 IDCAMS。为此，提供的 Java 程序可以重现此类行为并且可被调用（使用与传统程序相同的方式）。
- “驱动程序”，由执行框架或中间件作为入口点提供的专用程序。例如，在 IMS 环境中执行的 COBOL 程序依赖 CBLTDLI 来访问与 IMS 相关的服务（IMS 数据库、通过 MFS 的用户对话等）。

程序注册表

为了参与和利用这些结构、框架和服务，从传统程序现代化得到的 Java 程序遵循AWS 现代化应用程序的 Blu 时代结构中所述的特定结构。启动时， AWS Blu Age Runtime 会将所有此类程序收集到通用的 “程序注册表” 中，以便之后可以调用（并相互调用）。程序注册表提供了松耦合和分解的可能性（因为相互调用的程序不必同时进行现代化）。

执行环境

经常遇到的传统环境和编排如下：

- JCL 驱动的批处理一旦现代化为 Java 程序和 Groovy 脚本，即可以同步（阻塞）或异步（分离）的方式启动。在后一种情况下，可以通过 REST 端点监控其执行情况。
- AWS Blu Age 子系统通过以下方式提供类似于 CICS 的执行环境：
  - 一个入口点，用于启动 CICS 事务和运行相关程序，同时遵循 CICS 的“运行级别”编排；
  - 资源定义的外部存储；
  - 一组同构的 Java fluent API，用于重现 EXEC CICS 语句；
  - 一组可重现 CICS 服务的可插拔类，例如临时存储队列、临时数据队列或文件访问（通常有多种实现可用，例如适用于 Apache Flink 的亚马逊托管服务、Amazon Simple Queue Service 或适用于 TD 队列的 RabbitMQ）；
  - 对于面向用户的应用程序，BMS 屏幕描述格式已现代化为 Angular Web 应用程序，并支持相应的“伪对话”对话框。

- 同样，另一个子系统提供基于 IMS 消息的编排，并支持采用 MFS 格式的 UI 屏幕现代化。
- 此外，第三个子系统允许在类似 iSeries 的环境中执行程序，包括对 DSPF（显示文件）指定的屏幕进行现代化。

所有这些环境都建立在常见的操作系统级服务之上，例如：

- 仿真传统内存分配和布局（数据简化器）；
- 基于 Java 线程重现 COBOL“运行单元”执行和参数传递机制（CALL 语句）；
- 仿真扁平化、串联、VSAM（通过 Blues am 库集）和 GDG 数据集组织，
- 访问数据存储，例如 RDBMS（EXEC SQL 语句）。

无状态和会话处理

AWS Blu Age Runtime 的一个重要功能是在执行现代化程序时启用高可用性 (HA) 和水平可扩展性场景。

该特征基于无状态，其重要的示例是 HTTP 会话处理。

会话处理

Tomcat 是基于 Web 的，实现这一点的一个重要机制是 HTTP 会话处理（由 tomcat 和 Spring 提供）和无状态设计。这类无状态设计基于以下几点：

- 用户通过 HTTPS 进行连接；
- 应用程序服务器部署在负载均衡器后面；
- 当用户首次连接到应用程序时，将对其进行身份验证，并且应用程序服务器会创建一个标识符（通常在 Cookie 中）；
- 此标识符将用作将用户上下文保存到外部缓存（数据存储）和从外部缓存（数据存储）检索用户上下文的密钥。

Cookie 管理由 AWS Blu Age 框架和底层 tomcat 服务器自动完成，这对用户是透明的。用户的互联网浏览器将自动对此进行管理。

Gapwalk Web 应用程序可以将会话状态（上下文）存储在各种数据存储中：

- ElastiCache 适用于 Redis 的 Amazon

- Redis 集群
- 内存映射（仅适用于开发和独立环境，不适用于 HA）。

高可用性与无状态

更笼统地说， AWS Blu Age 框架的设计原则是无状态性：重现遗留程序行为所需的大多数非临时状态都不存储在应用程序服务器中，而是通过外部常见的 “单一事实来源” 共享。

此类状态的示例有 CICS 的临时存储队列或资源定义，而这些状态的典型外部存储是与 Redis 兼容的服务器或关系数据库。

这种设计与负载均衡和共享会话相结合，使大多数面向用户的对话（OLTP，“在线事务处理”）都可以在多个“节点”（此处为 tomcat 实例）之间分发。

实际上，用户可以在任何服务器上执行事务，无需关注下一个事务调用是否在另一台服务器上执行。然后，当生成新服务器时（由于自动扩缩或者为了替换运行不正常的服务器），我们可以保证任何可访问且运行良好的服务器都能按预期运行事务，并获得正确的结果（预期的返回值、数据库中的预期数据更改等）。

AWS 现代化应用程序的 Blu 时代结构

本文档详细介绍了现代化应用程序的结构（使用 AWS 大型机现代化重构工具），以便开发人员可以完成各种任务，例如：

- 顺畅地导航到应用程序。
- 开发可从现代化应用程序中调用的自定义程序。
- 安全地重构现代化应用程序。

我们假设您已经掌握了以下方面的基础知识：

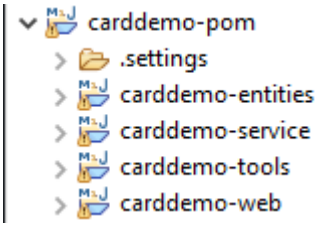
- 传统的常见编码概念，例如记录、数据集及其对记录（索引、顺序）、VSAM、运行单元、jcl 脚本、CICS 概念等的访问模式。
- 使用 Spring 框架进行 java 编码。
- 为方便阅读，我们在整篇文档中使用了`short class names`。有关更多信息，请参阅AWS Blu Age 完全限定名称映射检索 AWS Blu Age 运行时元素的相应完全限定名称和第三方完全限定名称映射检索第三方元素的相应完全限定名称。
- 所有工件和样本均取自 COBOL/ CardDemo CICS 应用程序样本的现代化过程输出。

主题

构件组织

AWS Blu Age 现代化应用程序打包为 java Web 应用程序 (.war)，您可以将其部署在 JEE 服务器上。通常，服务器是一个嵌入了 AWS Blu Age Velocity 运行时的 Tomcat 实例，该运行时目前建立在 Springboot 和 Angular（用于用户界面部分）框架之上。

war 聚合了几个组件构件 (.jar)。每个 jar 都是专用 java 项目的编译（使用 maven 工具）结果，该项目的元素是在现代化过程产生的。



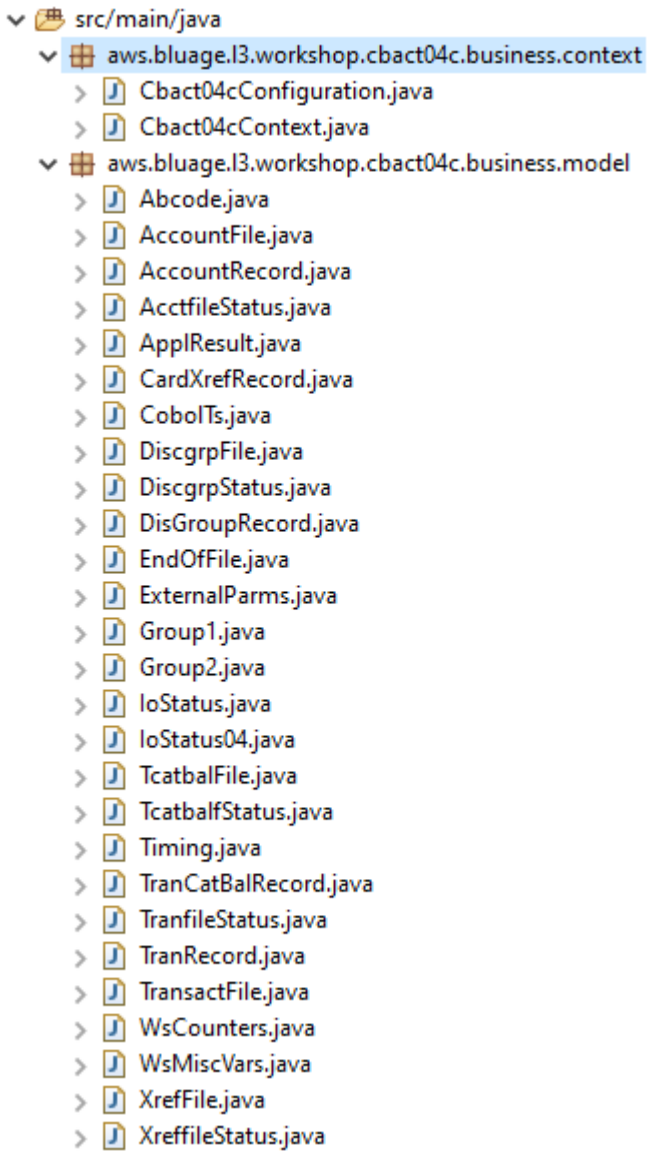
基本组织依赖以下结构：

- 实体项目：包含业务模型和上下文元素。项目名称通常以“-entities”结尾。通常，对于给定的传统 COBOL 程序，这与 I/O 部分（数据集）和数据部分的现代化相对应。您可以有多个实体项目。
- 服务项目：包含传统业务逻辑现代化元素。通常是 COBOL 程序的过程部分。您可以有多个服务项目。
- 实用程序项目：包含其他项目使用的共享常用工具和实用工具。
- Web 项目：包含 UI 相关元素的现代化（如果适用）。不用于仅限批处理的现代化项目。这些 UI 元素可能来自 CICS BMS 映射、IMS MFS 组件和其他大型机 UI 源。您可以有多个 Web 项目。

实体项目内容

> **Note**
>
> 以下描述仅适用于 COBOL 和 PL/I 现代化输出。RPG 现代化输出基于不同的布局。

在进行任何重构之前，实体项目中的包组织与现代化程序相关联。您可以通过几种不同的方法来实现这一点。首选方法是使用重构工具箱，该工具箱在您触发代码生成机制之前运行。这是一项高级操作，BluAge 培训中将对此进行了解释。有关更多信息，请参阅重构研讨会。使用这种方法时，您可以保留后续重新生成 Java 代码的能力，例如可以从将来的进一步改进中受益。另一种方法是使用您希望应用的任何 java 重构方法直接在生成的源代码上进行常规的 java 重构，但需自行承担风险。



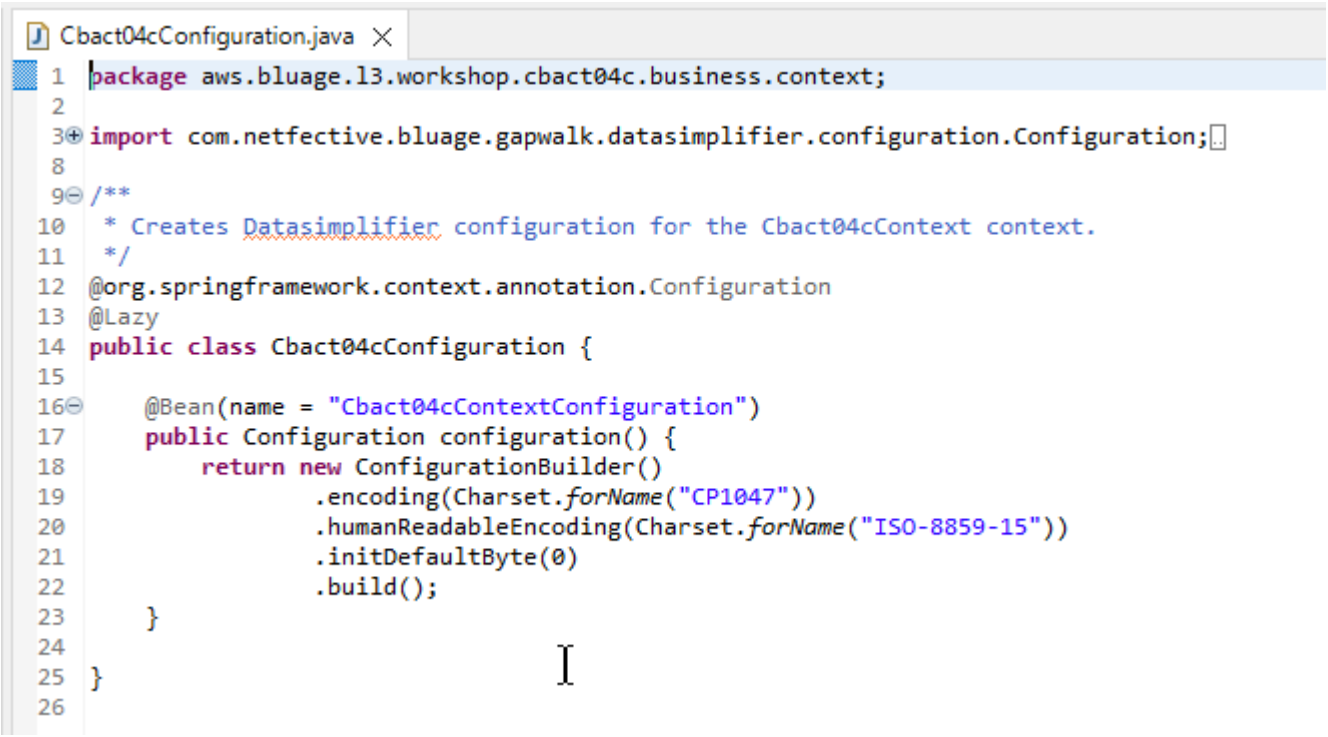
项目相关类

每个现代化程序都与两个包相关：business.context 包和 business.model 包。

- *base package*.*program*.business.context

  business.context 子包包含两个类，即一个配置类和一个上下文类。

- 程序的一个配置类，其中包含给定程序的特定配置详细信息，例如用于表示基于字符的数据元素的字符集、用于填充数据结构元素的默认字节值等。类名称以“Configuration”结尾。该类标有 `@org.springframework.context.annotation.Configuration` 注释，并且包含一个必须返回正确设置的 `Configuration` 对象的方法。



- 一个上下文类，用作程序服务类（见下文）与模型子包（见下文）中的数据结构 (`Record`) 和数据集 (`File`) 之间的桥梁。该类名称以“Context”结尾并且是 `RuntimeContext` 类的子类。

Cbact04cContext.java

```java
@Component("aws.bluage.l3.workshop.cbact04c.business.context.Cbact04cContext")
@Import({
    aws.bluage.l3.workshop.cbact04c.business.model.TcatbalFile.class
,   aws.bluage.l3.workshop.cbact04c.business.model.XrefFile.class
,   aws.bluage.l3.workshop.cbact04c.business.model.DiscgrpFile.class
,   aws.bluage.l3.workshop.cbact04c.business.model.AccountFile.class
,   aws.bluage.l3.workshop.cbact04c.business.model.TransactFile.class
})
@Lazy
@Scope("prototype")
public class Cbact04cContext extends JicsRuntimeContext {

    @Autowired
    private TcatbalFile tcatbalFile;

    @Autowired
    private XrefFile xrefFile;

    @Autowired
    private DiscgrpFile discgrpFile;

    @Autowired
    private AccountFile accountFile;

    @Autowired
    private TransactFile transactFile;

    private IndexedFile tcatbalFileFile;

    private IndexedFile xrefFileFile;

    private IndexedFile discgrpFileFile;

    private IndexedFile accountFileFile;

    private SequentialFile transactFileFile;

    private TranCatBalRecord tranCatBalRecord;
    private TcatbalfStatus tcatbalfStatus;
    private CardXrefRecord cardXrefRecord;
```

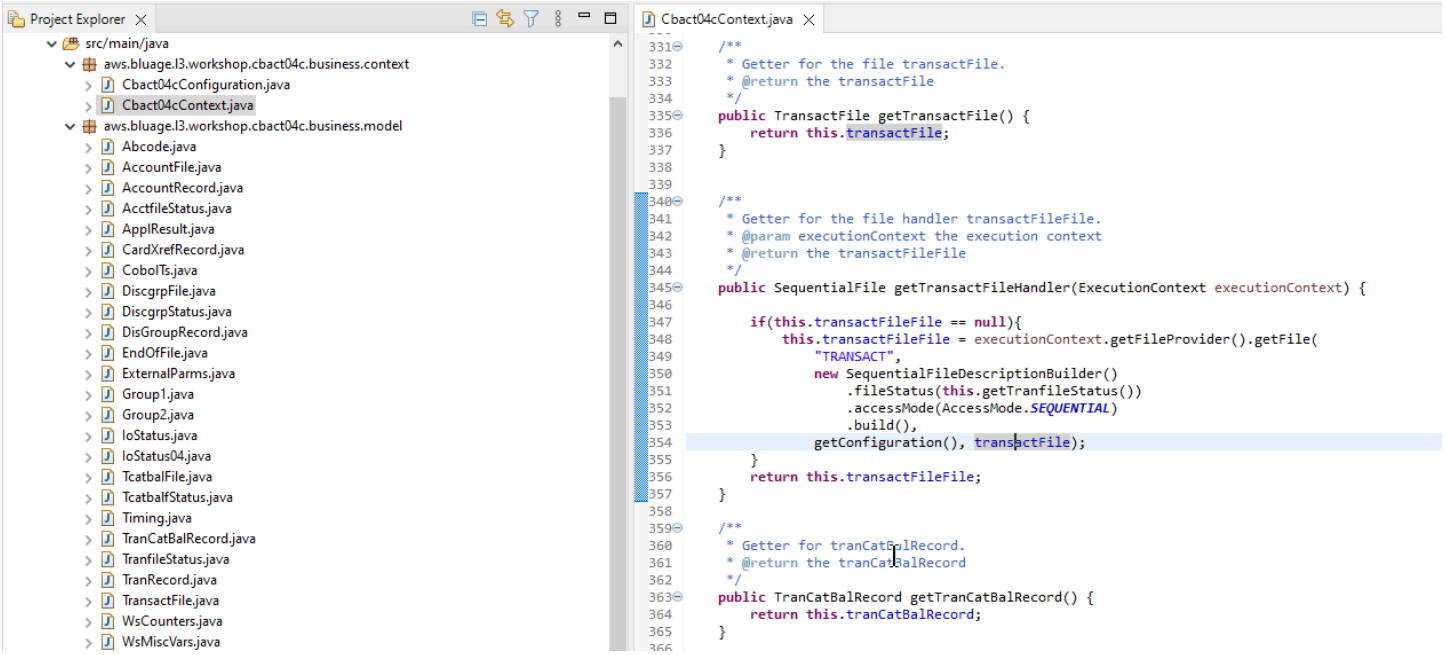
- *base package*.*program*.business.model

  模型子包包含给定程序可以使用的所有数据结构。例如，任何 01 级 COBOL 数据结构都对应于模型子包中的一个类（较低级别的数据结构是其所属 01 级结构的属性）。有关如何对 01 数据结构进行现代化的更多信息，请参阅数据简化器。

DiscgrpFile.java

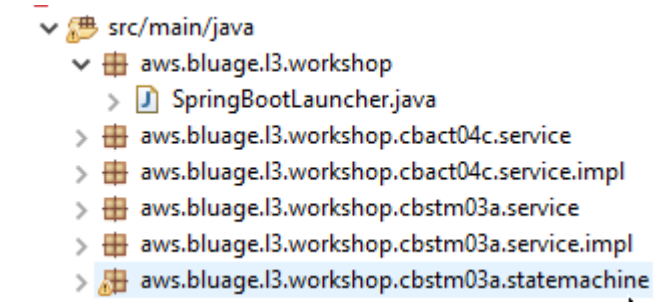
```java
package aws.bluage.l3.workshop.cbact04c.business.model;

import com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration;
import com.netfective.bluage.gapwalk.datasimplifier.data.structure.Elementary;
import com.netfective.bluage.gapwalk.datasimplifier.data.structure.Group;
import com.netfective.bluage.gapwalk.datasimplifier.entity.ElementaryRangeReference;
import com.netfective.bluage.gapwalk.datasimplifier.entity.RangeReference;
import com.netfective.bluage.gapwalk.datasimplifier.entity.RecordEntity;
import com.netfective.bluage.gapwalk.datasimplifier.metadata.type.AlphanumericType;
import com.netfective.bluage.gapwalk.datasimplifier.metadata.type.ZonedType;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Lazy;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

/**
 * Data simplifier file DiscgrpFile.
 *
 * <p>About 'fdDiscgrpRec' field, <br>uml entity: aws.bluage.l3.workshop.cbact04c.business.model.FdDiscgrpRec
 * <br></p>
 *
 */
@Component("aws.bluage.l3.workshop.cbact04c.business.model.DiscgrpFile")
@Lazy
@Scope("prototype")
public class DiscgrpFile extends RecordEntity {

    private final Group root = new Group(getData());
    private final Group fdDiscgrpRec = new Group(root);
    private final Group fdDiscgrpKey = new Group(fdDiscgrpRec);
    private final Elementary fdDisAcctGroupId = new Elementary(fdDiscgrpKey,new AlphanumericType(10));
    private final Elementary fdDisTranTypeCd = new Elementary(fdDiscgrpKey,new AlphanumericType(2));
    private final Elementary fdDisTranCatCd = new Elementary(fdDiscgrpKey,new ZonedType(4, 0, false));
    private final Elementary fdDiscgrpData = new Elementary(fdDiscgrpRec,new AlphanumericType(34));
```

所有类都扩展了 `RecordEntity` 类，该类表示对业务记录表示的访问权限。有些记录有特殊用途，因为它们绑定到 `File`。a `Record` 和 a 之间的绑定`File`是在创建文件对象时在上下文类中找到的相应的 * FileHandler 方法中进行的。例如，以下清单显示了如何绑定到 TransactFileRecord（来自模型子包）。 TransactfileFile `File`



服务项目内容

每个服务项目都附带一个专用的 Springboot 应用程序，该应用程序用作架构的主干。具体是通过名为 `SpringBootLauncher` 的类实现的，该类位于服务 java sources 的基础包中：



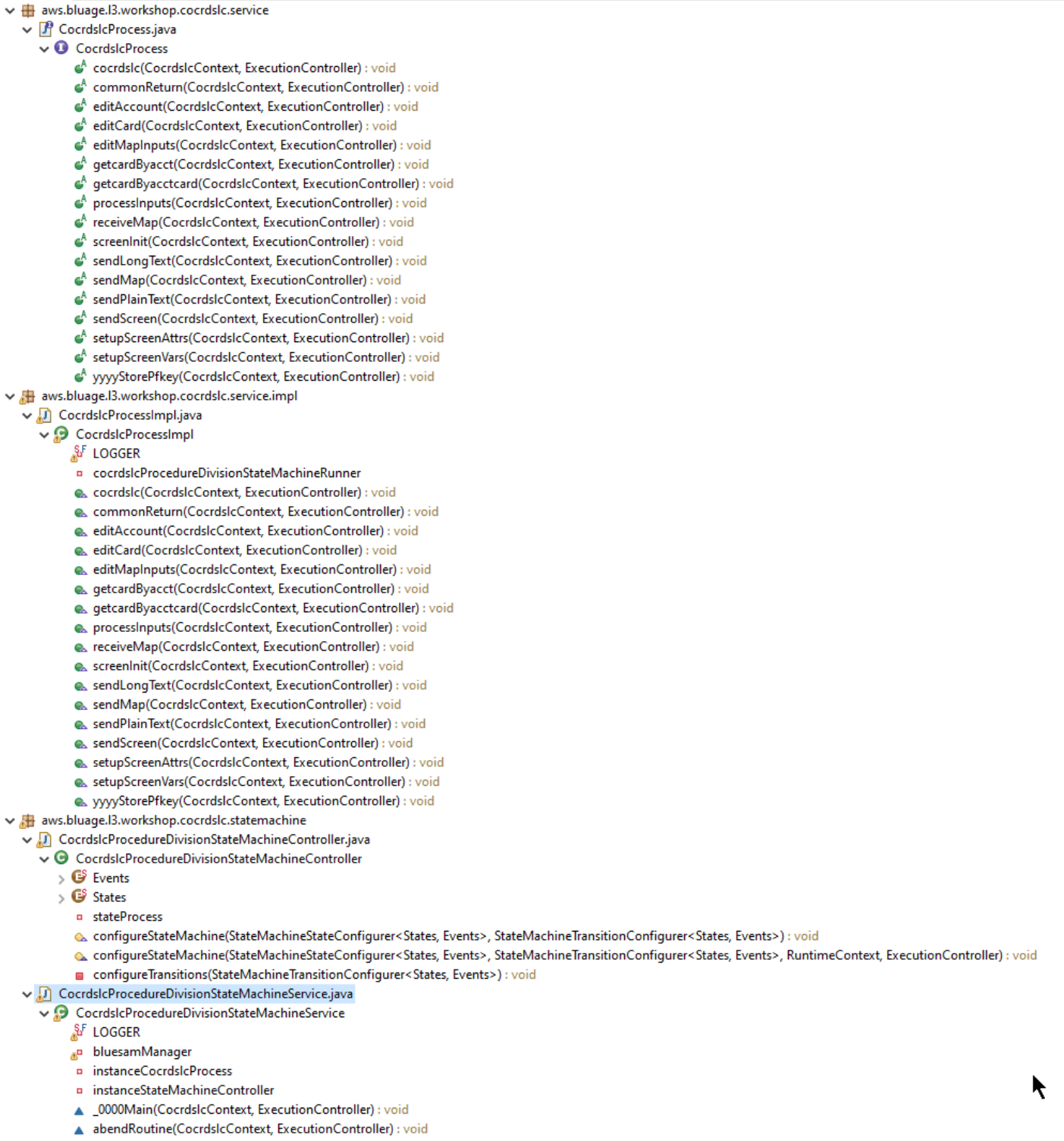
该类主要用于：

- 为程序类和托管资源（数据源/事务管理器/数据集映射/等...）之间建立联系。
- 为程序提供 `ConfigurableApplicationContext`。
- 发现所有标记为 spring 组件 (`@Component`) 的类。
- 确保程序在 `ProgramRegistry` 中正确注册。请参阅用于进行此注册的 initialize 方法。

```
/**
 * Initialization method called when the spring application is ready.
 * Register all programs and services to the gapwalk shared context.
 * @param event the application ready event
 */
@EventListener
public void initialize(ApplicationReadyEvent event) {
    Map<String, ProgramContainer> programContainers = event.getApplicationContext().getBeansOfType(ProgramContainer.class);
    programContainers.values().forEach(ProgramRegistry::registerProgram);
    Map<String, ServiceContainer> serviceContainers = event.getApplicationContext().getBeansOfType(ServiceContainer.class);
    serviceContainers.values().forEach(ServiceRegistry::registerService);
}
```

程序相关构件

在未进行事先重构的情况下，业务逻辑现代化输出将按每个传统程序两个或三个包进行组织：



最详尽的情况下将提供三个包：

- *base package*.*program*.service：包含一个名为 ProgramProcess 的接口，该接口具有处理业务逻辑的业务方法，保留了传统的执行控制流。

- *base package*.*program*.service.impl: 包含一个名为 Prog ra ProcessImpl m 的类，它是前面描述的 Process 接口的实现。这是依靠 AWS Blu Age 框架将遗留语句 “翻译” 为 java 语句的地方：

CocrdslcProcessImpl.java

```
210     /**
211      * Process operation sendScreen.
212      *
213      * @param ctx
214      * @param ctrl
215      */
216     @Override
217     public void sendScreen(final CocrdslcContext ctx, final ExecutionController ctrl) {
218         ctx.getCcWorkAreas().setCcardNextMapset(ctx.getWsLiterals().getLitThismapset());
219         ctx.getCcWorkAreas().setCcardNextMap(ctx.getWsLiterals().getLitThismap());
220         ctx.getCarddemoCommarea().setCdemoPgmReenter(true);
221         SendMapBuilder.newInstance(ctx.getDfheiblk(), ctx)
222         .withMap(ctx.getCcWorkAreas().getCcardNextMap())
223         .withMapset(ctx.getCcWorkAreas().getCcardNextMapset())
224         .withData(ctx.getGroup1().getCcrdslaoReference())
225         .withCursor()
226         .withErase()
227         .withFreeKB()
228         .execute();
229         ctx.getWsMiscStorage().setWsRespCd(ctx.getDfheiblk().getEibresp());
230     }
231
232     /**
233      * Process operation processInputs.
234      *
235      * @param ctx
236      * @param ctrl
237      */
238     @Override
239     public void processInputs(final CocrdslcContext ctx, final ExecutionController ctrl) {
240         receiveMap(ctx, ctrl);
241         editMapInputs(ctx, ctrl);
242         ctx.getCcWorkAreas().setCcardErrorMsg(ctx.getWsMiscStorage().getWsReturnMsg());
243         ctx.getCcWorkAreas().setCcardNextProg(ctx.getWsLiterals().getLitThispgm());
244         ctx.getCcWorkAreas().setCcardNextMapset(ctx.getWsLiterals().getLitThismapset());
245         ctx.getCcWorkAreas().setCcardNextMap(ctx.getWsLiterals().getLitThismap());
246     }
247
```

- *base package*.*program*.statemachine：该包有时可能不存在。当传统控制流的现代化必须使用状态机方法（即使用 Spring StateMachine 框架）来正确覆盖遗留的执行流程时，这是必需的。

  在这种情况下，状态机子包包含两个类：

  - *Program*ProcedureDivisionStateMachineController：该类扩展实现 StateMachineController（定义控制状态机执行所需的操作）和 StateMachineRunner（定义运行状态机所需的操作）接口的类，用于驱动 Spring 状态机机制；例如，示例用例中的 SimpleStateMachineController。

CocrdslcProcedureDivisionStateMachineController.java

```java
package aws.bluage.l3.workshop.cocrdslc.statemachine;

import aws.bluage.l3.workshop.cocrdslc.business.context.CocrdslcContext;

/**
 * Controller managing the state machine "CocrdslcProcedureDivisionStateMachine" execution.
 */
@Component("aws.bluage.l3.workshop.cocrdslc.statemachine.CocrdslcProcedureDivisionStateMachineController")
@Import({
aws.bluage.l3.workshop.cocrdslc.statemachine.CocrdslcProcedureDivisionStateMachineService.class
})
@Lazy
public class CocrdslcProcedureDivisionStateMachineController extends SimpleStateMachineController<States, Events> {

    /**
     * State machine states.
     */
    public enum States {
        _0000_MAIN_1, _0000_MAIN, ABEND_ROUTINE, FINAL, LOCAL_FINAL
    }

    /**
     * State machine events.
     */
    public enum Events {
        TO__0000_MAIN_1, TO__0000_MAIN, TO_ABEND_ROUTINE, TO_FINAL, TO_LOCAL_FINAL
    }

    /**
     * State machine state process service provider.
     */
    @Autowired
    @Lazy
    private CocrdslcProcedureDivisionStateMachineService stateProcess;

    @Override
    protected void configureStateMachine(StateMachineStateConfigurer<States, Events> states, StateMachineTransitionConfigurer<States, Events> transitions) throws Exception {
        throw new UnsupportedOperationException("Please use the four arguments configureStateMachine method instead: configureStateMachine(StateMachineStateConfigurer<States, Events> states, "
                + "StateMachineTransitionConfigurer<States, Events> transitions, RuntimeContext ctx, ExecutionController ctrl)");
    }

    @Override
    protected void configureStateMachine(StateMachineStateConfigurer<States, Events> states, StateMachineTransitionConfigurer<States, Events> transitions, RuntimeContext ctx, ExecutionController ctrl) throws Exception {
        StateConfigurer<States, Events> configurer = states.withStates();
        configurer.initial(States._0000_MAIN_1).end(States.FINAL);
        configurer.state(States._0000_MAIN_1);
        configurer.state(States.FINAL);

        StateConfigurer<States, Events> subConfigurer = states.withStates().parent(States._0000_MAIN_1);
        subConfigurer.initial(States._0000_MAIN).end(States.LOCAL_FINAL);
        CocrdslcContext lctx = (CocrdslcContext) ctx;
        subConfigurer.state(States._0000_MAIN, buildAction(() -> {stateProcess._0000Main(lctx, ctrl);}), null);
        subConfigurer.state(States.ABEND_ROUTINE, buildAction(() -> {stateProcess.abendRoutine(lctx, ctrl);}), null);

        configureTransitions(transitions);
    }

    /**
     * Declare state machine transitions.
     * @param transitions the transitions configuration helper
     */
    private void configureTransitions(StateMachineTransitionConfigurer<States, Events> transitions) throws Exception {
        transitions.withLocal().source(States._0000_MAIN_1).target(States.ABEND_ROUTINE).event(Events.TO_ABEND_ROUTINE);
        transitions.withExternal().source(States.ABEND_ROUTINE).target(States.FINAL).event(Events.TO_FINAL);
    }
}

```

状态机控制器定义了可能的不同状态以及状态之间的转换，可重现给定程序的传统执行控制流。

在构建状态机时，控制器会引用状态机包的关联服务类中定义的方法，如下所述：

```
subConfigurer.state(States._0000_MAIN, buildAction(() ->
 {stateProcess._0000Main(lctx, ctrl);}), null);
subConfigurer.state(States.ABEND_ROUTINE, buildAction(() ->
 {stateProcess.abendRoutine(lctx, ctrl);}), null);
```

- *Program*ProcedureDivisionStateMachineService：该服务类代表一些业务逻辑，这些逻辑需要与状态机控制器创建的状态机绑定，如前所述。

  该类方法中的代码使用状态机控制器中定义的事件：

CocrdslcProcedureDivisionStateMachineService.java

```
59      /**
60       * State process operation _0000Main.
61       *
62       * @param ctx
63       * @param ctrl
64       */
65      void _0000Main(CocrdslcContext ctx, ExecutionController ctrl) {
66          ctx.getDfheiblk().bind(ArgUtils.get(ctx, 0));
67          ctx.getDfhcommarea().bind(ArgUtils.get(ctx, 1));
68
69          /*
70          ****************************************************************
71          Program:     COCRDSLC.CBL                                     *
72          Layer:       Business logic                                   *
73          Function:    Accept and process credit card detail request    *
74          ****************************************************************
75          Copyright Amazon.com, Inc. or its affiliates.
76          All Rights Reserved.
77          Licensed under the Apache License, Version 2.0 (the \"License\").
78          You may not use this file except in compliance with the License.
79          You may obtain a copy of the License at
80          http://www.apache.org/licenses/LICENSE-2.0
81          Unless required by applicable law or agreed to in writing,
82          software distributed under the License is distributed on an
83          \"AS IS\" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
84          either express or implied. See the License for the specific
85          language governing permissions and limitations under the License
86          ****************************************************************
87          Ver: CardDemo_v1.0-15-g27d6c6f-68 Date: 2022-07-19 23:16:00 CDT */
88          instanceStateMachineController.registerSignalHandler(Events.TO_ABEND_ROUTINE, "!ABEND");
89          HandleAbendBuilder.newInstance(ctx.getDfheiblk(), ctx).execute().handleException();
90          ctx.getCcWorkAreas().getCcWorkAreaReference().getField().initialize();
91          ctx.getWsMiscStorage().getField().initialize();
92          DataUtils.initialize(ctx.getWsCommarea().getWsCommareaReference());
93
```

CocrdslcProcedureDivisionStateMachineService.java

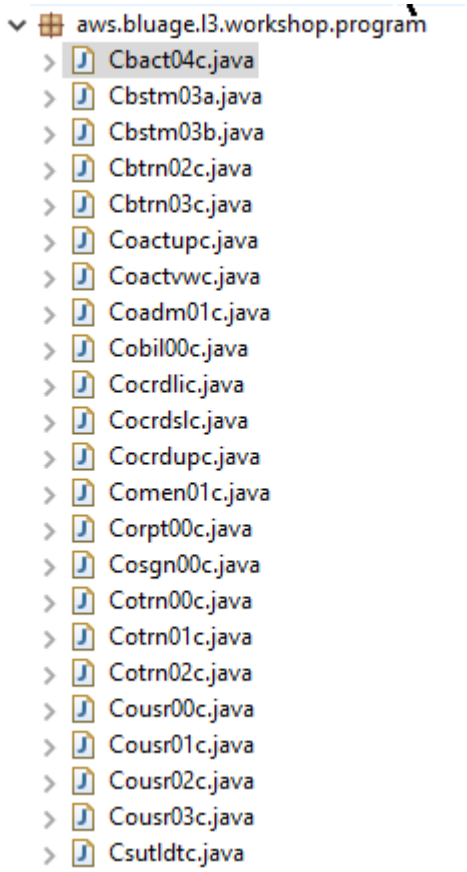
```
221     *
222     * @param ctx
223     * @param ctrl
224     */
225    void abendRoutine(CocrdslcContext ctx, ExecutionController ctrl) {
226        if (DataUtils.isLowValue(ctx.getAbendData().getAbendMsgReference())) {
227            ctx.getAbendData().setAbendMsg("UNEXPECTED ABEND OCCURRED.");
228        }
229        ctx.getAbendData().setAbendCulprit(ctx.getWsLiterals().getLitThispgm());
230        SendTextBuilder.newInstance(ctx.getDfheiblk(), ctx)
231        .withData(ctx.getAbendData())
232        .withLength(134)
233        .execute();
234        HandleAbendBuilder.newInstance(ctx.getDfheiblk(), ctx).cancel().execute().handleException();
235        AbendBuilder.newInstance(ctx.getDfheiblk(), ctx).withAbendCode("9999").execute().handleException();
236
237        /*
238        Ver: CardDemo_v1.0-15-g27d6c6f-68 Date: 2022-07-19 23:12:33 CDT */
239        instanceStateMachineController.sendEvent(Events.TO_FINAL);
240
241    }
242 }
243
```

状态机服务还会调用前面所述的过程服务实现：

CocrdslcProcedureDivisionStateMachineService.java

```
166
167             /*
168             ************************************************************
169             COMING FROM CREDIT CARD LIST SCREEN
170             SELECTION CRITERIA ALREADY VALIDATED
171             ************************************************************ */
172         } else if (ctx.getCarddemoCommarea().isCdemoPgmEnter() && DataUtils.compare(ctx.getCarddemoCommarea().getCdemoFromProgramReference(), ctx.getWsLiterals().getLitCclistpgmReference()) == 0) {
173             ctx.getWsMiscStorage().setInputOk(true);
174             ctx.getCcWorkAreas().setCcAcctIdN(ctx.getCarddemoCommarea().getCdemoAcctId());
175             ctx.getCcWorkAreas().setCcCardNumN(ctx.getCarddemoCommarea().getCdemoCardNum());
176             instanceCocrdslcProcess.getcardByacctcard(ctx, ctrl);
177             instanceCocrdslcProcess.sendMap(ctx, ctrl);
178             instanceCocrdslcProcess.commonReturn(ctx, ctrl);
179         } else if (ctx.getCarddemoCommarea().isCdemoPgmEnter()) {
180
181             /*
182             ************************************************************
183             COMING FROM SOME OTHER CONTEXT
184             SELECTION CRITERIA TO BE GATHERED
185             ************************************************************ */
186             instanceCocrdslcProcess.sendMap(ctx, ctrl);
187             instanceCocrdslcProcess.commonReturn(ctx, ctrl);
188         } else if (ctx.getCarddemoCommarea().isCdemoPgmReenter()) {
189             instanceCocrdslcProcess.processInputs(ctx, ctrl);
190             if (ctx.getWsMiscStorage().isInputError()) {
191                 instanceCocrdslcProcess.sendMap(ctx, ctrl);
192                 instanceCocrdslcProcess.commonReturn(ctx, ctrl);
193             } else {
194                 instanceCocrdslcProcess.getcardByacctcard(ctx, ctrl);
195                 instanceCocrdslcProcess.sendMap(ctx, ctrl);
196                 instanceCocrdslcProcess.commonReturn(ctx, ctrl);
197             }
198         } else {
199             ctx.getAbendData().setAbendCulprit(ctx.getWsLiterals().getLitThispgm());
200             ctx.getAbendData().setAbendCode("0001");
201             DataUtils.setToBlank(ctx.getAbendData().getAbendReasonReference());
202             ctx.getWsMiscStorage().setWsReturnMsg("UNEXPECTED DATA SCENARIO");
203             instanceCocrdslcProcess.sendPlainText(ctx, ctrl);
204         }
205
206         /*
207         If we had an error setup error message that slipped through
208         Display and return */
209         if (ctx.getWsMiscStorage().isInputError()) {
210             ctx.getCcWorkAreas().setCcardErrorMsg(ctx.getWsMiscStorage().getWsReturnMsg());
211             instanceCocrdslcProcess.sendMap(ctx, ctrl);
212             instanceCocrdslcProcess.commonReturn(ctx, ctrl);
213         }
214         instanceCocrdslcProcess.commonReturn(ctx, ctrl);
```

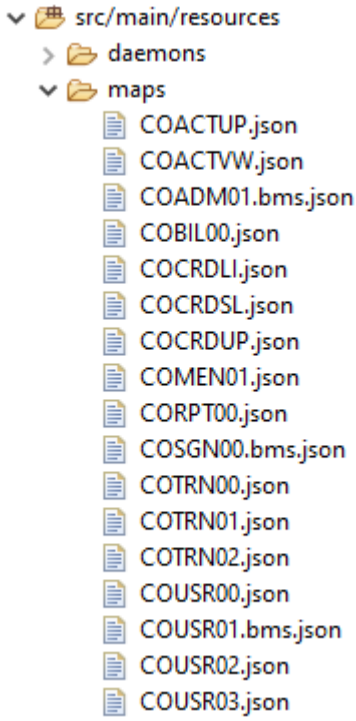
除此之外，名为 *base package*.program 的包也发挥重要作用，因为该包为每个程序收集一个类，该类将作为程序的入口点（稍后会详细介绍）。每个类都实现 Program 接口，即程序入口点的标记。



其他构件

- BMS 映射配套构件

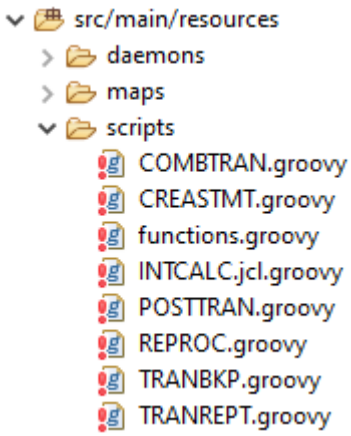
  除了与程序相关的构件外，服务项目还可以包含用于各种用途的其他构件。在对 CICS 在线应用程序进行现代化时，现代化过程会生成一个 json 文件并放入 /src/main/resources 文件夹的映射文件夹中：



Blu Age 运行时使用这些 json 文件将 SEND MAP 语句使用的记录与屏幕字段进行绑定。

- Groovy 脚本

  如果传统应用程序有 JCL 脚本，则这些脚本已现代化为 groovy 脚本，存储在 /src/main/resources/scripts 文件夹中（稍后会详细介绍该特定位置）：

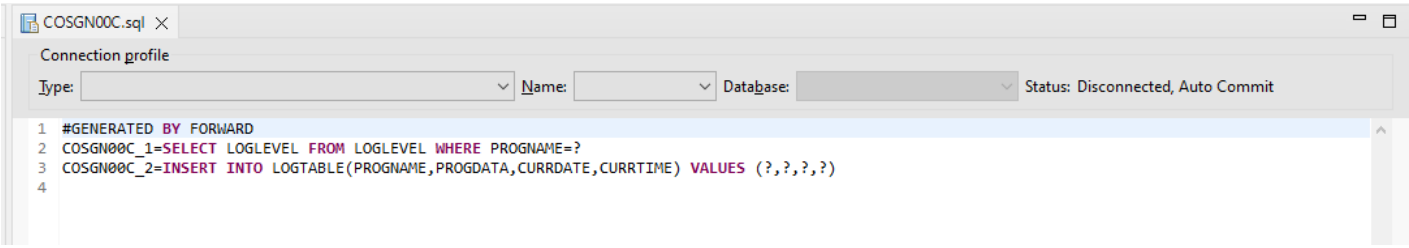


  这些脚本用于启动批处理作业（专用、非交互式、CPU 密集型数据处理工作负载）。

- SQL 文件

  如果传统应用程序使用 SQL 查询，则相应的现代化 SQL 查询已收集在专用属性文件中，使用命名模式 program.sql，其中 program 是使用这些查询的程序的名称。

这些 sql 文件的内容是 (key=query) 条目的集合，其中每个查询都与一个唯一的键相关联，现代化程序使用该键来运行给定查询：

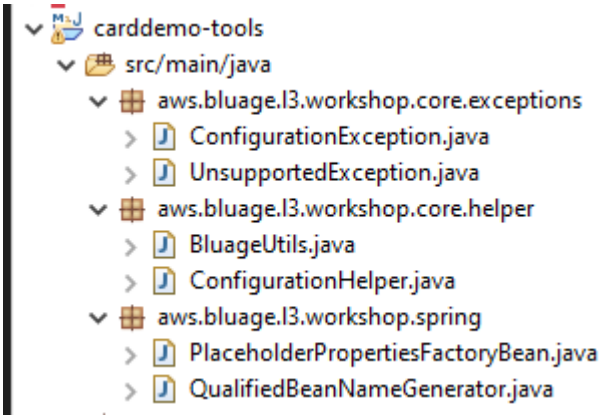


例如，COSGN00C 程序正在使用键 “COSGN00C_1”（sql 文件中的第一个条目）执行查询：

```
    /**
     * Process operation getProgramLogLevel.
     *
     * ****************************************************************
     *                       GET PROGRAM LOG LEVEL
     * ****************************************************************
     *
     * @param ctx
     * @param ctrl
     */
    @Override
    public void getProgramLogLevel(final Cosgn00cContext ctx, final ExecutionController ctrl) {
        SQLExecutorBuilder.newInstance(ctrl, ctx, ctx.getSqlca())
            .mapInParameter(SQLParameterBuilder.newInstance(ctx.getLogData().getLogProgramName()).type(JDBCType.CHAR))
            .mapOutParameter(SQLParameterBuilder.newInstance(ctx.getLogData().getLogProgramLevelReference()))
            .execute("COSGN00C_1");
        if (ctx.getSqlca().getSqlcode() == 100) {
            ctx.getLogData().setLogProgramLevel("N");
        }
    }
```

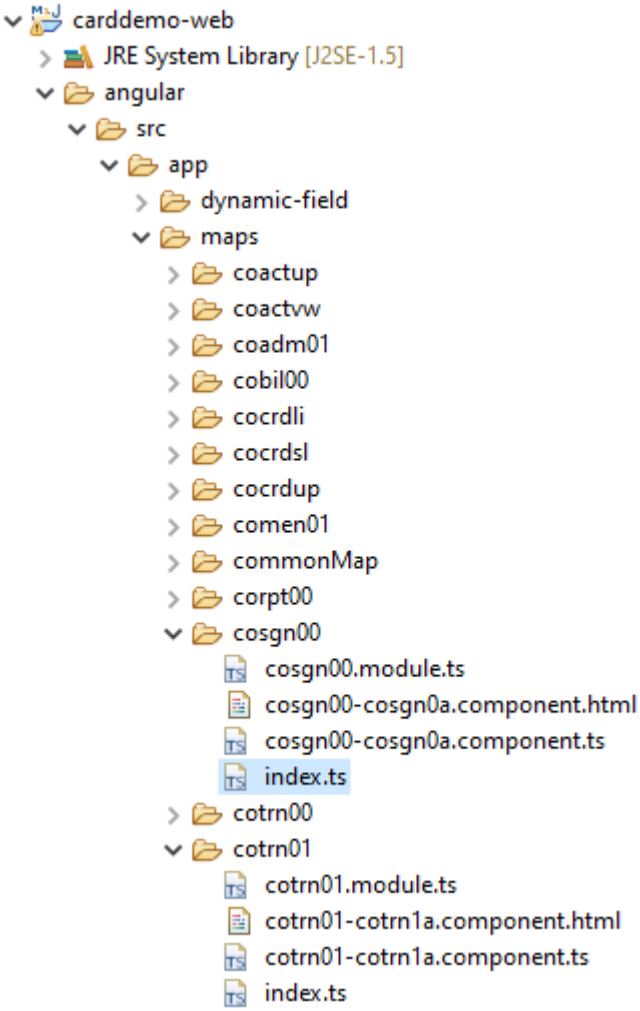
实体项目内容

名字以“-tools”结尾的实用程序项目包含一组技术实用工具，这些工具可用于所有其他项目。



Web 项目内容

Web 项目仅在对传统 UI 元素进行现代化时才存在。用于构建现代化应用程序前端的现代 UI 元素基于 Angular 用于显示现代化构件的示例应用程序为在大型机上运行的 COBOL/CICS 应用程序。CICS 系统使用映射 来表示 UI 屏幕。每个映射相应的现代元素为一个 html 文件和多个 Typescript 文件：



Web 项目仅处理应用程序的前端方面。依赖于实用程序和实体项目的服务项目提供后端服务。前端和后端之间的链接是通过名为 Gapwalk-Application 的 Web 应用程序建立的，该应用程序是标准 AWS Blu Age 运行时发行版的一部分。

运行和调用程序

在遗留系统上，程序被编译为独立的可执行文件，可以通过 CALL 机制（例如 COBOL CALL 语句）进行自我调用，并根据需要传递参数。现代化应用程序提供了相同的功能，但使用了不同的方法，因为所涉及的构件的性质与传统构件的性质不同。

在现代化系统上，程序入口点是实现 `Program` 接口的特定类，是 Spring 组件 (@Component)，位于服务项目中名为 *`base package`*`.program` 的包中。

程序注册

每次启动托管现代化应用程序的 Tomcat 服务器时，也会启动服务 Springboot 应用程序，从而触发程序注册。名为 `ProgramRegistry` 的专用注册表中填充了程序条目，每个程序都使用其标识符进行注册，每个已知的程序标识符对应一个条目，这意味着如果一个程序具有多个不同的已知标识符，则注册表包含的条目与标识符的数量相等。

给定程序的注册依赖于 getProgramIdentifiers () 方法返回的标识符集合：

Cbact04c.java

```java
package aws.bluage.l3.workshop.program;

import aws.bluage.l3.workshop.SpringBootLauncher;

/**
 * Reference the spring application of program CBACT04C.
 *  Provides an access to the contained program for the run unit.
 */
@Component
@Import({
    aws.bluage.l3.workshop.cbact04c.business.context.Cbact04cConfiguration.class,
    aws.bluage.l3.workshop.cbact04c.business.context.Cbact04cContext.class,
    aws.bluage.l3.workshop.cbact04c.service.impl.Cbact04cProcessImpl.class
})
public class Cbact04c implements Program {
    /**
     * Unique identifiers for the contained program.
     */
    private static final Set<String> programIdentifiers = Collections.unmodifiableSet(Stream.of("CBACT04C").collect(Collectors.toSet()));

    /**
     * Main program identifier for the contained program.
     */
    private static final String programIdentifier = "CBACT04C";
    @Autowired
    PlatformTransactionManager transactionManager;

    @Autowired
    Map<String, DataSource> datasources;
    @Autowired
    BeanFactory beanFactory;
    /**
     * {@inheritDoc}
     */
    @Override
    public ConfigurableApplicationContext getSpringApplication() {
        return SpringBootLauncher.getCac();
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public void updateExecutionContext(ExecutionContext executionContext) {
        executionContext.setDatasources(datasources);
        executionContext.setDatabaseSupport(ExecutionContext.DatabaseSupport.POSTGRE);
        executionContext.setSqlcaVersion(ExecutionContext.SqlcaVersion.getEnum("ansi-comp5"));
        executionContext.setTransactionManager(transactionManager);
        executionContext.setUseSQLDateNewParadigm(true);
        executionContext.setUseSQLTrimStringType(false);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public Set<String> getProgramIdentifiers() {
        return programIdentifiers;
    }
```

在此示例中，该程序注册了一次，名为“CBACT04C”（查看 programIdentifiers 集合的内容）。Tomcat 日志显示每个程序的注册情况。程序注册仅取决于声明的程序标识符，而不取决于程序类名词本身（尽管通常程序标识符和程序类名称是一致的）。

同样的注册机制适用于各种实用程序 AWS Blu Age Web 应用程序带来的实用程序，这些应用程序是 AWS Blu Age 运行时发行版的一部分。例如，Gapwalk-Utility-Pgm Web 应用程序提供了 z/OS 系统实用程序（IDCAMS、ICEGENER、SORT 等）的等效功能，可以由现代化程序或脚本调用。在 Tomcat 启动时注册的所有可用实用程序都记录在 Tomcat 日志中。

脚本和进程守护程序注册

在 Tomcat 启动时，位于 /src/main/resources/scripts 文件夹层次结构中的 groovy 脚本也会发生类似的注册过程。遍历脚本文件夹层次结构后，所有发现的 groovy 脚本（特殊的 functions.groovy 保留脚本除外）都将注册到 `ScriptRegistry` 中，并使用其短名称（脚本文件名中位于第一个点字符之前的部分）作为密钥进行检索。

> **Note**
>
> - 如果多个脚本的文件名会导致生成相同的注册密钥，则仅注册最新的脚本，并覆盖该给定密钥之前的任何注册。
> - 考虑到上述注意事项，使用子文件夹时需引起注意，因为注册机制会使层次结构扁平化并可能导致意外覆盖。层次结构不计入注册过程：通常 /scripts/A/myscript.groovy 和 /scripts/B/myscript.groovy 会导致 /scripts/B/myscript.groovy 覆盖 /scripts/A/myscript.groovy。

/src/main/resources/daemons 文件夹中的 groovy 脚本的处理方式略有不同。它们仍被注册为常规脚本，但此外，它们仅在 Tomcat 启动时以异步方式启动一次。

在 `ScriptRegistry` 中注册脚本后，即可使用 Gapwalk-Application 公开的专用端点进行 REST 调用来启动这些脚本。有关更多信息，请参阅相应文档。

程序调用程序

每个程序都可以将另一个程序作为子程序进行调用，并向其传递参数。程序使用 `ExecutionController` 接口实现来进行调用（大多数情况下，这是一个 `ExecutionControllerImpl` 实例），并使用名为 `CallBuilder` 的 fluent API 机制来构建程序调用参数。

所有程序的方法都将 `RuntimeContext` 和 `ExecutionController` 作为方法参数，因此 `ExecutionController` 始终可用于调用其他程序。

例如，下图显示了 CBST03A 程序如何将 CBST03B 程序作为子程序调用，并向其传递参数：

CbstmO3aProcessImpl.java

```java
    /**
     * Process operation xreffileGetNext.
     *
     * ----------------------------------------------------------------*
     *
     * @param ctx
     * @param ctrl
     */
    @Override
    public void xreffileGetNext(final Cbstm03aContext ctx, final ExecutionController ctrl) {
        ctx.getWsM03bArea().setWsM03bDd("XREFFILE");
        ctx.getWsM03bArea().setM03bRead(true);
        DataUtils.setToZeroes(ctx.getWsM03bArea().getWsM03bRcReference());
        DataUtils.setToBlank(ctx.getWsM03bArea().getWsM03bFldtReference());
        ctrl.callSubProgram("CBSTM03B", CallBuilder.newInstance()
            .byReference(ctx.getWsM03bArea())
            .getArguments(), ctx);
        if (DataUtils.compare(ctx.getWsM03bArea().getWsM03bRcReference(), "00") == 0) {

            /*
            Do nothing */
        } else if (DataUtils.compare(ctx.getWsM03bArea().getWsM03bRcReference(), "10") == 0) {
            ctx.getMiscVariables().setEndOfFile("Y");
        } else {
            if (LOGGER.isInfoEnabled()) LOGGER.info("ERROR READING XREFFILE");
            if (LOGGER.isInfoEnabled()) LOGGER.info("{}{}" , "RETURN CODE: " , ctx.getWsM03bArea().getWsM03bRc());
            abendProgram(ctx, ctrl);
        }
        ctx.getCardXrefRecord().setBytes(ctx.getWsM03bArea().getWsM03bFldtReference().getBytes());
    }
```

- `ExecutionController.callSubProgram` 的第一个参数是要调用的程序的标识符（即用于程序注册的标识符之一，请参阅以上段落）。
- 第二个参数是在 `CallBuilder` 上构建的结果，是一个 `Record` 数组，对应于从调用方传递给被调用方的数据。
- 第三个也是最后一个参数是调用者 `RuntimeContext` 实例。

所有三个参数都是必需的，不能为空，但第二个参数可以是空数组。

被调用方仅在最初设计为能够处理传递的参数时才能处理这些参数。对于传统的 COBOL 程序，这意味着要有一个 LINKAGE 部分和一个 USING 子句，以便过程部分使用 LINKAGE 元素。

例如，请参阅相应的 CBSTM03B.CBL COBOL 源文件：

github.com/aws-samples/aws-mainframe-modernization-carddemo/blob/main/app/cbl/CBSTM03B.CBL

```
 98
 99          LINKAGE SECTION.
100          01  LK-M03B-AREA.
101              05  LK-M03B-DD          PIC X(08).
102              05  LK-M03B-OPER        PIC X(01).
103                88  M03B-OPEN       VALUE 'O'.
104                88  M03B-CLOSE      VALUE 'C'.
105                88  M03B-READ       VALUE 'R'.
106                88  M03B-READ-K     VALUE 'K'.
107                88  M03B-WRITE      VALUE 'W'.
108                88  M03B-REWRITE    VALUE 'Z'.
109              05  LK-M03B-RC          PIC X(02).
110              05  LK-M03B-KEY         PIC X(25).
111              05  LK-M03B-KEY-LN      PIC S9(4).
112              05  LK-M03B-FLDT        PIC X(1000).
113
114          PROCEDURE DIVISION USING LK-M03B-AREA.
115
```

因此，CBSTM03B 程序采用单个 `Record` 作为参数（大小为 1 的数组）。这是 `CallBuilder` 使用 byReference() 和 getArguments() 方法链接构建的内容。

`CallBuilder` fluent API 类有几种方法可用于填充要传递给被调用方的参数数组：

- asPointer (RecordAdaptable)：通过引用添加指针类型的参数。指针表示目标数据结构的地址。
- byReference (RecordAdaptable)：通过引用添加参数。调用方将看到被调用方执行的修改。
- byReference (RecordAdaptable...)：先前方法的可变参数变体。
- byValue(Object)：添加一个按值转换为 `Record` 的参数。调用方将看到被调用方执行的修改。
- byValue (RecordAdaptable)：与前面的方法相同，但该参数可以直接作为`RecordAdaptable`.
- byValueWith边界（Object、int、int）：添加一个参数，将其转换为 `aRecord`，按值提取由给定边界定义的字节数组部分。

最后，getArguments 方法将收集所有添加的参数，并将其作为 `Record` 数组返回。

> **Note**
>
> 调用方需确保参数数组具有所需的大小，在内存布局与 linkage 元素的预期布局方面，项目顺序正确且兼容。

调用程序的脚本

从 groovy 脚本中调用注册程序需要使用用于实现 `MainProgramRunner` 接口的类实例。通常，获得这样的实例是通过 Spring 的 ApplicationContext 使用来实现的：

REPROC.groovy

```groovy
// Import
import com.netfective.bluage.gapwalk.rt.provider.ScriptRegistry
import com.netfective.bluage.gapwalk.rt.call.MainProgramRunner
import com.netfective.bluage.gapwalk.io.support.FileConfigurationUtils
import com.netfective.bluage.gapwalk.rt.job.support.DefaultJobContext
import com.netfective.bluage.gapwalk.rt.utils.GroovyUtils
import com.netfective.bluage.gapwalk.rt.io.support.FileConfiguration
import com.netfective.bluage.gapwalk.rt.shared.AbendException
import com.netfective.bluage.gapwalk.rt.call.exception.GroovyExecutionException
// Variables
mpr = applicationContext.getBean("com.netfective.bluage.gapwalk.rt.call.ExecutionController", MainProgramRunner.class)
TreeMap mapTransfo = [:]
```

`MainProgramRunner` 接口可用后，使用 runProgram 方法调用程序并将目标程序的标识符作为参数传递：

REPROC.groovy

```
50 //*******************************************************************
51 //*                              STEPS                              *
52 //*******************************************************************
53 // STEP PRC001 - PGM - IDCAMS*****************************************************
54 def stepPRC001(Object shell, Map params, Map programResults){
55     shell.with {
56         if (checkValidProgramResults(programResults)) {
57             return execStep("PRC001", "IDCAMS", programResults, {
58                 mpr
59                     .withFileConfigurations(new FileConfigurationUtils()
60                         .systemOut("SYSPRINT")
61                         .output("*")
62                         .build()
63                         .bluesam("FILEIN")
64                         .dataset("NULLFILE")
65                         .disposition("SHR")
66                         .build()
67                         .bluesam("FILEOUT")
68                         .dataset("NULLFILE")
69                         .disposition("SHR")
70                         .build()
71                         .fileSystem("SYSIN")
72                         .path("&CNTLLIB(REPROCT)")
73                         .disposition("SHR")
74                         .build()
75                         .getFileConfigurations(fcmap))
76                     .withParameters(params)
77                     .runProgram("IDCAMS")
78                 })
79         }
80     }
81 }
```

在上述示例中，作业步骤调用 IDCAMS（文件处理实用程序），提供实际数据集定义与其逻辑标识符之间的映射。

在处理数据集时，传统程序大多使用逻辑名称来标识数据集。当从脚本调用程序时，脚本必须将逻辑名称与实际的物理数据集进行映射。这些数据集可能位于文件系统中、Blusam 存储中，甚至可以由内联流、多个数据集的串联或生成 GDG 来定义。

使用该 withFileConfiguration 方法构建数据集的逻辑到物理映射，并将其提供给被调用的程序。

自行编写程序

自行编写程序供脚本或其他现代化程序调用是一项常见任务。通常，在现代化项目中，当可执行的传统程序是使用现代化过程不支持的语言编写的，或者源代码丢失（是的，可能会发生这种情况），或者该程序是一个源代码不可用的实用程序时，您需要自行编写程序。

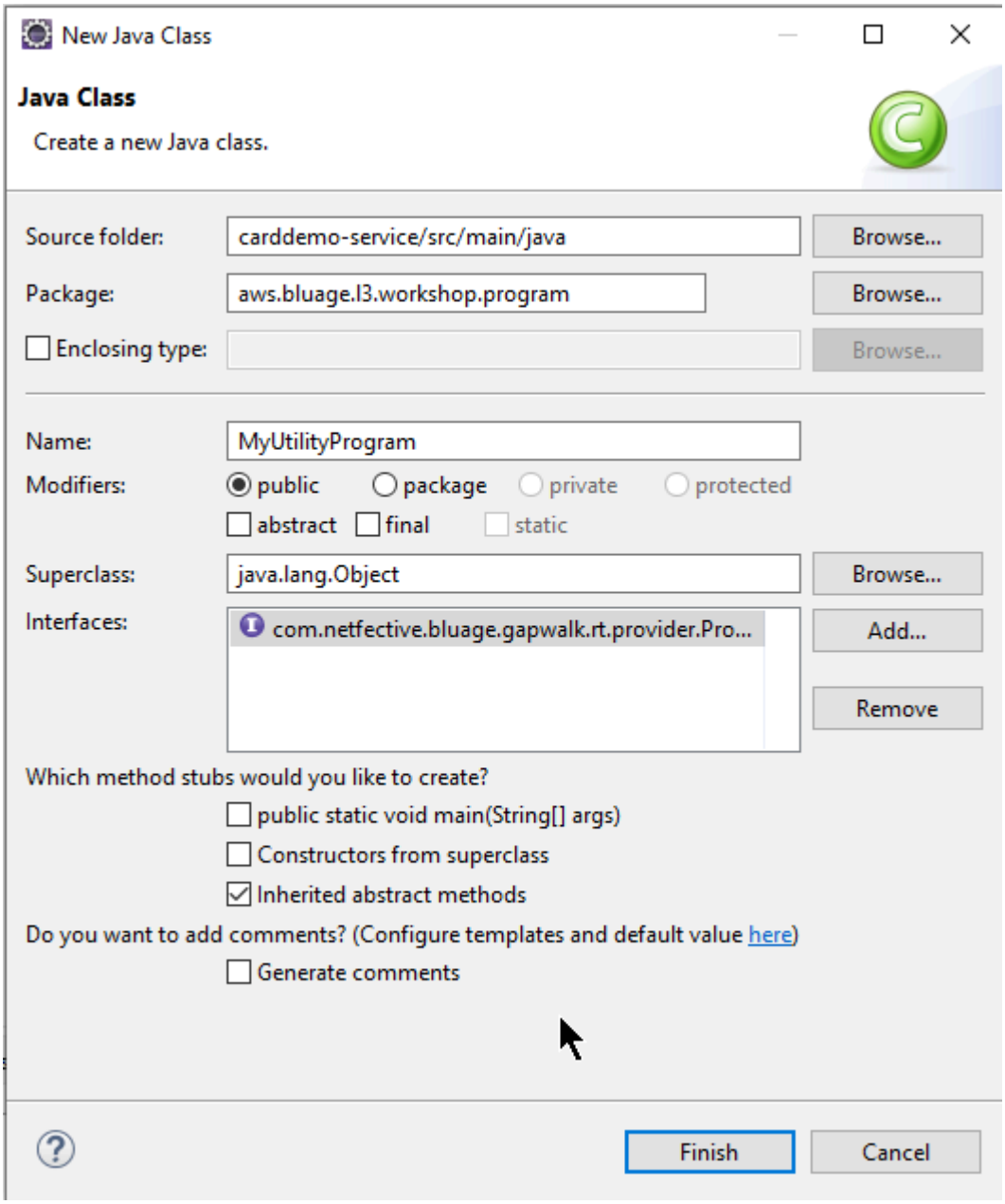
在这种情况下，您可能必须自己使用 java 编写缺失的程序（前提是您对程序的预期行为、程序参数的内存布局（如果有）等有足够的了解。）您的 java 程序必须符合本文档中描述的程序机制，以便其他程序和脚本可以运行它。

要确保该程序可用，您必须完成两个必需的步骤：

- 编写一个能正确实现 `Program` 接口的类，以便可以注册和调用该接口。
- 确保您的程序已正确注册，以便其他程序/脚本可以看到该程序。

编写程序实现

使用 IDE 创建用于实现 `Program` 接口的新 java 类：



下图显示了 Eclipse IDE，用于创建所有要实现的必需的方法：

MyUtilityProgram.java

```java
package aws.bluage.l3.workshop.program;

import java.util.Set;

public class MyUtilityProgram implements Program {

    @Override
    public ConfigurableApplicationContext getSpringApplication() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Set<String> getProgramIdentifiers() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Context getContext() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void run(ExecutionController ctrl) {
        // TODO Auto-generated method stub

    }

}
```

Spring 集成

首先，必须将该类声明为 Spring 组件。为该类添加 `@Component` 注释：

```java
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.stereotype.Component;

import com.netfective.bluage.gapwalk.rt.call.ExecutionController;
import com.netfective.bluage.gapwalk.rt.context.Context;
import com.netfective.bluage.gapwalk.rt.provider.Program;

import aws.bluage.l3.workshop.SpringBootLauncher;

@Component
public class MyUtilityProgram implements Program {
```

接下来，正确实现所需的方法。在本示例中，我们将 `MyUtilityProgram` 添加到已包含所有现代化程序的包中，该位置允许程序使用现有的 Springboot 应用程序来提供 getSpringApplication 方法实现`ConfigurableApplicationContext`所需的内容：

```
public class MyUtilityProgram implements Program {

    @Override
    public ConfigurableApplicationContext getSpringApplication() {
        return SpringBootLauncher.getCac();
    }
```

您可以为自己的程序选择不同的位置。例如，您可以将给定的程序放在另一个专门的服务项目中。确保给定的服务项目有自己的 Springboot 应用程序，这样就可以检索 ApplicationContext （应该是 aConfigurableApplicationContext ）。

为程序添加标识

要使程序能被其他程序和脚本调用，必须为其提供至少一个标识符，该标识符不得与系统中任何现有的其他已注册程序发生冲突。选择标识符可以根据要涵盖现有传统程序替换的需求来确定；在这种情况下，您必须使用所需的标识符，例如在整个传统程序中 CALL 次数中所出现的标识符。在传统系统中，大多数程序标识符的长度为 8 个字符。

在程序中创建一组不可修改的标识符是实现此目的的一种方法。以下示例显示了如何选择“MYUTILPG”作为单一标识符：

```
@Component
public class MyUtilityProgram implements Program {

    /**
     * Unique identifiers for the contained program.
     */
    private static final Set<String> programIdentifiers = Collections.unmodifiableSet(Stream.of("MYUTILPG").collect(Collectors.toSet()));

    public ConfigurableApplicationContext getSpringApplication() {

    @Override
    public Set<String> getProgramIdentifiers() {
        return programIdentifiers;
    }
```

将程序与上下文关联

程序需要一个配套 RuntimeContext 实例。对于现代化程序， AWS Blu Age 会使用传统程序中的数据结构自动生成配套上下文。

如果您正在编写自己的程序，则还必须编写配套上下文。

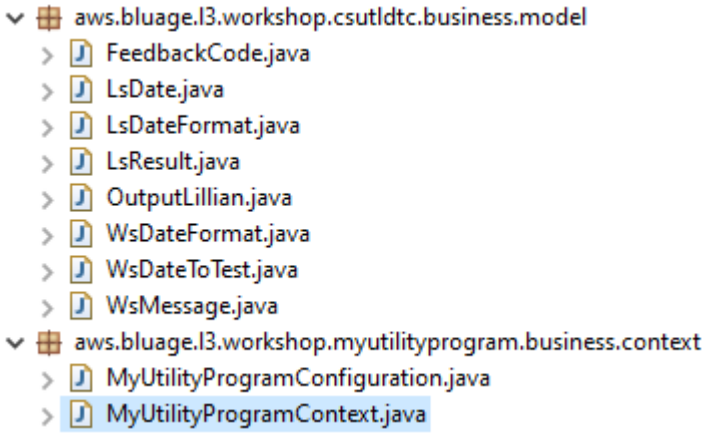
请参阅项目相关类，即可以看到一个程序至少需要两个配套类：

- 配置类。
- 使用配置的上下文类。

如果实用程序使用任何额外的数据结构，则还应编写相应的数据结构并使其被上下文使用。

这些类应位于包层次结构中的包中，应用程序启动时会扫描该包层次结构，从而确保上下文组件和配置由 Spring 框架处理。

我们在实体项目中新创建的 *base package*.myutilityprogram.business.context 包中编写一个最小配置和上下文：



以下是配置内容，其使用的配置构建与附近其他（现代化）程序类似。您可能需要根据自己的特定需求对其进行自定义。

MyUtilityProgramConfiguration.java

```java
package aws.bluage.l3.workshop.myutilityprogram.business.context;

import java.nio.charset.Charset;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Lazy;

import com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration;
import com.netfective.bluage.gapwalk.datasimplifier.configuration.ConfigurationBuilder;

/**
 * Creates Datasimplifier configuration for the MyUtilityProgram context.
 */
@org.springframework.context.annotation.Configuration
@Lazy
public class MyUtilityProgramConfiguration {

    @Bean(name = "MyUtilityProgramContextConfiguration")
    public Configuration configuration() {
        return new ConfigurationBuilder()
                .encoding(Charset.forName("CP1047"))
                .humanReadableEncoding(Charset.forName("ISO-8859-15"))
                .initDefaultByte(0)
                .build();
    }
}

```

注意：

- 一般命名约定是ProgramName配置。
- 必须使用 @org.springframework.context.annotation.Configuration 和 @Lazy 注释。
- bean 名称通常遵循ProgramNameContextConfiguration 惯例，但这不是强制性的。确保在整个项目中避免 Bean 名称冲突。
- 要实现的单个方法必须返回一个 `Configuration` 对象。使用 `ConfigurationBuilder` fluent API 来帮助您构建此方法。

相关的上下文：

MyUtilityProgramContext.java

```java

import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Lazy;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

import com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration;
import com.netfective.bluage.gapwalk.rt.context.RuntimeContext;

@Component("aws.bluage.l3.workshop.mytuilityprogram.business.context.MyUtilityProgramContext")
@Lazy
@Scope("prototype")
public class MyUtilityProgramContext extends RuntimeContext{

    protected MyUtilityProgramContext(@Qualifier("MyUtilityProgramContextConfiguration") Configuration configuration) {
        super(configuration);
    }

    @Override
    public void cleanUp() {
        // TODO implement clean-up of associated data structures if any
    }

    @Override
    protected void doReset() {
        // TODO implement reset of associated data structures if any
    }

}

```

注意事项

- 上下文类应扩展现有的 `Context` 接口实现（`RuntimeContext` 或者 `JicsRuntimeContext`（即具有 JICS 特定项的增强 `RuntimeContext`））。
- 一般命名约定是ProgramName上下文。
- 必须将其声明为 Prototype 组件，并使用 @Lazy 注解。
- 构造函数引用关联的配置，使用 @Qualifier 注释来定位正确的配置类。
- 如果实用程序使用一些额外的数据结构，这些数据结构应：

- 已编写并添加到 *base package*.business.model 包中。
- 在上下文中引用。查看其他现有的上下文类，了解如何引用数据结构类并根据需要调整上下文方法（构造函数/清理/重置）。

专用上下文可用后，新程序即可使用该上下文：

MyUtilityProgram.java

```
import aws.bluage.l3.workshop.SpringBootLauncher;

@Component
@Import({
    aws.bluage.l3.workshop.myutilityprogram.business.context.MyUtilityProgramConfiguration.class,
    aws.bluage.l3.workshop.myutilityprogram.business.context.MyUtilityProgramContext.class
})
public class MyUtilityProgram implements Program {

    @Autowired
    BeanFactory beanFactory;

    /**
     * Unique identifiers for the contained program.
     */
    private static final Set<String> programIdentifiers = Collections.unmodifiableSet(Stream.of("MYUTILPG").collect(Collectors.toSet()));

    private static final String programIdentifier = "MYUTILPG";

    @Override
    public ConfigurableApplicationContext getSpringApplication() {
        return SpringBootLauncher.getCac();
    }

    @Override
    public Set<String> getProgramIdentifiers() {
        return programIdentifiers;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public String getProgramMainIdentifier() {
        return programIdentifier;
    }


    @Override
    public Context getContext() {
        return ProgramContextStore.getOrCreate(
        getProgramMainIdentifier(),
        aws.bluage.l3.workshop.myutilityprogram.business.context.MyUtilityProgramContext.class,
        beanFactory);
    }
```

注意:

- getContext 方法必须严格如图所示，使用对ProgramContextStore类的 getOrCreate 方法和自动连接的 Spring 进行委托。BeanFactory单个程序标识符用于在 ProgramContextStore 中存储程序上下文；该标识符被称为“程序主标识符”。
- 必须使用 @Import spring 注释来引用配套配置和上下文类。

实现业务逻辑

完成程序框架后，实现新实用程序的业务逻辑。

在程序的 `run` 方法中执行此操作。该方法会在程序被其他程序或脚本调用时执行。

祝您编码顺利！

处理程序注册

最后，请确保将新程序在 `ProgramRegistry` 中正确注册。如果您将新程序添加到已包含其他程序的包中，则无需执行任何其他操作。在应用程序启动时，新程序会被选中并与其所有邻居程序一并注册。

如果您为程序选择了其他位置，则必须确保该程序在 Tomcat 启动时能够正确注册。有关如何执行此操作的一些灵感，请查看服务项目中生成 SpringbootLauncher 类的初始化方法（请参阅服务项目内容）。

查看 Tomcat 启动日志。每个程序注册都会被记录在日志中。如果您的程序成功注册，可找到匹配的日志条目。

确定程序已正确注册后，即可开始迭代业务逻辑编码。

完全限定名称映射

本节包含用于现代化应用程序的 AWS Blu Age 和第三方完全限定的名称映射列表。

AWS Blu Age 完全限定名称映射

短名称	完全限定名称
`CallBuilder`	`com.netfective.bluage.gapwalk.runtime.statements.CallBuilder`
`Configuration`	`com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration`
`ConfigurationBuilder`	`com.netfective.bluage.gapwalk.datasimplifier.configuration.ConfigurationBuilder`

短名称	完全限定名称
ExecutionController	com.netfective.bluage.gapwalk.rt.call.ExecutionController
ExecutionControllerImpl	com.netfective.bluage.gapwalk.rt.call.internal.ExecutionControllerImpl
File	com.netfective.bluage.gapwalk.rt.io.File
MainProgramRunner	com.netfective.bluage.gapwalk.rt.call.MainProgramRunner
Program	com.netfective.bluage.gapwalk.rt.provider.Program
ProgramContextStore	com.netfective.bluage.gapwalk.rt.context.ProgramContextStore
ProgramRegistry	com.netfective.bluage.gapwalk.rt.provider.ProgramRegistry
Record	com.netfective.bluage.gapwalk.datasimplifier.data.Record
RecordEntity	com.netfective.bluage.gapwalk.datasimplifier.entity.RecordEntity
RuntimeContext	com.netfective.bluage.gapwalk.rt.context.RuntimeContext
SimpleStateMachineController	com.netfective.bluage.gapwalk.rt.statemachine.SimpleStateMachineController

短名称	完全限定名称
StateMachineController	com.netfective.bluage.gapwalk.rt.statemachine.StateMachineController
StateMachineRunner	com.netfective.bluage.gapwalk.rt.statemachine.StateMachineRunner

第三方完全限定名称映射

短名称	完全限定名称
@Autowired	org.springframework.beans.factory.annotation.Autowired
@Bean	org.springframework.context.annotation.Bean
BeanFactory	org.springframework.beans.factory.BeanFactory
@Component	org.springframework.stereotype.Component
ConfigurableApplicationContext	org.springframework.context.ConfigurableApplicationContext
@Import	org.springframework.context.annotation.Import
@Lazy	org.springframework.context.annotation.Lazy

数据简化器

在大型机和中型机系统（以下主题中称为“传统”系统）上，常用的编程语言（例如 COBOL、PL/I 或 RPG）提供对内存的低级别访问。这种访问侧重于通过本机类型（例如分区、打包或字母数字，也可能包括组或数组聚合）访问内存布局。

在给定程序中，通过类型化字段和直接访问字节（原始内存）来访问给定内存的两种方式共存。例如，COBOL 程序会将参数作为连续的字节集（LINKAGE）传递给调用者，或者以相同的方式从文件（记录）中读取/写入数据，同时使用在抄写本中组织的类型化字段来解释此类内存范围。

这种对内存的原始访问和结构化访问的组合、对精确的字节级内存布局的依赖以及传统类型（例如分区或打包）是在 Java 编程环境中不容易实现的非本机特征。

作为将传统程序现代化为 Java 的 AWS Blu Age 解决方案的一部分，Data Simplifier 库为现代化的 Java 程序提供了此类结构，并以 Java 开发人员尽可能熟悉的方式公开这些结构（获取器/设置器、字节数组、基于类的字节数组）。数据简化器是从这些程序生成的现代化 Java 代码的核心依赖项。

为简单起见，以下大部分说明都基于 COBOL 结构，但您可以对 PL1 和 RPG 数据布局现代化使用相同的 API，因为大多数概念都是相似的。

主题

主要的类

为了便于阅读，本文档使用了 AWS Blu Age API 接口和类的 Java 简称。有关更多信息，请参阅所讨论的 Java 类型的 FQN。

低级别内存表示

在最低级别上，内存（以快速、随机方式访问的连续字节范围）由 `Record` 接口表示。该接口本质上是固定大小字节数组的抽象。因此，它提供了能够访问或修改底层字节的 setter 和 getter。

结构化数据表示

要表示结构化数据，例如 COBOL DATA DIVISION 中的“01 data item”或“01 copybook”，则使用 `RecordEntity` 类的子类。它们通常不是手写的，而是由 AWS 蓝光时代的现代化工具从相应的遗留

结构中生成的。了解它们的主要结构和 API 也仍有用，可方便您了解现代化程序中的代码如何使用它们。就 COBOL 而言，该代码是从 PROCEDURE DIVISION 生成的 Java。

生成的代码使用 `RecordEntity` 子类表示每个“01 data item”；组成它的每个基本字段或聚合都表示为一个私有 Java 字段，采用树状结构进行组织（每个项都有一个父项，根项除外）。

为便于说明，以下是一个 COBOL 数据项示例，后面是相应的 B AWS lu Age 生成的对其进行现代化改造的代码：

```
01 TST2.
 02 FILLER PIC X(4).
 02 F1     PIC 9(2) VALUE 42.
 02 FILLER PIC X.
 02        PIC 9(3) VALUE 123.
 02 F2     PIC X VALUE 'A'.
```

```
public class Tst2 extends RecordEntity {

    private final Group root = new Group(getData()).named("TST2");
    private final Filler filler = new Filler(root,new AlphanumericType(4));
    private final Elementary f1 = new Elementary(root,new ZonedType(2, 0, false),new
 BigDecimal("42")).named("F1");
    private final Filler filler1 = new Filler(root,new AlphanumericType(1));
    private final Filler filler2 = new Filler(root,new ZonedType(3, 0, false),new
 BigDecimal("123"));
    private final Elementary f2 = new Elementary(root,new
 AlphanumericType(1),"A").named("F2");



    /**
     * Instantiate a new Tst2 with a default record.
     * @param configuration the configuration
     */
    public Tst2(Configuration configuration) {
        super(configuration);
        setupRoot(root);
    }
    /**
     * Instantiate a new Tst2 bound to the provided record.
     * @param configuration the configuration
     * @param record the existing record to bind
     */
    public Tst2(Configuration configuration, RecordAdaptable record) {
```

```
        super(configuration);
        setupRoot(root, record);
    }

    /**
     * Gets the reference for attribute f1.
     * @return the f1 attribute reference
     */
    public ElementaryRangeReference getF1Reference() {
        return f1.getReference();
    }

    /* *
     * Getter for f1 attribute.
     * @return f1 attribute
     */
    public int getF1() {
        return f1.getValue();
    }

    /**
     * Setter for f1 attribute.
     * @param f1 the new value of f1
     */
    public void setF1(int f1) {
        this.f1.setValue(f1);
    }
    /**
     * Gets the reference for attribute f2.
     * @return the f2 attribute reference
     */
    public ElementaryRangeReference getF2Reference() {
        return f2.getReference();
    }

    /**
     * Getter for f2 attribute.
     * @return f2 attribute
     */
    public String getF2() {
        return f2.getValue();
    }

    /**
```

```
     * Setter for f2 attribute.
     * @param f2 the new value of f2
     */
    public void setF2(String f2) {
        this.f2.setValue(f2);
    }
}
```

基本字段

类 Elementary（或者 Filler，如果未命名）的字段代表传统数据结构的“叶子”。这些字段与底层字节的连续跨度（“范围”）相关联，并且通常具有表示如何解析和修改这些字节（通过分别对字节数组中的值“解码”和“编码”）的类型（可能已参数化）。

所有基本类型都是 RangeType 的子类。常见的类型如下：

COBOL 类型	数据简化器类型
PIC X(n)	AlphanumericType
PIC 9(n)	ZonedType
PIC 9(n) COMP-3	PackedType
PIC 9(n) COMP-5	BinaryType

聚合字段

聚合字段组织其内容（其他聚合或基本字段）的内存布局，本身没有基本类型。

Group 字段表示内存中的连续字段。其中包含的每个字段在内存中的布局顺序相同，第一个字段位于偏移 0（相对于组字段在内存中的位置），第二个字段处于偏移 0 + (size in bytes of first field)，依此类推。它们用于表示同一包含字段下的 COBOL 字段序列。

Union 字段表示访问同一内存的多个字段。其中包含的每个字段都布局在偏移 0（相对于内存中的 Union 字段位置）。例如，它们用于表示 COBOL“REDEFINES”结构（第一个 Union 子项是重新定义的数据项，第二个子项是对第一个的重定义，以此类推）。

数组字段（Repetition 的子类）表示其子字段（无论本身是聚合还是基本项目）在内存中的布局重复。它们在内存中布局了给定数量的此类子布局，每个字段都位于偏移 index * (size in bytes of child)，用于表示 COBOL“OCCURS”结构。

基本数据类型

在某些现代化案例中，“Primitives” 也可以用来呈现独立的 “根” 数据项。它们在使用上非常相似，RecordEntity但不是来自它，也不是基于生成的代码。相反，它们由 AWS Blu Age 运行时作为Primitive接口的子类直接提供。提供的此类的示例包括 Alphanumeric 或 ZonedDecimal。

数据绑定和访问

结构化数据和底层数据之间的关联可以通过多种方式实现。

用于实现关联的一个重要接口是 RecordAdaptable，该接口用于获取提供 RecordAdaptable 底层数据的“可写视图”的 Record。如下所示，多个类实现了 RecordAdaptable。相反， AWS Blu Age API 和操纵低级内存（例如程序参数、文件 I/O 记录、CICS commarea、分配的内存...）的代码通常会期望使用RecordAdaptable作为该内存的句柄。

在 COBOL 现代化案例中，大多数数据项都与内存相关联，内存在相应程序执行的生命周期内固定。为此，RecordEntity 子类在生成的父对象（程序 Context）中实例化一次，并根据 RecordEntity 字节大小负责实例化其底层 Record。

在其他 COBOL 情况下，例如将 LINKAGE 元素与程序参数相关联，或者对 SET ADDRESS OF 结构进行现代化，必须将 RecordEntity 实例与提供的 RecordAdaptable 关联。为此，可使用两种机制：

- 如果 RecordEntity 实例已经存在，则可以使用 RecordEntity.bind(RecordAdaptable) 方法（继承自 Bindable）使此实例“指向”此 RecordAdaptable。然后，在 RecordEntity 上调用的任何 getter 或 setter 都将由底层 RecordAdaptable 字节支持（字节读取或写入）。
- 如果要将 RecordEntity 实例化，生成的接受 RecordAdaptable 的结构可用。

相反，可以访问结构化数据当前绑定的 Record。为此，RecordEntity 实现了 RecordAdaptable，因此可以在任何此类实例上调用 getRecord()。

最后，许多 COBOL 或 CICS 动词都需要访问单个字段来进行读写。RangeReference 类用于表示此类访问。它的实例可以从 RecordEntity 生成的 getXXXReference() 方法（XXX 即访问的字段）中获取，然后传递给运行时方法。RangeReference 通常用于访问整个 RecordEntity 或Group，而其子类 ElementaryRangeReference 表示对 Elementary 字段的访问。

请注意，上面的大多数观察结果都适用于Primitive子类，因为它们努力实现与 AWS Blu Age 运行RecordEntity时提供的行为相似的行为（而不是生成的代码）。为此，Primitive 的所有子类实现 RecordAdaptable、ElementaryRangeReference 和 Bindable，以便可以代替 RecordEntity 子类和基本字段。

所讨论的 Java 类型的 FQN

下表显示了本节中所讨论 Java 类型的完全限定名称。

短名称	完全限定名称
Alphanumeric	com.netfective.bluage.gapwalk.datasimplifier.elementary.Alphanumeric
AlphanumericType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.AlphanumericType
BinaryType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.BinaryType
Bindable	com.netfective.bluage.gapwalk.datasimplifier.data.Bindable
Elementary	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Elementary
ElementaryRangeReference	com.netfective.bluage.gapwalk.datasimplifier.entity.ElementaryRangeReference
Filler	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Filler
Group	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Group

短名称	完全限定名称
PackedType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.PackedType
Primitive	com.netfective.bluage.gapwalk.datasimplifier.elementary.Primitive
RangeReference	com.netfective.bluage.gapwalk.datasimplifier.entity.RangeReference
RangeType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.RangeType
Record	com.netfective.bluage.gapwalk.datasimplifier.data.Record
RecordAdaptable	com.netfective.bluage.gapwalk.datasimplifier.data.RecordAdaptable
RecordEntity	com.netfective.bluage.gapwalk.datasimplifier.entity.RecordEntity
Repetition	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Repetition
Union	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Union

短名称	完全限定名称
ZonedDecimal	com.netfective.bluage.gapwalk.datasimplifier.elementary.ZonedDecimal
ZonedType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.ZonedType

AWS Blu Age 运行时配置和配置文件

Velocity 框架和客户端代码是使用 Spring Boot 框架的 Web 应用程序。它利用 Spring 功能来提供配置，其中包含多个可能的位置和优先级规则。提供许多其他文件（例如 groovy 脚本、sql 等）时，也有类似的优先级规则。

Velocity 框架还包含其他可选的 Web 应用程序，可以根据需要选择加入这些应用程序。

主题

应用程序配置基础知识

处理应用程序配置的默认方法是使用应用程序服务器`config`文件夹中提供的专用 YAML 文件。有两个主要的 YAML 配置文件：

- `application-main.yaml`
- `application-`*`profile`*`.yaml`（其中 *`profile`* 值是在应用程序生成期间设置的）。

第一个文件用于配置框架，即 Gapwalk-application.war，第二个文件用于专门针对客户端应用程序的其他选项。可以使用 Spring 配置文件来实现：Gapwalk 应用程序使用 main 配置文件，而客户端应用程序使用 *profile* 配置文件。

下面示例显示了典型的主 YAML 文件。

```
########################################
##### JICS datasource configuration #####
########################################
datasource:
  jicsDs:
    driver-class-name : org.postgresql.Driver
    url: jdbc:postgresql://localhost/jics
    username: jics
    password: jics
    type : org.postgresql.ds.PGSimpleDataSource

#####################################################
##### Embedded Bluesam datasource configuration #####
#####################################################
  bluesamDs :
    driver-class-name : org.postgresql.Driver
    url : jdbc:postgresql://localhost/bluesam
    username : bluesam
    password : bluesam
    type : org.postgresql.ds.PGSimpleDataSource

########################################
##### Embedded Bluesam configuration #####
########################################
bluesam :
  remote : false
  cache : ehcache
  persistence : pgsql #pgsql, mssql, xodus...
  ehcache:
    resource-pool:
      size: 4GB
  write-behind:
```

下面示例显示了典型的客户端 YAML 文件。

```
# Logback context logger integration.
logging.config : classpath:logback-[redacted].xml
# Limits Spring logger output.
logging.level.org.springframework.beans.factory.support.DefaultListableBeanFactory : WARN
logging.level.org.springframework.statemachine : WARN
# If the datasource support mode is not static-xa, spring JTA transactions autoconfiguration must me disabled
spring.jta.enabled : false

spring:
  aws:
    client:
      datasources:
        names: primary
        primary:
          secret: arn:aws:secretsmanager:[redacted]

spring.jta.atomikos.datasource.primary.unique-resource-name: primary
spring.jta.atomikos.datasource.primary.xa-data-source-class-name: org.postgresql.xa.PGXADataSource
spring.jta.atomikos.datasource.primary.maxPoolSize: 20
spring.jta.atomikos.datasource.primary.autoCommit: false
```

有关 YAML 文件内容的信息，请参阅启用属性。

应用程序优先级

Spring 优先规则适用于上述配置文件。请注意：

- application-mainYAML 文件以默认值出现在 Gapwalk 主战争文件中，config文件夹中的文件取代了它。
- 这点应同样适用于客户端应用程序配置
- 其他参数可以在服务器启动时在命令行中传递。他们会覆盖 YAML 的。

有关更多信息，请参阅 Spring Boot 官方文档。

用于数据库的 JNDI

数据库配置可能与 Tomcat 中 context.xml 文件中的 JNDI 一并提供。任何这样的配置都会覆盖 YAML 的配置。不过，请注意，使用这类配置不允许将您的凭证封装在 Secret Manager 中（请参阅下文）。

以下示例显示了 JICS 和 BluSam 数据库的示例配置。

```
<Resource auth="Container" driverClassName="org.postgresql.Driver" initialSize="0"
 maxIdle="5"
   maxOpenPreparedStatements="-1" maxTotal="10" maxWaitMillis="-1" name="jdbc/jics"
   poolPreparedStatements="true" testOnBorrow="false" type="javax.sql.DataSource"
   url="jdbc:postgresql://XXXX.rds.amazonaws.com:5432/XXXX" username="XXXX"
 password="XXXX" />
```

jdbc/jics

将jdbc/jics用于 JICS 数据库，jdbc/bluesam（注意 'e'）用于 blusam 数据库。

url="jdbc:postgresql://XXXX.rds.amazonaws.com:5432/XXXX" username="XXXX" password="XXXX"

数据库 URL、用户名和密码。

使用 AWS 秘密

使用 AWS 密钥可以进一步保护某些包含凭据的资源配置。这个想法是将关键数据存储在 AWS 机密中，并在YAML配置中引用该密钥，以便在tomcat启动时即时选择秘密内容。

适用于 Aurora 的密钥

Aurora 数据库配置（用于 jics、blusam、客户数据库等）将使用内置的数据库密钥，该密钥将自动填充相应数据库中的所有相关字段。

> **Note**
>
> dbname 密钥是可选的，可以包含在或不包含在密钥中，具体取决于您的数据库配置。您可以手动将其添加到那里，也可以通过向 YAML 文件提供名称来添加。

其他密钥

其他密钥用于具有单一密码的资源（尤其是受密码保护的 redis 缓存）。在这种情况下，必须使用其他密钥类型，只有一个 password 键。

YAML 对密钥的引用

application-main.yaml可以引用各种资源的秘密 ARN。重要资源密钥 ARN 如下：

- 带有 JICS 数据库凭证 spring.aws.jics.db.secret
- JICS TS 使用队列 Redis 凭证 spring.aws.client.jics.queues.ts.redis.secret
- Blusam 数据库凭证 (spring.aws.client.bluesam.db.secret)
- Blusam 缓存密码 (spring.aws.client.bluesam.redis.secret)
- Blusam 锁缓存密码 (spring.aws.client.bluesam.locks.redis.secret)

以下示例说明如何在 YAML 文件中声明这些密钥。

```
spring:
  aws:
    client:
      bluesam:
        locks:
          redis:
            secret: arn:aws:secretsmanager:XXXX
        db:
          dbname: bluesam
          secret: arn:aws:secretsmanager:XXXX
        redis:
          secret: arn:aws:secretsmanager:XXXX
      jics:
         queues:
          ts:
           redis:
            secret: arn:aws:secretsmanager:XXXX
    jics:
      db:
        secret: arn:aws:secretsmanager:XXXX
```

dbname: bluesam

在此示例中，数据库的名称不在密钥中，而是在此处提供。

客户端 application-*profile*.yaml 可以引用客户端数据库的密钥 ARN。这需要一个额外的属性来列出数据源，如下例所示：

```
spring:
  aws:
    client:
      datasources:
        names: primary,host
        primary:
          secret: arn:aws:secretsmanager:XXXX
        host:

          secret: arn:aws:secretsmanager:XXXX
```

names: primary,host

本例中包含名为 primary 和 host 的两个客户端数据源，每个数据源都有各自的数据库和凭证。

dbname: mydb

在此示例中，“host""数据库的名称不在密钥中，因此在此处提供，而“primary”数据库在密钥中。

不支持 XA 的密钥

- 引擎 (postgres/oracle/db2/mssql)
- port
- dbname
- 当前架构
- username
- password
- url

除了 `spring.aws.rds.ssl.cert-path` yml 属性外，`postgres`只有值为 (`disable/allow/prefer/require/verify-ca/verify-full`) 的`sslMode`密钥才允许与 SSL 连接。

XA 支持的密钥

如果客户端数据库使用 XA，则通过机密值支持子 xa-properties。

- host
- port
- dbname
- 当前架构
- username
- password
- url
- SSLConnection（对/错）

但是，对于其他 xa-properties（例如maxPoolSize或driverType），仍spring.jta.atomikos.datasource.XXXX.unique-resource-name必须提供常规 YAML 密钥。

密钥值会覆盖 YAML 属性。

其他文件（groovy、sql 等）

客户项目使用的其他文件使用与 Spring 配置文件相似的优先级规则。示例：

- Groovy 脚本是 `scripts` 文件夹或子文件夹中的 `.groovy` 文件。
- SQL 脚本是 `sql` 文件夹或子文件夹中的 `.sql` 文件。
- 进程守护程序脚本是 `daemons` 文件夹或子文件夹中的 `.groovy` 文件。
- 查询数据库映射文件是 `sql` 文件夹或子文件夹中名为 `queries-database.mapping` 的文件。
- Jasper 模板是 `templates` 文件夹或子文件夹中的 `.jrxml` 文件。
- 数据集目录是 `catalog` 文件夹中的 `.json` 文件。
- Lnk 文件是 `lnk` 文件夹中的 `.json` 文件。

所有这些位置都可以通过系统属性或客户端 yml 属性进行覆盖。

- 对于 Groovy 脚本：`configuration.scripts`
- 对于 SQL 脚本：`configuration.sql`
- 对于进程守护程序脚本：`configuration.daemons`
- 对于查询数据库映射文件：`configuration.databaseMapping`
- 对于 Jasper 模板：`configuration.templates`
- 对于数据集目录：`configuration.catalog`
- 对于 Lnk 文件：`configuration.lnk`

如果找不到该属性，则文件将从上述默认位置获取。首先将以 tomcat 工作目录作为根目录进行查找，最后在应用程序 war 文件中进行查找。

其他 Web 应用程序

Velocity 框架的 `webapps-extra` 文件夹中包含其他 Web 应用程序。默认情况下，tomcat 服务器不为这些应用程序提供服务。

是否选择加入这些 Web 应用程序取决于现代化项目，可通过将所需的 war 文件从 webapps-extra 文件夹移动到 webapps 文件夹来完成。之后，war 文件将在下次启动时由 tomcat 服务器提供服务。

对于每增加一次 war，也可以在 YAML 配置文件中添加一些特定于项目的额外配置，如application-main.yml文件中所做的那样，如上所述。其他 war 如下：

- gapwalk-utility-pgm.war：包含对 ZOS 实用程序的支持并将 application-utility-pgm.yaml 用作其配置。
- gapwalk-cl-command.war：包含对 AS/400 实用程序的支持并将 application-cl-command.yaml 用作其配置。
- gapwalk-hierarchical-support.war: 包含 IMS/MFS 事务支持并将 application-jhdb.yaml 用作其配置。

启用属性

Spring Boot 应用程序application-main.yml中有一个配置文件，我们在其中定义了不同类型的属性，例如侦听端口、数据库连接等。

主题

YML 表示法

在以下文档中，诸如parent.child1.child2=true这样的属性以 YAML 格式写成，如下所示。

```
parent:
 child1:
   child2: true
```

快速入门/用例

以下用例显示了适用的键和值的示例。

- 默认 application-main.yml

```
---
#### DEFAULT APPLICATION-MAIN.YML FILE      #####
#### SHOWING USEFUL CONFIGURATION ELEMENTS #####
#### SHOULD BE OVERRIDDEN AND EXTERNALIZED   #####

##################################
##### Logging configuration #####
##################################
logging:
  config: classpath:logback-main.xml
  level.org.springframework.beans.factory.support.DefaultListableBeanFactory : WARN

##################################
##### Spring configuration #####
##################################
spring:
  quartz:
    auto-startup: false
    scheduler-name: Default
    properties:
      org.quartz.threadPool.threadCount: 1
  jta:
    enabled: false
    atomikos.properties.maxTimeout : 600000
    atomikos.properties.default-jta-timeout : 100000
  jpa:
# DISABLE OpenEntityManagerInViewInterceptor
    open-in-view: false
    # Fix Postgres JPA Error:
    # Method org.postgresql.jdbc.PgConnection.createClob() is not yet implemented.
    properties.hibernate.temp.use_jdbc_metadata_defaults : false
#####################################
##### Jics tables configuration #####
#####################################

    # The dialect should match the jics datasource choice
    database-platform : org.hibernate.dialect.PostgreSQLDialect #
 org.hibernate.dialect.PostgreSQLDialect, org.hibernate.dialect.SQLServerDialect

    # those properties can be used to create and initialize jics tables
 automatically.
#    properties:
#      hibernate:
```

```
#      globally_quoted_identifiers: true
#        hbm2ddl:
#           import_files_sql_extractor :
 org.hibernate.tool.hbm2ddl.MultipleLinesSqlCommandExtractor
#           import_files : file:./setup/initJics.sql
#           auto : create

##########################
###### Level 2 cache #####
##########################
#        cache:
#          use_second_level_cache: true
#          use_query_cache: true
#          region:
#            factory_class: org.hibernate.cache.ehcache.EhCacheRegionFactory
#      javax:
#        persistence:
#          sharedCache:
#            mode: ENABLE_SELECTIVE
##########################
###### Redis settings #####
##########################
  session:
    store-type: none #redis

########################################
##### JICS datasource configuration #####
########################################
datasource:
  jicsDs:
    driver-class-name : org.postgresql.Driver # org.postgresql.Driver,
 com.microsoft.sqlserver.jdbc.SQLServerDriver
    url: jdbc:postgresql://localhost/jics # jdbc:postgresql://localhost:5433/jics,
 jdbc:sqlserver://localhost\SQLEXPRESS:1434;databasename=jics;
    username: jics
    password: jics
    type : org.postgresql.ds.PGSimpleDataSource #
 org.postgresql.ds.PGSimpleDataSource,
 com.microsoft.sqlserver.jdbc.SQLServerDataSource

#####################################################
##### Embedded Bluesam datasource configuration #####
#####################################################
  bluesamDs :
```

```
    driver-class-name : org.postgresql.Driver # org.postgresql.Driver,
 com.microsoft.sqlserver.jdbc.SQLServerDriver
    url : jdbc:postgresql://localhost/bluesam # jdbc:postgresql://localhost:5433/
jics, jdbc:sqlserver://localhost\SQLEXPRESS:1434;databasename=jics;
    username : bluesam
    password : bluesam
    type : org.postgresql.ds.PGSimpleDataSource #
 org.postgresql.ds.PGSimpleDataSource,
 com.microsoft.sqlserver.jdbc.SQLServerDataSource

########################################
##### Embedded Bluesam configuration #####
########################################
bluesam :
  remote : false
  cache : ehcache
  persistence : pgsql #pgsql, mssql, xodus...
  ehcache:
    resource-pool:
      size: 4GB
  write-behind:
    enabled: true
  pgsql :
    dataSource : bluesamDs

#########################
##### Jics settings #####
#########################
rabbitmq.host: localhost
jics:
  cache: false #redis
  resource-definitions.store-type: jpa # default value: jpa, other possible value:
 redis
  redis.hostname: 127.0.0.1 # Redis server host.
  redis.password: redis # Login password of the redis server.
  redis.port: 6379 # Redis server port.
  redis.username: # Redis username
  redis.mode: standalone # Redis mode. Possible values: standalone, cluster
jics.disableSyncpoint : false
#jics.initList:
#jics.parameters.datform: DDMMYY
#jics.parameters.applid: VELOCITY
#jics.parameters.sysid: CICS
#jics.parameters.eibtrmid: TERM
```

```
#jics.parameters.userid: MYUSERID
#jics.parameters.username: MYUSERNAME
#jics.parameters.opid: XXX
#jics.parameters.cwa.length: 0
#jics.parameters.netname: MYNETNAME
#jics.parameters.jobname: MJOBNAME
#jics.parameters.sysname: SYSNAME

################################################
##### Jics RunUnitLauncher pool settings #####
################################################
#jics.runUnitLauncherPool.enable: false
#jics.runUnitLauncherPool.size: 20
#jics.runUnitLauncherPool.validationInterval: 1000

#########################
##### Jhdb settings #####
#########################
#jhdb.lterm: LTERMVAL
#jhdb.identificationCardData: SomeIDData

###################################
##### DateHelper configuration ####
###################################
#forcedDate: "2013-08-26T12:59:58+01:57"

#############################
##### Sort configuration ####
#############################
#externalSort.threshold: 256MB

###################################
##### Server timeout (10 min)  ####
###################################
spring.mvc.async.request-timeout: 600000

#############################
##### DATABASE STATISTICS  ####
#############################
databaseStatistics : false

######################
##### CALLS GRAPH ####
######################
```

```
callGraph : false

################################
##### SQL SHIFT CODE POINT ####
################################
# Code point 384 match unicode character \u0180
sqlCodePointShift : 384

################################
##### LOCK TIMEOUT RECORD  ####
################################
# Blu4IV record lock timeout
lockTimeout : 100

################################
##### REPORTS OUTPUT PATH ####
################################
reportOutputPath: reports

################################
##### TASK EXECUTOR       ####
################################
taskExecutor:
  corePoolSize: 5
  maxPoolSize: 10
  queueCapacity: 50
  allowCoreThreadTimeOut: false

################################
##### PROGRAM NOT FOUND   ####
################################
stopExecutionWhenProgNotFound: false

#########################################################
#####  DISP DEFAULT VALUE (to be removed one day) ####
#########################################################
defaultKeepExistingFiles: true

#####################################
#####  JOBQUEUE CONFIGURATION    ####
#####################################
jobqueue:
  api.enabled: false
  impl: none # possible values: quartz, none
```

```
  schedulers:  # list of schedulers
    -
      name: queue1
      threadCount: 5
    -
      name: queue2
      threadCount: 5

################################################################################
##### QUERY BUILDING                                                        ##
# useConcatCondition : false by default
# if true, in the query, the where condition is build with key concatenation  ##
################################################################################
# query.useConcatCondition: true
----
```

- 在 LISTCAT 命令中使用长度可变的文件

```


[**/*.*]
encoding=IBM930
reencoding=false

[global]
listcat.variablelengthpreprocessor.enabled=true
listcat.variablelengthpreprocessor.type=rdw
# use "rdw" if your .listcat file contains a set of records (RDW)
# use "bdw" if your .listcat file contains a set of blocks (bdw)
```

- 在 LOAD/UNLOAD 实用程序中提供空字节指示符值

```


# Unload properties
# For date/time: if use database configuration is enabled, formats are ignored
# For nbi; use hexadecimal syntax to specify the byte value
# - When the value is null in database : the value dumped to the file is filled by
 low value characters and the NBI is
# equal to the byte 6F (the ? character)
# - When the value is not null in database and the column is nullable: the NBI is
 equal to the byte 00 (low value) and NOT
# equal to the byte 40 (space)
```

```
unload:
  sqlCodePointShift: 0
  nbi:
    whenNull: "6F"
    whenNotNull: "00"
  useDatabaseConfiguration: false
  format:
    date: MM/dd/yyyy
    time: HH.mm.ss
    timestamp: yyyy-MM-dd-HH.mm.ss.SSSSSS
```

主应用程序的可用属性

下表提供了键/值参数的详细视图。

键	Type	默认值	描述
`logging.config`	路径	classpath:logback-main.xml	用于引用 logback 配置文件的标准密钥。其他标准日志记录密钥也可用。
`spring.jta.enabled`	布尔值	false	标准密钥。如果数据源支持模式不是 static-xa，则必须禁用 Spring JTA 事务自动配置。
`datasource.jicsDs + -driver-class-name + -url + -username + -password + -type`	带子键的标准 Spring 数据源		包含 Jics 数据库的连接信息。此外，强烈建议使用 AWS 密钥，如 xref:../configuration/configuration.adoc[Configuration] 中所述。
`datasource.bluesamDs + -driver-cl`	带子键的标准 Spring 数据源		包含 Blusam 数据库的连接信息。此外，强烈建议使用

键	Type	默认值	描述
ass-name + -url + -username + -password + -type			AWS 密钥，如 xref:../configuration/configuration.adoc[Configuration] 中所述。
bluesam.disabled	布尔值	false	是否完全禁用 blusam。
bluesam.cache	字符串		如果未设置，则不会使用 blusam 缓存。可能的值（缓存实现）为 ehcache 和 redis。
forcedDate	字符串		强制将日期（如果有）更改为提供的日期。
frozenDate	布尔值	true	指定是否冻结日期。仅在设置了 forcedDate 时适用。
externalSort.threshold	数据大小（例如 12MB）		排序阈值：何时切换到外部（合并）排序。
jics.parameters.datform	字符串	MMDDYY	日期形式。

键	Type	默认值	描述
`jics.initList`	字符串		初始化 jics 列表，用逗号分隔。指定时，该键定义列表中以逗号分隔的名称，以便在 tomcat 启动时在 CICS 列表中激活。示例值：`$UUU,DFH$IVPL,PEZ1` 。这将级联到这些列表中包含的组及其基础资源定义，之后会对运行时可见。默认为空。
`jics.parameters.applid`	字符串	VELOCITY	用于标识 JICS 中的应用程序的 appid（至少 4 个字符，无最大长度限制）。
`jics.parameters.sysid`	字符串	CICS	系统标识 (SYSID)。
`jics.parameters.eibtrmid`	字符串	TERM	终端标识符（最多 4 个字符，最少 1 个字符）。
`jics.parameters.userid`	字符串		用户 ID（最多 8 个字符，无最小字符数限制）。如果未提供任何值（默认为空白），则使用 HTTP 会话 ID 作为用户 ID。
`jics.parameters.username`	字符串	MYUSERNAME	用户 ID（最多 10 个字符，最少 1 个字符）。

键	Type	默认值	描述
`jics.parameters.netname`	字符串	MYNETNAME	网络名称（最多 8 个字符，最少 1 个字符）。
`jics.parameters.opid`	字符串	XXX	3 个字符的操作员标识。
`jics.parameters.jobname``	字符串	MJOBNAME	作业名称。
`jics.parameters.sysname`	字符串	SYSNAME	AS400 系统名称 (sysname)。
`jics.parameters.cwa.length`	数字	0	公共工作区 (cwa) 长度。
`jics.parameters.charset`	字符串	CP037	JICS 全球使用的字符集。
`jics.parameters.tsqimpl`	字符串	bluesam	JICS 临时存储队列 (TSQ) 实现（允许的值为 `bluesam/memory/redis`）
`jics.queues.ts.redis.hostname`	字符串	127.0.0.1	jics 缓存 redis 服务器的主机名。
`jics.queues.ts.redis.port`	number	6379	jics 缓存 redis 服务器的端口。
`jics.queues.ts.redis.password`	字符串	redis	jics 缓存 redis 服务器的密码。

键	Type	默认值	描述
jics.queues.ts.redis.username	字符串		jics 缓存 redis 服务器的用户名。默认值为空（无用户名）。
jics.queues.ts.redis.mode	字符串	standalone	jics 缓存模式。可能的值为 standalone 或 cluster。默认值为 standalone 。
lockTimeout	number	500	锁定超时，以毫秒为单位。
sqlCodePointShift	number		可选。sql 代码点转换。对在将旧版 rdbms 数据迁移到现代 rdbms 时可能遇到的控制字符的代码点进行转换。例如，您可以指定 384 来匹配 unicode 字符 \u0180。
sqlIntegerOverflowAllowed	布尔值	false	指定是否允许 SQL 整数溢出，即是否允许在主机变量中放置更大的值。

键	Type	默认值	描述
database.cursor.overflow.allowed	布尔值	true	指定是否允许光标溢出。设置为 true 时，无论光标位置如何，均可在光标处执行下一次调用。设置为 false 时，会在光标处执行下一次调用之前检查光标是否位于最后一个位置。仅在光标可滚动（敏感或不敏感）的情况下将其启用。
reportOutputPath	字符串	/reports	报告输出路径。
spring.session.store-type	字符串	none	高可用性环境的会话缓存。可能的值为 none 或 redis。默认值为 none。
stopExecutionWhenProgNotFound	布尔值	true	指定在找不到程序时是否停止运行。如果设置为 true，则在找不到程序时中断运行。
forceHR	布尔值	false	指定是否在控制台上或文件输出中使用人类可读的 SYSPRINT。
rollbackOnRTE	布尔值	false	指定是否在运行时异常时回滚隐式运行单元事务。

键	Type	默认值	描述
sctThreadLimit	long	5	用于触发脚本的线程限值。
dataSimplifier.onInvalidNumericData	字符串	reject	解码无效的数字数据时如何应对。允许的值包括：reject、toleratespaces 、toleratespaceslowvalues 、toleratemost 。默认值为 reject。
filesDirectory	字符串		批处理输入/输出文件的目录。
ims.messages.extendedSize	布尔值	false	指定是否在 ims 消息上设置 extendedSize。
defaultKeepExistingFiles	布尔值	false	指定是否设置数据集的默认先前值。
jics.db.ddlScriptLocation	字符串		Jics ddl 脚本位置。允许您使用 .sql 脚本启动 jics 数据库架构。默认情况下为空。例如 ./jics/sql/jics.sql 。
jics.db.schemaTestQueryLocation	字符串		sql 文件的位置，该文件应包含唯一查询，且该查询返回 jics 架构中对象的数量（如果有）。

键	Type	默认值	描述
jics.db.dataScriptLocation	字符串		initJics.sql 脚本的位置，该脚本由 Analyzer 在解析大型机的 CSD 导出时准备。
jics.db.dataTestQueryLocation	字符串		包含单个 sql 查询的 sql 脚本的位置，该查询应返回对象计数（例如：计算 jics 程序表中的记录数）。如果计数等于 0，则将使用 jics.db.dataScriptLocation 脚本加载数据库，否则将跳过数据库加载。
jics.data.dataJsonInitLocation	字符串		
jics.xa.agent.timeout	number		
query.useConcatCondition	布尔值	false	指定键条件是否通过键连接构建。
system.qdecfmt	字符串		
disposition.checkexistence	布尔值	false	指定是否针对配置 DISP SHR 或 OLD 的数据集检查文件是否存在。

键	Type	默认值	描述
useControlMVariable	布尔值	false	指定是否使用 control-M 规范进行变量替换。
card.encoding	字符串	CP1145	卡片编码：与 useControlMVariable`. 一并使用。
mapTransfo.prefixes	字符串	&,@,%%	转换 controlM 变量时要使用的前缀列表。每一个前缀均用逗号隔开。
checkinputfilesize	布尔值	false	指定在文件大小为记录大小倍数的情况下是否要进行检查。
stepFailWhenAbend	布尔值	true	指定在步骤失败或完成执行时是否引发异常中止。
bluesam.fileLoading.commitInterval	number	100000	bluesam 提交间隔。
uppercaseUserInput	布尔值	true	指定用户输入是否必须采用大写形式。
jhdb.lterm	字符串		允许您在进行 IMS 仿真的情况下强制使用通用的逻辑终端 ID。如果未设置，则使用会话 ID。

键	Type	默认值	描述
jhdb.identificationCardData	字符串		用于将某些“操作员识别卡数据”硬编码到 CARD 参数指定的 MID 字段。默认为空白，没有输入限制。
encoding	字符串	ASCII	项目中使用的编码（不在 groovy 文件中）。预期的有效编码包含 CP1047、IBM930、ASCII、U
cl.configuration.context.encoding	字符串	CP297	CL 文件的编码。预期的有效编码包含 CP1047、IBM930、ASCII、U 默认值为 CP297。
cl.zonedMode	字符串	EBCDIC_STRICT	对控制语言 (CL) 命令进行编码或解码的模式。允许的值包括：EBCDIC_STRICT 、EBCDIC_MODIFIED 、AS400。
ims.programs	字符串		要使用的 IMS 程序列表。用分号 (;) 分隔每个参数，用逗号 (,) 分隔每个事务。例如：PCP008,PCT008;PCP054,PCT054;PCP066,PCT066;PCP068,PCT068;

键	Type	默认值	描述
jhdb.configuration.context.encoding	字符串	CP297	Java 分层数据库 (JHDB) 编码。预期的有效编码字符串包含 CP1047、IBM930、ASCII、U
jhdb.metadata.extrapath	字符串	file:./setup/	用于为 psbs 和 dbds 文件夹指定一个特定于运行时的额外根文件夹的配置参数。
jhdb.checkpointPersistence	字符串	none	检查点持久性模式。允许的值包括：none、add、end。使用 add 可在创建新检查点并将其添加到注册表后保留该检查点。使用 end 可在服务器关闭时保留检查点。任何其他值都会禁用持久性。请注意，每次向注册表中添加新的检查点时，所有现有的检查点都将被序列化并且文件会被擦除，而不是添加到文件中现有数据中。这样可能会对性能产生一些影响，具体取决于检查点的数量。

键	Type	默认值	描述
jhdb.checkpointPath	字符串	file:./setup/	如果 jhdb.checkpointPersistence 不是 none，则此参数允许您设置检查点持久性路径（checkpoint.dat 文件存储位置），注册表中包含的所有检查点数据都将序列化并备份到所提供文件夹的文件 (checkpoint.dat) 中。请注意，此备份仅涉及检查点数据（scriptId、stepId、数据库位置和检查点区域）。
jhdb.navigation.cachenexts	number	5000	RDBMS 分层导航中使用的缓存持续时间（以毫秒为单位）。
jhdb.use-db-prefix	布尔值	true	指定是否在 RDBMS 的分层导航中启用数据库前缀。
jhdb.query.limitJoinUsage	布尔值	true	指定是否在 RDBMS 图形上使用限制联接使用参数。
taskExecutor.corePoolSize	number	5	当通过 groovy 脚本启动终端中的事务时，会创建一个新线程。可以使用此参数设置核心池大小。

键	Type	默认值	描述
`taskExecutor.maxPoolSize`	number	10	当通过 groovy 脚本启动终端中的事务时，会创建一个新线程。使用此参数设置池的最大大小（并行线程的最大数量）。
`taskExecutor.queueCapacity`	number	50	当通过 groovy 脚本启动终端中的事务时，会创建一个新线程。使用此参数设置队列大小。（= 达到 `taskExecutor.maxPoolSize` 时待处理事务的最大数量）
`taskExecutor.allowCoreThreadTimeOut`	布尔值	false	指定是否允许核心线程在 JCIS 中超时。即使与非零队列结合使用，也可以实现动态增长和缩小（因为只有在队列已满后，池最大大小才会增长）。
`jics.runUnitLauncherPool.enable`	布尔值	false	指定是否在 JICS 中激活运行单元启动器池。
`jics.runUnitLauncherPool.size`	number	20	JICS 中的运行单元启动器池大小。

键	Type	默认值	描述
jics.runUnitLauncherPool.validationInterval	number	1000	可选：JICS 中运行单元启动器池的验证间隔，以毫秒为单位。
spring.aws.application.credentials	字符串	null	从 JICS 的凭证配置文件中加载 AWS 凭证。
jics.queues.sqs.region	字符串	eu-west-1	在 JICS 中使用的 AWS Simple Queue Service 的 AWS 区域。
mq.queues.sqs.region	字符串	eu-west-3	AWS SQS MQ 服务的 AWS 区域。
quartz.scheduler.stand-by-if-error	布尔值	false	指定当作业调度程序处于待机模式时是否触发作业执行。如果设置为 true，则启用时不会触发作业执行。
databaseStatistics	布尔值	false	指定是否允许 SQL 生成器收集和显示统计信息。
dbDateFormat	字符串	yyyy-MM-dd	数据库目标日期格式。
dbTimeFormat	字符串	HH:mm:ss	数据库目标时间格式。

键	Type	默认值	描述
dbTimestampFormat	字符串	yyyy-MM-dd HH:mm:ss.SSSSSS	数据库目标时间戳格式。
dateTimeFormat	字符串	ISO	dateTimeFormat 描述了如何将数据库日期时间戳类型泄漏到数据简化器实体中。允许的值包括：ISO、EUR、EUR、USA、L
localDateFormat	字符串		本地日期格式列表。使用 \ 分隔每种格式。
localTimeFormat	字符串		本地时间格式列表。使用 \ 分隔每种格式。
localTimestampFormat	字符串		本地时间戳格式列表。使用 \ 分隔每种格式。
pgmDateFormat	字符串	yyyy-MM-dd	日期时间格式
pgmTimeFormat	字符串	HH.mm.ss	程序 (pgm) 执行的时间格式。
pgmTimestampFormat	字符串	yyyy-MM-dd-HH.mm.ss.SSSSSS	时间戳格式。
cacheMetadata	布尔值	true	指定是否缓存数据库元数据。
forceDisableSQLTrimStringType	布尔值	false	指定是否禁用所有 sql 字符串参数的 TRIM。

键	Type	默认值	描述
fetchSize	number		游标的 fetchSize 值。在通过加载/卸载 utils 使用区块获取数据时使用。
check-groovy-file	布尔值	true	指定是否在注册前检查 groovy 文件内容。
qtemp.uuid.length	number	9	QTEMP 的唯一 ID 长度。
qtemp.dblog	布尔值	false	是否启用 QTEMP 数据库日志记录。
qtemp.cleanup.threshold.hours	数字	0	指定何时启用 qtemp.dblog 。数据库分区的生命周期（以小时为单位）。
sort.function	字符串		blu4iv 数据库的排序函数名称。

可选 Web 应用程序的可用属性

您可能需要配置一个或多个可选 Web 应用程序来支持依赖项，例如 z/OS、AS/400 或 IMS/MFS，具体取决于您的现代化应用程序。以下表格包含用于配置每个可选 Web 应用程序的可用键/值参数列表。

gapwalk-utility-pgm.war

此可选的 Web 应用程序包含对 Z/OS 实用程序的支持。

下表提供了该应用程序键/值参数的详细信息。

键	Type	默认值	描述
logging.config	路径	classpath:logback-utility.xml	用于引用 logback 配置文件的标准密钥。

键	Type	默认值	描述
			其他标准日志记录密钥也可用。
`spring.jta.enabled`	布尔值	false	标准密钥。如果数据源支持模式不是 static-xa，则必须禁用 Spring JTA 事务自动配置。
`spring.datasource.primary.jndi-name`	字符串	jdbc/primary	主要数据源的 Java 命名和目录接口 (jndi) 名称（如果使用 JNDI）。
`primary.datasource + -driver-class-name + -url + -username + -password`	带子键的标准 Spring 数据源		如果不使用 JNDI，则包含应用程序数据库的连接信息。必须具有与现代化应用程序 yml 文件相同的配置。 此外，强烈建议使用 AWS 密钥，如 xref:../configuration/configuration.adoc[Configuration] 中所述。
`encoding`	字符串	ASCII	实用程序中使用的编码。预期的有效编码包含 `CP1047、IBM930、ASCII、U`
`sysPunchEncoding`	字符串	ASCII	syspunch 编码字符集。预期的有效编码包含 `CP1047、IBM930、ASCII、U`

键	Type	默认值	描述
zonedMode	字符串	EBCDIC_STRICT	对分区数据类型进行编码或解码的模式。允许的值包括：EBCDIC_STRICT 、EBCDIC_MODIFIED 、AS400。
unload.chunkSize	数字	0	卸载实用程序使用的区块大小。
unload.sqlCodePointShift	数字	0	卸载实用程序的 SQL 代码点转换。运行字符转换进程。如果来自 DB2 目标数据库的是 Postgresql，则该参数是必需的。
unload.columnFiller	字符串	space	卸载实用程序列填充内容。
unload.varCharIsNull	布尔值	false	在 INFTILB 程序中使用此参数，如果设置为 true，则所有具有空（空格）值的不可为空的字段都将返回一个空字符串。
unload.useDatabaseConfiguration	布尔值	false	指定是否在卸载实用程序中使用 application-main.yml 中的日期或时间配置。

键	Type	默认值	描述
unload.format.date	字符串	MM/dd/yyyy	卸载实用程序中使用的日期格式（如果启用 unload.useDatabaseConfiguration ）。
unload.format.time	字符串	HH.mm.ss	卸载实用程序中使用的时间格式（如果启用 unload.useDatabaseConfiguration ）。
unload.format.timestamp	字符串	yyyy-MM-dd-HH.mm.ss.SSSSSS	卸载实用程序中使用的时间戳格式（如果启用 unload.useDatabaseConfiguration ）。
unload.nbi.whenNull	十六进制	6F	数据库中的值为空时，要添加的空字节指示符 (nbi) 值。
unload.nbi.whenNotNull	十六进制	00	数据库中的值不为空时，要添加的空字节指示符 (nbi) 值。
unload.nbi.writeNullIndicator	布尔值	false	指定是否在卸载输出文件中写出空指示符。
unload.fetchSize	数字	0	允许您在卸载实用程序中处理光标时调整提取大小。

键	Type	默认值	描述
treatLargeNumberAsInteger	布尔值	false	指定是否将大数字作为 Integer 处理。默认情况下，大数字作为 BigDecimal 处理。
load.batchSize	数字	0	加载实用程序批次大小。
load.format.localDate	字符串	dd.MM.yyyy\dd/MM/yyyy\yyyy-MM-dd	要使用的加载实用程序本地日期格式。
load.format.localTime	字符串	HH:mm:ss\HH.mm.ss	要使用的加载实用程序本地时间格式。
load.format.dbDate	字符串	yyyy-MM-dd	要使用的加载实用程序数据库格式。
load.format.dbTime	字符串	HH:mm:ss	要使用的加载实用程序数据库时间。
load.sqlCodePointShift	number	0s	加载实用程序的 SQL 代码点转换。运行字符转换进程。如果来自 DB2 目标数据库的是 Postgresql，则该参数是必需的。
forcedDate	字符串		强制将日期（如果有）更改为提供的日期。
frozenDate	布尔值	true	指定是否冻结日期。仅在设置了 forcedDate 时适用。

键	Type	默认值	描述
jcl.type	字符串	MV	.jcl 文件类型。允许的值包括：jcl、vse。如果非 vse jcl 的文件为空，IDCAMS 实用程序 PRINT/REPRO 命令将返回 4。
hasGraphic	布尔值	false	INFUTILB 实用程序是否需要处理 GRAPHIC DB2 列。

gapwalk-cl-command.war

此可选的 Web 应用程序包含对 AS/400 实用程序的支持。

下表提供了该应用程序键/值参数的详细信息。

键	Type	默认值	描述
logging.config	路径	classpath:logback-utility.xml	用于引用 logback 配置文件的标准密钥。其他标准日志记录密钥也可用。
spring.jta.enabled	布尔值	false	标准密钥。如果数据源支持模式不是 static-xa，则必须禁用 Spring JTA 事务自动配置。
spring.datasource.primary.jndi-name	字符串	jdbc/primary	主要数据源的 Java 命名和目录接口 (jndi) 名称（如果使用 JNDI）。

键	Type	默认值	描述
primary.datasource + -driver-class-name + -url + -username + -password	带子键的标准 Spring 数据源		如果不使用 JNDI，则包含应用程序数据库的连接信息。必须具有与现代化应用程序 yml 文件相同的配置。 此外，强烈建议使用 AWS 密钥，如 xref:../configuration/configuration.adoc[Configuration] 中所述。
encoding	字符串	ASCII	实用程序中使用的编码。预期的有效编码包含 CP1047、IBM930、ASCII、U
zonedMode	字符串	EBCDIC_STRICT	对分区数据类型进行编码或解码的模式。允许的值包括：EBCDIC_STRICT 、EBCDIC_MODIFIED 、AS400。
commands-off	字符串		要关闭的命令列表，以逗号分隔。允许的值包括：PGM_BASIC 、RCVMSG、SNDRCVF、CHGV 在禁用或覆盖现有程序时非常有用。PGM_BASIC 是专为调试目的而设计的特定 Velocity 程序。

gapwalk-hierarchical-support.war

此可选 Web 应用程序包含 IMS/MFS 事务支持。

下表提供了该应用程序键/值参数的详细信息。

键	Type	默认值	描述
`logging.config`	路径	classpath:logback-utility.xml	用于引用 logback 配置文件的标准密钥。其他标准日志记录密钥也可用。
`spring.jta.enabled`	布尔值	false	标准密钥。如果数据源支持模式不是 static-xa，则必须禁用 Spring JTA 事务自动配置。
`jhdb.configuration.context.encoding`	字符串		Java 分层数据库 (JHDB) 编码。预期的有效编码字符串包含 CP1047、IBM930、ASCII、U
`jhdb.checkpointPersistence`	字符串	none	检查点持久性模式。允许的值包括：none、add、end。使用 add 可在创建新检查点并将其添加到注册表后保留该检查点。使用 end 可在服务器关闭时保留检查点。任何其他值都会禁用持久性。请注意，每次向注册表中添加新的检查点时，所有现有的检查点都将被序列化并且文件

键	Type	默认值	描述
			会被擦除，而不是添加到文件中现有数据中。这样可能会对性能产生一些影响，具体取决于检查点的数量。

为 Gapwalk 应用程序配置身份验证

本节介绍如何使用 Cognito、Azure AD、Keycloak 等身份提供商 (IdP) 为 Gapwalk 应用程序配置 OAuth2 身份验证。

主题

先决条件

在本教程中，我们将使用 Amazon Cognito 作为 IdP，并使用 planetDemo 作为现代化项目。

您可以使用任何其他外部身份提供者。这些 ClientRegistration 信息必须从您的 IdP 处获取，并且是 gapwalk 身份验证所必需的。有关更多信息，请参阅您的 IdP 的官方文档。

ClientRegistration 信息：

client-id

的 ID，在我们的示例中 ClientRegistration，它将是 PlanetDemo。

client-secret

您的客户端密钥。

授权端点

授权服务器的授权端点 URI。

令牌端点

授权服务器的令牌端点 URI。

jwks 端点

用于获取 JSON Web 密钥 (JWK) 的 URI，其中包含用于验证授权服务器颁发的 JSON Web 签名的密钥。

重定向 URI

授权服务器将最终用户重定向到的 URI（如果授予访问权限）。

Amazon Cognito 设置

首先，我们将创建和配置一个 Amazon Cognito 用户池和用户，并将其与已部署的 gapwalk 应用程序用于进行测试。

> **Note**
>
> 如果您使用的是其他 IdP，则可以跳过此步骤。

创建用户池

1. 前往中的 Amazon Cognito AWS Management Console 并使用您的 AWS 凭证进行身份验证。
2. 选择用户池。
3. 选择创建用户池。
4. 在配置登录体验中，保持 Cognito 用户池的默认提供者类型。您可以选择一个或多个 Cognito 用户池登录选项；现在，选择用户名，然后选择下一步。
5. 在配置安全要求中，选择否 MFA，然后选择下一步，保留默认设置并禁用多因素身份验证。
6. 出于安全考虑，禁用启用自动注册，然后选择下一步。
7. 选择使用 Cognito 发送电子邮件。选择下一步。
8. 在集成应用程序中，为您的用户池选择一个名称。在托管身份验证页面中，选择使用 Cognito 托管 UI。
9. 为简单起见，在域中，选择使用 Cognito 域并输入域前缀；例如，`https://planetsdemo`。必须将演示应用程序添加为客户端。

1. 在初始应用程序客户端中，选择机密客户端。输入应用程序客户端名称（例如 `planetsdemo`），然后选择生成客户端密钥。
2. 在允许的回调 URL 中，输入用户在通过身份验证后要重定向到的 url。也可以使用临时的 URL (http://localhost:8080/planetsdemo)，然后稍后再进行编辑。
3. 在高级应用程序客户端设置和属性读写权限部分中保留默认值。
4. 选择下一步。

10. 在检查并创建中，验证您的选择，然后选择创建用户池。

有关更多信息，请参阅创建用户池。

创建用户

鉴于自助注册已禁用，请创建一个 Amazon Cognito 用户。导航到 AWS Management Console中的 Amazon Cognito。选择您创建的用户池，然后在用户中，选择创建用户。

在用户信息中，选择发送电子邮件邀请，输入用户名和电子邮件地址，然后选择生成密码。选择创建用户。

将 Amazon Cognito 集成到 Gapwalk 应用程序中

现在，您的 Amazon Cognito 用户池和用户已准备就绪，请访问现代化应用程序`main-application.yml`的文件并添加以下代码：

```
spring:
  security:
    oauth2:
      client:
        registration:
          cognito:
            client-id: client-id
            client-name: client-name
            client-secret: client-secret
            provider: cognito
            authorization-grant-type: authorization_code
            scope: openid
            redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"
        provider:
          cognito:
            issuerUri: ${gapwalk-application.security.issuerUri}
```

```
            authorization-uri: ${gapwalk-application.security.domainName}/oauth2/
authorize
            jwk-set-uri: ${gapwalk-application.security.issuerUri}/.well-known/
jwks.json
            token-uri: ${gapwalk-application.security.domainName}/oauth2/token
            user-name-attribute: cognito:username
      resourceserver:
        jwt:
          jwk-set-uri: ${gapwalk-application.security.issuerUri}/.well-known/jwks.json

gapwalk-application.security: enabled
gapwalk-application.security.identity: oauth

gapwalk-application.security.issuerUri: https://cognito-idp.region.amazonaws.com/pool-
id
gapwalk-application.security.domainName: your-cognito-domain
```

按如下所述替换以下占位符：

1. 前往中的 Amazon Cognito AWS Management Console 并使用您的 AWS 凭证进行身份验证。
2. 选择用户池，然后选择您创建的用户池。您可以在用户池 ID 中找到您的 *pool-id*。
3. 选择应用程序集成，您可以在其中找到自己的应用程序，*your-cognito-domain*然后转到应用程序客户端和分析，然后选择您的应用程序。
4. 在应用程序客户端：yourApp 中，您可以找到 *client-name*、*client-id* 和 *client-secret*（显示客户端密钥）。
5. *region-id* 对应于您在其中创建 Amazon Cognito 用户和用户池的区域 ID，例如 `eu-west-3`。
6. 对于 redirect-uri，请输入用户通过身份验证后要重定向到的 URI。

您可以立即部署 Gapwalk 应用程序，并使用之前创建的用户登录您的应用程序。

> **Note**
>
> 如果在登录过程中出现异常 “属性中缺少属性 'cognito: username'” 的错误，请删除以下行:: cognito: username user-name-attribute

AWS 蓝光时代运行时 API

AWS Blu Age Runtime 使用多个 Web 应用程序来公开 REST 端点，从而提供使用 REST 客户端与现代化应用程序交互的方法（例如，使用调度器调用作业）。

本文档旨在列出可用的 REST 端点，并提供以下方面的详细信息：

- 他们的角色
- 正确使用它们的方法

端点列表的分类具体取决于所提供服务的性质和公开端点的 Web 应用程序。

我们假设您已经具备以下基础知识：使用 POSTMAN、Thunder Client、CURL、Web 浏览器等专用工具使用 REST 端点的基础知识，或者编写一段代码来进行 API 调用。

主题

- 构建 URL
- Gapwalk-Application
- Blusam 应用程序控制台 REST 端点
- JICS 应用程序控制台
- 数据结构

构建 URL

以下每个 Web 应用程序均定义一个根路径，根路径可由所有端点共用。然后，每个端点会添加自己的专用路径。要使用的 URL 是通过将这些路径进行拼接得出的。例如，假设要得到 Gapwalk-Application 的第一个端点，我们使用：

- `/gapwalk-application` 作为根 Web 应用程序路径。
- `/scripts` 作为专用端点路径。

得出 URL 为 `http://server:port/gapwalk-application/scripts`

服务器

表示服务器名称（托管给定 Web 应用程序的服务器）。

port

服务器公开的端口。

Gapwalk-Application

Blusam Web 应用程序的端点使用根路径 `/gapwalk-application`。

主题

- 批处理作业（现代化 JCL 等）相关端点
- 指标端点
- 其他端点
- 作业队列相关端点

批处理作业（现代化 JCL 等）相关端点

批处理作业既可以同步运行，也可以异步运行（请参阅以下详细信息）。批处理作业是使用 groovy 脚本执行的，这些脚本是旧脚本 (JCL) 现代化的结果。

主题

- 列出已部署的脚本
- 同步启动脚本
- 异步启动脚本
- 列出触发的脚本
- 检索作业执行详细信息
- 列出可以终止的异步启动脚本
- 列出可以终止的同步启动脚本
- 终止给定的作业执行
- 列出用于实现可重启性的现有检查点
- 重新启动作业（同步）
- 重新启动作业（异步）
- 为异步作业执行设置线程限值

列出已部署的脚本

- 支持的方法：GET
- 路径：`/scripts`
- 参数：无
- 此端点以字符串形式返回服务器上已部署的 groovy 脚本列表。此端点主要从 Web 浏览器中使用，因为返回的字符串是一个 HTML 页面，其中包含活动链接（每个可启动脚本一个链接，请参阅以下示例）。

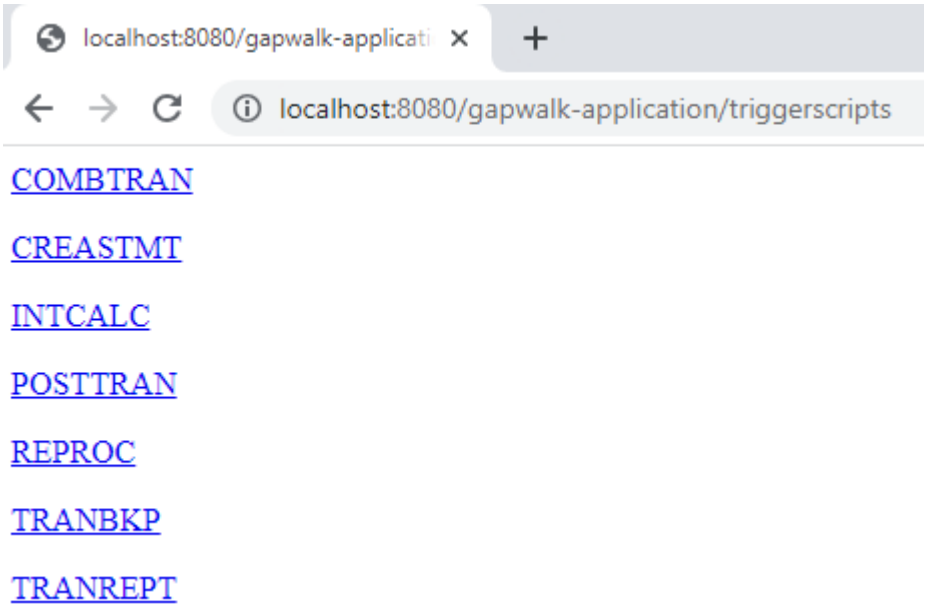
示例响应:

```
<p><a href=./script/COMBTRAN>COMBTRAN</a></p><p><a href=./script/CREASTMT>CREASTMT</
a></p><p><a href=./script/INTCALC>INTCALC</a></p><p><a href=./script/
POSTTRAN>POSTTRAN</a></p><p><a href=./script/REPROC>REPROC</a></p><p><a href=./script/
TRANBKP>TRANBKP</a></p><p><a href=./script/TRANREPT>TRANREPT</a></p><p><a href=./
script/functions>functions</a></p>
```

> **Note**
>
> 这些链接是用于同步启动每个列出的脚本的 URL。

- 支持的方法：GET
- 路径：`/triggerscripts`
- 参数：无
- 此端点以字符串形式返回服务器上已部署的 groovy 脚本列表。此端点主要从 Web 浏览器中使用，因为返回的字符串是一个 HTML 页面，其中包含活动链接（每个可启动脚本一个链接，请参阅以下示例）。

  与之前的端点响应相反，这些链接是用于异步启动每个列出的脚本的 URL。



同步启动脚本

此端点有两个变体，分别针对 GET 和 POST 方法使用专用路径（请参阅下文）。

- 支持的方法：GET
- 路径：`/script/{scriptId:.+}`
- 支持的方法：POST
- 路径：`/post/script/{scriptId:.+}`
- 参数：
  - 要启动的脚本的标识符
  - 可选：使用请求参数（显示为 `Map<String,String>`）传递给脚本的参数。给定的参数将自动添加到调用的 groovy 脚本的绑定中。
- 该调用将使用额外的参数（如果提供）启动具有给定标识符的脚本，并等待脚本执行完成，然后再返回一条消息 (String)。该消息为以下两者之一：
  - “完成。”（如果作业执行顺利）。
  - 一条 JSON 错误消息，其中详细说明了作业执行期间出现的问题。可以从服务器日志中检索更多详细信息，以了解作业执行出现的问题。

```
{
    "exitCode": -1,
    "stepName": "STEP15",
```

```
    "program": "CBACT04C",
    "status": "Error"
}
```

通过查看服务器日志，可以发现此处出现了部署问题（预期的程序未正确部署，因此无法被找到，导致作业执行失败）：

```
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.s.BatchWebController          - --> executing script INTCALC
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.s.BatchWebController          - Bound jobContext 419695287 - GDGEventsQueueHandler :907380469
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.s.ScriptControlTower          - Added jobExecutor [a65c2791-864f-43c9-972a-b5f2353389e6] to Sync Script Control Tower.
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.j.s.JobExecutor              - a65c2791-864f-43c9-972a-b5f2353389e6 - worker :Thread-26 [1547512424]
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.j.s.JobExecutor              - Triggered script: INTCALC - [a65c2791-864f-43c9-972a-b5f2353389e6] - jobContext [419695287]
2023-06-09_10-27-29-613 | [JOB] INTCALC - Started
2023-06-09_10-27-29-651 | [STEP] STEP15 - Started
2023-06-09 10:27:29 default        ERROR - c.n.b.g.r.c.i.ExecutionControllerImpl   - Could not find program "CBACT04C" in the program registry.
2023-06-09 10:27:29 default        ERROR - c.n.b.g.r.c.i.ExecutionControllerImpl   - Could not find program "CBACT04C" in the program registry.
2023-06-09_10-27-29-760 | Program not found => not executed !
2023-06-09_10-27-29-761 | [STEP] STEP15 - Ended
2023-06-09_10-27-29-772 | [JOB] INTCALC - Ended
2023-06-09 10:27:29 default        INFO  - c.n.b.g.r.j.s.DefaultJobContext         - Job [419695287] - starting final operation
2023-06-09 10:27:29 default        INFO  - c.n.b.g.r.j.s.DefaultJobContext         - End of job [419695287]
2023-06-09 10:27:29 default        INFO  - c.n.b.g.r.s.ScriptControlTower          - Removed jobExecutor [a65c2791-864f-43c9-972a-b5f2353389e6] from Script Control Tower.
2023-06-09 10:27:29 default        INFO  - c.n.b.g.r.s.ScriptControlTower          - Remaining jobExecutors:0
```

> **Note**
>
> 同步调用应用于短时间运行的作业。长时间运行的作业应采用异步启动（请参阅以下专用端点）。

异步启动脚本

- 支持的方法：GET/POST
- 路径：`/triggerscript/{scriptId:.+}`
- 参数：
  - 要启动的脚本的标识符
  - 可选：使用请求参数（显示为 `Map<String,String>`）传递给脚本的参数。给定的参数将自动添加到调用的 groovy 脚本的 https://docs.groovy-lang.org/latest/html/api/groovy/lang/Binding.html[bindings] 中。
- 与上面的同步模式相反，此端点不会等待作业执行完成后再发送响应。如果可以找到可用的线程，作业执行会立即启动，并立即向调用方发送响应。响应中包含作业执行 ID（表示作业执行的唯一标识符），该 ID 可用于查询作业执行状态或强制终止出现故障的作业执行。响应的格式为：

```
Triggered script <script identifier> [unique job execution id] @ <date and time>
```

- 由于作业异步执行依赖于固定、有限数量的线程，因此如果找不到可用的线程，则可能无法启动作业执行。在这种情况下，返回的消息将如下所示：

```
Script [<script identifier>] NOT triggered - Thread limit reached (<actual thread
 limit>) - Please retry later or increase thread limit.
```

请参阅下文的 settriggerthreadlimit 端点，了解如何提高线程限制。

示例响应:

```
Triggered script INTCALC [d43cbf46-4255-4ce2-aac2-79137573a8b4] @ 06-12-2023 16:26:15
```

借助唯一作业执行标识符，您可以根据需要快速检索服务器日志中的相关日志条目。其他几个端点也会使用该标识符，详细信息如下文所述。

列出触发的脚本

- 支持的方法：GET
- 路径：/triggeredscripts/{status:.+}、/triggeredscripts/{status:.+}/{namefilter}
- 参数：
  - Status（必需）：要检索的已触发脚本的状态。可能的值为：
    - all：显示所有作业执行详细信息，无论作业是否仍在运行。
    - running：仅显示当前正在运行的作业的作业详细信息。
    - done：仅显示已结束执行的作业的作业详细信息。
    - killed：仅显示已使用专用端点强制终止执行的作业的作业详细信息（请参阅下文）。
    - triggered：仅显示已触发但尚未启动的作业的作业详细信息。
    - failed：仅显示已标记为执行失败的作业的作业详细信息。
    - _namefilter（可选）_：仅检索给定脚本标识符的执行。
- 以 JSON 形式返回作业执行详细信息的集合。有关更多信息，请参阅作业执行详细信息消息结构。

示例响应:

```
[
    {
      "scriptId": "INTCALC",
      "caller": "127.0.0.1",
```

```
      "identifier": "d43cbf46-4255-4ce2-aac2-79137573a8b4",
      "startTime": "06-12-2023 16:26:15",
      "endTime": "06-12-2023 16:26:15",
      "status": "DONE",
      "executionResult": "{ \"exitCode\": -1, \"stepName\": \"STEP15\", \"program\":
\"CBACT04C\", \"status\": \"Error\" }",
      "executionMode": "ASYNCHRONOUS"
    }
 ]
```

检索作业执行详细信息

- 支持的方法：GET
- 路径：`/getjobexecutioninfo/{jobexecutionid:.+}`
- 参数：
  - jobexecutionid（必需）：用于检索相应作业执行详细信息的唯一作业执行标识符。
- 返回：表示单个作业执行详细信息的 JSON 字符串（请参阅 作业执行详细信息消息结构），如果根据给定标识符找不到相应的作业执行详细信息，则响应为空。

列出可以终止的异步启动脚本

- 支持的方法：GET
- 路径：`/killablescripts`
- 返回一组异步启动作业的作业执行标识符，这些作业当前仍在运行并且可以被强制终止（请参阅下文的 `/kill` 端点）。

列出可以终止的同步启动脚本

- 支持的方法：GET
- 路径：`/killablesyncscripts`
- 返回一组同步启动作业的作业执行标识符，这些作业当前仍在运行并且可以被强制终止（请参阅下文的 `/kill` 端点）。

终止给定的作业执行

- 支持的方法：GET

- 路径：`/kill/{identifier:.+}`
- 参数：作业执行标识符（必需）：要强制终止的作业执行的唯一作业执行标识符。
- 返回：一条详细描述作业执行终止尝试结果的文本消息；该消息将包含脚本标识符、作业执行唯一标识符以及执行终止发生的日期和时间。如果找不到具有给定标识符且正在运行的作业执行，则将返回一条错误消息。

> **Warning**
>
> - 运行时会尽最大努力合理地终止目标作业执行。因此，来自 /kill 端点的响应可能需要一点时间才能到达调用方，因为 AWS Blu Age 运行时会尽量减少终止任务对业务的影响。
> - 强制终止作业执行应引起注意，因为它可能会产生直接的业务后果，包括可能的数据丢失或损坏。强制终止应该用于给定作业执行出现意外情况且已明确确定数据补救措施的情况。
> - 终止作业后，应进行进一步的调查（事后分析），以找出问题所在，并采取适当的补救措施。
> - 在任何情况下，终止正在运行的作业的尝试都会记录在服务器日志中，并提供警告级别的消息。

列出用于实现可重启性的现有检查点

作业可重启性依赖于脚本能否在 `CheckpointRegistry` 中注册检查点来跟踪作业执行进度。如果作业执行未能正常结束，并且已经注册了重新启动检查点，则只需从最后一个已知的已注册检查点重新启动作业执行即可（无需执行检查点上方的步骤）。

- 支持的方法：GET
- 路径：`/restarts`
- 以 html 页面的形式返回现有重新启动检查点的列表，这些重新启动检查点可用于重新启动未正确执行和结束的作业。如果任何脚本都未注册任何检查点，则页面内容显示“无注册检查点。”

重新启动作业（同步）

- 支持的方法：GET
- 路径：`/restart/{hashcode}`
- 参数：hashcode（整数 – 必需）：使用提供的哈希码作为检查点值（请参阅上文的 `/restarts` 端点，以了解如何检索有效的检查点值），重新启动先前中止的作业执行。

- 返回：请参阅上文的 `script` 返回内容描述。

重新启动作业（异步）

- 支持的方法：GET
- 路径：`/triggerrestart/{hashcode}`
- 参数：hashcode（整数 – 必需）：使用提供的哈希码作为检查点值（请参阅上文的 `/restarts` 端点，以了解如何检索有效的检查点值），重新启动先前中止的作业执行。
- 返回：请参阅上文的 `triggerscript` 返回内容描述。

为异步作业执行设置线程限值

作业异步执行依赖于 JVM 中的专用线程池。该池对可用线程的数量有固定的限值。用户能够根据主机容量（CPU 数量、可用内存等...）调整该限值。默认情况下，线程限值设置为 5 个线程。

- 支持的方法：GET
- 路径：`/settriggerthreadlimit/{threadlimit:.+}`
- 参数（整数）：要应用的新线程限值。其值必须为严格的正整数。
- 返回一条消息 (String)，其中包括新的线程限制值和前一个线程限值；如果提供的线程限值无效（不是严格的正整数），则返回错误消息。

示例响应:

```
Set thread limit for Script Tower Control to 10 (previous value was 5)
```

计算当前正在运行的已触发任务执行个数

- 支持的方法：GET
- 路径：`/countrunningtriggeredscripts`
- 返回一条消息，指示正在运行的异步启动作业数和线程限值（即可以同时运行的最大已触发作业数）。

示例响应:

```
0 triggered script(s) running (limit =10)
```

> **Note**
>
> 此端点可用于在启动作业之前检查是否未达到线程限值（达到该值时，将阻止作业启动）。

清除作业执行信息

只要服务器启动，作业执行信息就会保留在服务器内存中。此端点可便于清除内存中较旧的信息，因为这些信息已经不适用。

- 支持的方法：GET
- 路径：`/purgejobinformation/{age:.+}`
- 参数：一个严格的正整数值，表示要清除信息的留存时间（以小时为单位）。
- 返回一条包含以下信息的消息：
  - 清除文件的名称，该文件存储已清除的作业执行信息以供存档。
  - 已清除的作业执行信息的数量。
  - 内存中剩余的作业执行信息数量。

指标端点

JVM

此端点返回与 JVM 相关的可用指标。

- 支持的方法：GET
- 路径：`/metrics/jvm`
- 参数：无
- 返回一条包含以下信息的消息：
  - threadActiveCount：活动线程数。
  - jvmMemoryUsed：Java 虚拟机正在使用的内存。
  - jvmMemoryMax：Java 虚拟机允许的最大内存。
  - jvmMemoryFree：Java 虚拟机当前未使用的可用内存。

会话

此端点返回与当前打开的 HTTP 会话相关的指标。

- 支持的方法：GET
- 路径：/metrics/session
- 参数：无
- 返回一条包含以下信息的消息：
  - sessionCount：服务器当前维护的活跃用户会话数。

批处理

- 支持的方法：GET
- 路径：/metrics/batch
- 参数：
  - startTimestamp（可选，数字）：用于数据筛选的起始时间戳。
  - endTimestamp（可选，数字）：用于数据筛选的结束时间戳。
  - page（可选，数字）：用于分页的页码。
  - pageSize（可选，数字）：分页中每页的项目数。
- 返回一条包含以下信息的消息：
  - content：批量执行指标的列表。
  - pageNumber：分页中的当前页码。
  - pageSize：每页显示的项目数。
  - totalPages：可用的总页数。
  - numberOfElements：当前页面上的项目数。
  - last：最后一页的布尔标志。
  - first：第一页的布尔标志。

事务

- 支持的方法：GET
- 路径：/metrics/transaction
- 参数：
  - startTimestamp（可选，数字）：用于数据筛选的起始时间戳。
  - endTimestamp（可选，数字）：用于数据筛选的结束时间戳。
  - page（可选，数字）：用于分页的页码。

  - pageSize（可选，数字）：分页中每页的项目数。
- 返回一条包含以下信息的消息：
  - content：批量执行指标的列表。
  - pageNumber：分页中的当前页码。
  - pageSize：每页显示的项目数。
  - totalPages：可用的总页数。
  - numberOfElements：当前页面上的项目数。
  - last：最后一页的布尔标志。
  - first：第一页的布尔标志。

其他端点

使用以下端点列出已注册的程序或服务、了解运行状况和管理 JICS 事务。

主题

- 列出已注册程序
- 列出已注册服务
- 运行状况
- 列出可用的 JICS 事务
- 启动 JICS 事务
- 启动 JICS 事务（备用）

列出已注册程序

- 支持的方法：GET
- 路径：`/programs`
- 以 html 页面的形式返回已注册程序的列表。每个程序都由其主程序标识符指定。返回的列表中包含现代化的传统程序和实用程序（IDCAMS、IEBGENER等...）。请注意，可用的实用程序将取决于您的 tomcat 服务器上部署的实用程序 Web 应用程序。例如，z/OS 实用程序支持程序可能不适用于现代化的 iSeries 资产，因为它们不相关。

列出已注册服务

- 支持的方法：GET
- 路径：`/services`
- 以 html 页面的形式返回已注册运行时服务的列表。给定的服务由 AWS Blu Age 运行时作为实用程序提供，例如，可以在 groovy 脚本中使用。Blusam 加载服务（从传统数据集创建 Blusam 数据集）属于此类服务。

示例响应:

```
<p>BluesamESDSFileLoader</p><p>BluesamKSDSFileLoader</p><p>BluesamRRDSFileLoader</p>
```

运行状况

- 支持的方法：GET
- 路径：`/`
- 返回一条简单消息，表明 gapwalk-application 已启动并正在运行 (`Jics application is running.`)

列出可用的 JICS 事务

- 支持的方法：GET
- 路径：`/transactions`
- 返回一个 html 页面，其中列出了所有可用的 JICS 事务。这仅适用于具有 JICS 元素（对传统 CICS 元素进行现代化）的环境。

示例响应:

```
<p>INQ1</p><p>MENU</p><p>MNT2</p><p>ORD1</p><p>PRNT</p>
```

启动 JICS 事务

- 支持的方法：GET、POST
- 路径：`/jicstransrunner/{jtrans:.+}`
- 参数：

- JICS 事务标识符（字符串，必需）：要启动的 JICS 事务的标识符（最多 8 个字符）
- 必需：以 Map<String,Object> 形式传递给事务的其他输入数据。该映射的内容用于提供由 JICS 事务使用的 COMMAREA。如果运行事务不需要任何数据，则该映射可以为空。
- 可选：Http 标头条目，用于自定义给定事务的运行环境。支持以下标头键：
  - `jics-channel`：将在本事务启动时启动的程序要使用的 JICS CHANNEL 的名称。
  - `jics-container`：用于启动此 JICS 事务的 JICS 容器的名称。
  - `jics-startcode`：在 JICS 事务启动时使用的 STARTCODE（字符串，最多 2 个字符）。有关可能的值，请参阅 STARTCODE（向下浏览页面）。
  - `jicxa-xid`：由调用方发起的“全局事务”(XA) 的 XID（X/Open 事务标识符 XID 结构），当前的 JICS 事务启动将参与该全局事务。
- 返回：表示 JICS 事务启动结果的 `com.netfective.bluage.gapwalk.rt.shared.web.TransactionResultBean` JSON 序列化。

有关该结构的更多详细信息，请参阅事务启动结果结构。

启动 JICS 事务（备用）

- 支持的方法：GET、POST
- 路径：`/jicstransaction/{jtrans:.+}`
- 参数：

  JICS 事务标识符（字符串，必需）

  要启动的 JICS 事务的标识符（长度为最多 8 个字符）

  必需：以 Map<String,Object> 形式传递给事务的其他输入数据

  该映射的内容用于提供由 JICS 事务使用的 COMMAREA。如果运行事务不需要任何数据，则该映射可以为空。

  可选：Http 标头条目，用于自定义给定事务的运行环境。

  支持以下标头键：
  - `jics-channel`：将在本事务启动时启动的程序要使用的 JICS CHANNEL 的名称。
  - `jics-container`：用于启动此 JICS 事务的 JICS 容器的名称。
  - `jics-startcode`：在 JICS 事务启动时使用的 STARTCODE（字符串，最多 2 个字符）。有关可能的值，请参阅 STARTCODE（向下浏览页面）。

  - `jicxa-xid`：由调用方发起的“全局事务”(XA) 的 XID（X/Open 事务标识符 XID 结构），当前的 JICS 事务启动将参与该全局事务。
- 返回：表示 JICS 事务启动结果的 `com.netfective.bluage.gapwalk.rt.shared.web.RecordHolderBean` JSON 序列化。有关该结构的详细信息，请参阅事务启动记录结果结构。

作业队列相关端点

Job Queues 是 AWS Blu Age 对 AS400 作业提交机制的支持。在 AS400 中，作业队列用于在特定的线程池上运行作业。作业队列由名称和最大线程数定义，最大线程数对应于该队列上可以同时运行的最大程序数。如果队列中提交的作业大于最大线程数，则作业将等待线程可用。

有关队列中作业状态的详尽列表，请参阅队列中作业的可能状态。

作业队列上的操作通过以下专用端点处理。您可以使用以下根网址从Gapwalk应用程序网址调用这些操作:`http://server:port/gapwalk-application/jobqueue`.

主题

- 列出可用的队列
- 启动或重新启动作业队列
- 提交要启动的作业
- 列出所有计划作业
- 列出所有“暂停”作业
- 列出所有活动作业
- 列出所有等待启动的作业
- 释放所有“暂停”作业
- 释放给定作业名称下所有处于“暂停”状态的作业
- 释放具有给定作业编号的作业

列出可用的队列

- 支持的方法：GET
- 路径：`list-queues`
- 以 JSON 键值列表的形式返回可用队列列表及其状态。

示例响应:

```
{"Default":"STAND_BY","queue1":"STARTED","queue2":"STARTED"}
```

作业队列的可能状态为：

STAND_BY

作业队列正在等待启动。

STARTED

作业队列已启动并正在运行。

UNKNOWN

无法确定作业队列状态。

启动或重新启动作业队列

- 支持的方法：POST
- 路径：`/restart/{name}`
- 参数：要启动/重新启动的队列的名称（字符串 – 必需）。
- 端点不返回任何内容，而是依赖 http 状态来指示启动/重启操作的结果：

  HTTP 200

  启动/重启操作正常：给定的作业队列现在已启动。

  HTTP 404

  作业队列不存在。

  HTTP 503

  尝试启动/重新启动期间出现异常（应检查服务器日志以找出问题所在）。

提交要启动的作业

- 支持的方法：POST
- 参数：（必需，请求体）`com.netfective.bluage.gapwalk.rt.jobqueue.SubmitJobMessage` 对象的 JSON 序列化。有关更多信息，请参阅提交作业输入。

- 返回：一个包含原始 `SubmitJobMessage` 的 JSON 和一个指示任务是否已提交的日志。

列出所有计划作业

- 支持的方法：GET
- 路径：`list-jobs`
- 返回：所有计划作业的列表，采用 JSON 字符串形式。有关响应示例，请参阅计划作业响应列表。

列出所有“暂停”作业

- 支持的方法：GET
- 路径：`list-jobs-hold`
- 返回：所有计划作业的列表，采用 JSON 字符串形式。有关响应示例，请参阅“暂停”作业响应列表。

列出所有活动作业

- 支持的方法：GET
- 路径：`list-jobs-active`
- 返回：状态为 ACTIVE 的所有作业的列表，采用 JSON 字符串形式。该响应的结构与计划作业响应列表中的响应结构类似。

列出所有等待启动的作业

- 支持的方法：GET
- 路径：`list-jobs-waiting`
- 返回：状态为 EXECUTION_WAIT 的所有作业的列表（正在等待线程可用于启动的作业），采用 JSON 字符串形式。该响应的结构与 the section called “计划作业响应列表” 中的类似。

释放所有“暂停”作业

- 支持的方法：POST
- 路径：`release-all`
- 返回：一条消息，指示释放尝试操作的结果。这里有两种可能的情况：
  - HTTP 200 和消息“所有作业均已成功释放！”（如果所有作业均已成功释放）。

- HTTP 503 和消息“作业未释放。出现未知错误。有关更多详细信息，请参阅日志”（如果释放尝试出现问题）。

释放给定作业名称下所有处于“暂停”状态的作业

对于给定的作业名称，可以提交多个具有不同作业编号的作业（作业运行的唯一性由一组 <job name, job number> 来保证）。此端点将尝试释放具有给定作业名称的所有处于“暂停”状态的作业提交。

- 支持的方法：POST
- 路径：`/release/{name}`
- 参数：要查找的作业名称，采用字符串形式。必需。
- 返回：一条消息，指示释放尝试操作的结果。这里有两种可能的情况：
  - HTTP 200 和消息 “组 <name> (<number of released jobs>) 中的作业已成功释放！”（如果作业已成功释放）。
  - HTTP 503 和消息“组 <name> 中的作业未释放。出现未知错误。有关更多详细信息，请参阅日志”（如果释放尝试出现问题）。

释放具有给定作业编号的作业

此端点将尝试释放具有给定 <job name, job number> 且处于“暂停”状态的唯一作业提交。

- 支持的方法：POST
- 路径：`/release/{name}/{number}`
- 参数：

  name

  参数：要查找的作业名称，采用字符串形式。必需。

  number

  要查找的作业编号，以整数表示。必需。

  返回

  一条消息，指示释放尝试操作的结果。这里有两种可能的情况：
  - HTTP 200 和消息“作业 <name/number> 已成功释放！”（如果该作业已成功释放）。
  - HTTP 503 和消息“作业 <name/number> 未释放。出现未知错误。有关更多详细信息，请参阅日志”（如果释放尝试出现问题）。

Blusam 应用程序控制台 REST 端点

Blusam 应用程序控制台是用于简化现代化 VSAM 数据集管理的 API。Blusam Web 应用程序的端点使用根路径 `/bac`。

主题

数据集相关端点

使用以下端点创建或管理特定的数据集。

主题

创建数据集

创建数据集端点允许创建数据集定义，并且需要身份验证。

- 支持的方法：POST
- 路径：`` `/api/services/rest/bluesamservice/createDataSet` ``
- 参数：

name

（必需，字符串）：数据集的名称。

type

（必需，字符串）：数据集类型。可能的值为：ESDS、KSDS、RRDS。

recordSize

（可选，字符串）：数据集每条记录的最大大小。

fixedLength

（可选，布尔值）：表示记录长度是否固定。

compression

（可选，布尔值）：表示数据集是否压缩。

cacheEnable

（可选，布尔值）：表示是否为数据集启用缓存。

alternativeKeys

（可选，键列表）：

- offset（必需，数字）
- length（必需，数字）
- name（必需，数字）

- 返回一个代表新创建数据集的 json 文件。

示例请求：

```
POST /api/services/rest/bluesamservice/createDataSet
{
  "name": "DATASET",
  "checked": false,
  "records": [],
  "primaryKey": {
    "name": "PK"
  },
  "alternativeKeys": [
    {
```

```
      "offset": 10,
      "length": 10,
      "name": "ALTK_0"
    }
  ],
  "type": "ESDS",
  "recordSize": 10,
  "compression": true,
  "cacheEnable": true
}
```

示例响应:

```
{
    "dataSet": {
      "name": "DATASET",
      "checked": false,
      "nbRecords": 0,
      "keyLength": -1,
      "recordSize": 10,
      "compression": false,
      "fixLength": true,
      "type": "ESDS",
      "cacheEnable": false,
      "cacheWarmup": false,
      "cacheEviction": "100ms",
      "creationDate": 1686744961234,
      "modificationDate": 1686744961234,
      "records": [],
      "primaryKey": {
        "name": "PK",
        "offset": null,
        "length": null,
        "columns": null,
        "unique": true
      },
      "alternativeKeys": [
        {
          "offset": 10,
          "length": 10,
          "name": "ALTK_0"
        }
      ],
```

```
      "readLimit": 0,
      "readEncoding": null,
      "initCharacter": null,
      "defaultCharacter": null,
      "blankCharacter": null,
      "strictZoned": null,
      "decimalSeparator": null,
      "currencySign": null,
      "pictureCurrencySign": null
    },
    "message": null,
    "result": true
  }
```

上传文件

此端点允许将文件上传到服务器。文件存储在与每个特定用户相对应的临时文件夹中。每次需要上传文件时均应使用此端点。此端点需要身份验证。

- 支持的方法：POST
- 路径：/api/services/rest/bluesamservice/upload
- 参数：

  file

  （必需，多部分/形式数据）：要上传的文件。
- 返回表示上传状态的布尔值

加载数据集

使用前述创建数据集端点创建数据集定义后，您可以将与上传文件关联的记录加载到特定的数据集中。此端点需要身份验证。

- 支持的方法：POST
- 路径：/api/services/rest/bluesamservice/loadDataSet
- 参数：

  name

  （必需，字符串）：数据集的名称。
- 返回请求的状态和已加载的数据集。

从 Amazon S3 存储桶加载数据集

使用 Amazon S3 存储桶中的 listcat 文件加载数据集。

- 支持的方法：GET
- 路径：`/api/services/rest/bluesamservice/loadDataSetFromS3`
- 参数：

  listcatFileS3Location

  （必需，字符串）：listcat 文件在 Amazon S3 中的位置。

  datasetFileS3Location

  （必需，字符串）：数据集文件在 Amazon S3 中的位置。

  region

  （必填，字符串）：存储文件的 Amazon S3 AWS 区域 。
- 返回新创建的数据集

示例请求：

```
/BAC/api/services/rest/bluesamservice/loadDataSetFromS3?region=us-
east-1&listcatFileS3Location=s3://bucket-name/listcat.json&datasetFileS3Location=s3://
bucket-name/dataset.DAT
```

将数据集导出到 Amazon S3 存储桶

将数据集导出到指定的 Amazon S3 存储桶。

- 支持的方法：GET
- 路径：/api/services/rest/bluesamservice/exportDataSetToS3
- 参数：

  s3Location

  （必需，字符串）：将数据集文件导出到 Amazon S3 中的位置。

  datasetName

  （必需，字符串）：要导出的数据集的名称。

region

（必填，字符串）：Amazon S3 存储桶的。 AWS 区域

- 返回导出的数据集

示例请求：

```
/BAC/api/services/rest/bluesamservice/exportDataSetToS3?region=eu-
west-1&s3Location=s3://bucket-name/dump&datasetName=dataset
```

清除数据集

此端点用于清除数据集中的所有记录，需要身份验证。

- 支持的方法：POST
- 路径：`/api/services/rest/bluesamservice/clearDataSet`
- 参数：

  name

  （必需，字符串）：要清除的数据集的名称。
- 返回请求的状态。

删除数据集

删除数据集定义和记录。此端点需要身份验证。

- 支持的方法：POST
- 路径：`/api/services/rest/bluesamservice/deleteDataSet`
- 参数：

  name

  （必需，字符串）：要删除的数据集的名称。
- 返回请求的状态和已删除的数据集。

统计数据集记录数

此端点返回与数据集相关的记录数，需要身份验证。

- 支持的方法：POST
- 路径：`/api/services/rest/bluesamservice/countRecords`
- 参数：

  name

  （必需，字符串）：数据集的名称。
- 返回：记录数

批量处理数据集相关端点

使用以下端点同时创建或管理多个数据集。

主题

- 导出数据集
- 创建多个数据集
- 列出所有数据集
- 直接列出所有数据集
- 删除所有数据集
- 从 listcat 文件中获取数据集定义
- 从上传的 listcat 文件中获取数据集定义
- 从 json 文件中加载 listcat

导出数据集

- 支持的方法：GET
- 路径：`/api/services/rest/bluesamservice/exportDataSet`
- 参数：

  name

  （必需，字符串）：要删除的数据集的名称。

  datasetOutputFile

  （可选，字符串）：服务器上存储导出数据集的路径

rdw

（可选，布尔值）：是否导出 RDW 字段。

- 返回请求的状态和包含导出数据集的文件。

创建多个数据集

- 支持的方法：POST
- 路径：/api/services/rest/bluesamservice/createAllDataSets
- 参数：
  - 数据集列表

    name

    （必需，字符串）：数据集的名称。

    type

    （必需，字符串）：数据集类型。可能的值为：ESDS、KSDS、RRDS。

    recordSize

    （可选，字符串）：数据集每条记录的最大大小。

    fixedLength

    （可选，布尔值）：表示记录长度是否固定。

    compression

    （可选，布尔值）：表示数据集是否压缩。

    cacheEnable

    （可选，布尔值）：表示是否为数据集启用缓存。
- 返回：请求的状态和新创建的数据集。

列出所有数据集

- 支持的方法：GET
- 路径：/api/services/rest/bluesamservice/listDataSet
- 参数：无

- 返回：请求的状态和数据集列表。

直接列出所有数据集

- 支持的方法：GET
- 路径：`/api/services/rest/bluesamservice/directListDataSet`
- 参数：无
- 返回：请求的状态和数据集列表。

删除所有数据集

- 支持的方法：POST
- 路径：`/api/services/rest/bluesamservice/removeAll`
- 参数：无
- 返回：表示请求状态的布尔值。

从 listcat 文件中获取数据集定义

- 支持的方法：POST
- 路径：`/api/services/rest/bluesamservice/getDataSetsDefinitionFromListcat`
- 参数：

  paramFilePath

  （必需，字符串）：listcat 文件的路径。
- 返回：数据集列表

从上传的 listcat 文件中获取数据集定义

- 支持的方法：POST
- 路径：`` `/api/services/rest/bluesamservice/getDataSetsDefinitionFromUploadedListcat` ``
- 参数：无
- 返回：数据集列表

从 json 文件中加载 listcat

- 支持的方法：GET
- 路径：`/api/services/rest/bluesamservice/loadListcatFromJsonFile`
- 参数：

  filePath

  （必需，字符串）：listcat 文件的路径。
- 返回：数据集列表

记录

使用以下端点在数据集中创建或管理记录。

主题

- 创建记录
- 读取数据集
- 删除记录
- 更新记录
- 保存记录

创建记录

此端点用于创建新记录，需要身份验证。

- 支持的方法：POST
- 路径：`/api/services/rest/crud/createRecord`
- 参数：

  数据集

  （必填， DataSet）：数据集对象

  mask

  （必需，掩码）：掩码对象。
- 返回：请求的状态和创建的记录。

读取数据集

此端点用于读取数据集。

- 支持的方法：GET
- 路径：/api/services/rest/crud/readDataSet
- 参数：

  数据集

  （必填，DataSet）：数据集对象。
- 返回请求的状态和带有记录的数据集。

删除记录

此端点用于从数据集中删除记录，需要身份验证。

- 支持的方法：DELETE
- 路径：/api/services/rest/crud/deleteRecord
- 参数：

  数据集

  （必填，DataSet）：数据集对象

  记录

  （必需，记录）：要删除的记录
- 返回删除状态。

更新记录

此端点用于更新与数据集相关的记录，需要身份验证。

- 支持的方法：POST
- 路径：/api/services/rest/crud/updateRecord
- 参数：

  数据集

  （必填，DataSet）：数据集对象

记录

（必需，记录）：要更新的记录

- 返回请求的状态和带有记录的数据集。

保存记录

此端点用于将记录保存到数据集中并使用掩码，需要身份验证。

- 支持的方法：POST
- 路径：`/api/services/rest/crud/saveRecord`
- 参数：

  数据集

  （必填， DataSet）：数据集对象

  记录

  （必需，记录）：要保存的记录
- 返回请求的状态和带有记录的数据集。

掩码

使用以下端点对数据集加载或应用掩码。

主题

- 加载掩码
- 应用掩码
- 应用掩码过滤器

加载掩码

此端点用于检索与特定数据集相关的所有掩码，需要身份验证。

- 支持的方法：POST
- 路径：`/api/services/rest/crud/loadMasks`
- 参数：

数据集

（必填，DataSet）：数据集对象

- 返回请求的状态和掩码列表。

应用掩码

此端点用于对特定数据集应用掩码，需要身份验证。

- 支持的方法：POST
- 路径：`/api/services/rest/crud/applyMask`
- 参数：

  数据集

  （必填，DataSet）：数据集对象

  mask

  （必需，掩码）：数据集对象
- 返回请求的状态和已应用掩码的数据集。

应用掩码过滤器

此端点用于对特定数据集应用掩码和过滤器，需要身份验证。

- 支持的方法：POST
- 路径：`/api/services/rest/crud/applyMaskFilter`
- 参数：

  数据集

  （必填，DataSet）：数据集对象

  mask

  （必需，掩码）：数据集对象
- 返回请求的状态以及已应用掩码和过滤器的数据集。

其他

使用以下端点管理数据集的缓存或检查数据集的特征

主题

检查预热缓存

检查是否已为特定数据集启用预热缓存。此端点需要身份验证。

- 支持的方法：POST
- 路径：`/api/services/rest/bluesamservice/warmupCache`
- 参数：

  name

  （必需，字符串）：数据集的名称。
- 返回：如果已启用预热缓存，则返回 true，否则返回 false。

检查是否已启用缓存

检查是否已为特定数据集启用缓存。此端点需要身份验证。

- 支持的方法：GET
- 路径：/api/services/rest/bluesamservice/isEnableCache
- 参数：无
- 如果已启用缓存，则返回 true。

启用缓存

- 支持的方法：GET
- 路径：/api/services/rest/bluesamservice/enableDisableCache/{enable}
- 参数：

  enable

  （必需，布尔值）：如果设置为 true，将启用缓存。
- 不返回任何内容

检查已分配的 RAM 缓存

此端点允许检索已分配的 RAM 缓存，需要身份验证。

- 支持的方法：GET
- 路径：/api/services/rest/bluesamservice/allocatedRamCache
- 参数：无
- 返回：表示内存大小的字符串

检查持久性

- 支持的方法：GET
- 路径：/api/services/rest/bluesamservice/persistence
- 参数：无
- 返回：表示使用的持久性，以字符串形式表示。

检查支持的数据集类型

- 路径：/api/services/rest/bluesamservice/getDataSetTypes
- 参数：无
- 返回：支持的数据集类型列表，以字符串列表形式显示。

检查服务器运行状况

- 支持的方法：GET

- 路径：/api/services/rest/bluesamservice/serverIsUp
- 参数：无
- 返回：无

用户

使用以下端点管理用户互动。

主题

登录

- 支持的方法：POST
- 路径：/api/services/security/servicelogin/login
- 参数：

  username

  （必需，字符串）

  password

  （必需，字符串）
- 返回登录用户的用户名和角色

示例响应

```
{"login":"some-user","roles":[{"id":0,"roleName":"ROLE_ADMIN"}]}
```

检查用户账户

- 支持的方法：POST

- 路径：/api/services/security/servicelogin/hasAccount
- 参数：无
- 返回：如果用户已经登录，则返回 true

首次登录

- 支持的方法：POST
- 路径：/api/services/security/servicelogin/recorduser
- 参数：无
- 返回：如果用户已经登录，则返回 true

列出所有用户

- 支持的方法：GET
- 路径：/api/services/security/servicelogin/listusers
- 参数：无
- 返回：所有用户的列表

注销

- 支持的方法：POST
- 路径：/api/services/security/servicelogout/logout
- 参数：无
- 返回：如果用户已成功注销，则返回 true。

JICS 应用程序控制台

JICS 组件是 AWS Blu Age 对传统 CICS 资源现代化的支持。JICS 应用程序控制台 Web 应用程序专门用于管理 JICS 资源。借助以下端点，无需与 JAC 用户界面交互即可执行管理任务。端点需要身份验证时，请求须包含身份验证详细信息（根据基本身份验证要求，通常为用户名/密码）。JICS 应用程序控制台 Web 应用程序的端点使用根路径/jac/。

主题

- JICS 资源管理

- JAC 用户管理端点

JICS 资源管理

以下所有端点都与 JICS 资源管理有关，可供 JICS 管理员用于每天处理资源。

主题

- 列出 JICS LIST 和 GROUP
- 列出 JICS GROUP
- 列出给定 LIST 的 JICS 组
- 列出给定 GROUP 的 JICS 资源
- 列出给定 GROUP（也可以使用名称）的 JICS 资源
- 编辑多个 LIST 中的 GROUP
- 删除 LIST
- 删除 GROUP
- 删除 TRANSACTION
- 删除 PROGRAM
- 删除 FILE
- 删除 TDQUEUE
- 删除 TSMODEL
- 创建 LIST
- 创建 GROUP
- 创建 RESOURCE 的常见注意事项
- 创建 TRANSACTION
- 创建 PROGRAM
- 创建 FILE
- 创建 TDQUEUE
- 创建 TSMODEL
- 更新 LIST
- 更新 GROUP

- RESOURCE 更新常见注意事项
- 更新 TRANSACTION
- 更新 PROGRAM
- 更新 FILE
- 更新 TDQUEUE
- 更新 TSMODEL

列出 JICS LIST 和 GROUP

LIST 和 GROUP 是 JICS 组件中拥有的主要容器资源。所有 JICS 资源都必须属于一个 GROUP。GROUP 可以属于 LIST，但这不是强制性的。给定的 JICS 环境中甚至可能不存在 LIST，但大多数时候，LIST 可为资源提供额外的组织层。有关 CICS 资源组织的更多信息，请参阅 CICS 资源。

- 支持的方法：GET
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/listJicsListsAndGroups`
- 返回：以 JSON 格式返回序列化 JicsContainer 对象的列表，包括列表和组。

示例响应:

```
[
    {
      "name": "Resources",
      "children": [
        {
          "jacType": "JACList",
          "name": "MURACHS",
          "isActive": true,
          "children": [
            {
              "jacType": "JACGroup",
              "name": "MURACHS",
              "isActive": true,
              "children": []
            }
          ]
        },
```

```
        {
          "jacType": "JACGroup",
          "name": "TEST",
          "isActive": true,
          "children": []
        }
      ],
      "isExpanded": true
    }
  ]
```

列出 JICS GROUP

- 支持的方法：GET
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/listJicsGroups`
- 返回：以 JSON 格式返回序列化 JicsContainer 对象（组）列表。这些 GROUP 被返回时不带有组自身的 LIST 信息。

示例响应:

```
[
    {
      "jacType": "JACGroup",
      "name": "MURACHS",
      "isActive": true,
      "children": []
    },
    {
      "jacType": "JACGroup",
      "name": "TEST",
      "isActive": true,
      "children": []
    }
  ]
```

列出给定 LIST 的 JICS 组

- 支持的方法：POST
- 需要身份验证

- 路径：/api/services/rest/jicsservice/listGroupsForList
- 参数：JSON 有效负载，表示要查找的 GROUP 所属的 JICS LIST。这是 com.netfective.bluage.jac.entities.JACList 对象的 JSON 序列化。

  示例请求：

```
{
    "jacType":"JACList",
    "name":"MURACHS",
    "isActive":true
  }
```

- 返回：作为 JSON 的序列化 JicsContainer 对象（组）列表，这些对象附加到给定列表中。这些 GROUP 被返回时不带有组自身的 LIST 信息。

  示例响应:

```
[
    {
      "jacType": "JACGroup",
      "name": "MURACHS",
      "isActive": true,
      "children": []
    }
  ]
```

列出给定 GROUP 的 JICS 资源

- 支持的方法：POST
- 需要身份验证
- 路径：/api/services/rest/jicsservice/listResourcesForGroup
- 参数：JSON 有效负载，表示要查找的资源所在的 JICS GROUP。这是 com.netfective.bluage.jac.entities.JACGroup 对象的 JSON 序列化。您无需指定 GROUP 的所有字段，但名称是必需的。

  示例请求：

```
{
    "jacType":"JACGroup",
```

```
    "name":"MURACHS",
    "isActive":true
  }
```

- 返回：由给定 GROUP 拥有的序列化 JicsResource 对象的列表。这些对象不按特定顺序返回，并且类型不同（程序、事务、文件等...）。

列出给定 GROUP（也可以使用名称）的 JICS 资源

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/listResourcesForGroupName`
- 参数：要查找的资源所在 GROUP 的名称。
- 返回：由给定 GROUP 拥有的序列化 JicsResource 对象的列表。这些对象不按特定顺序返回，并且类型不同（程序、事务、文件等...）。

编辑多个 LIST 中的 GROUP

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/editGroupsList`
- 参数：带有子 GROUP 的 LIST 集合，以 JSON 表示形式；

  示例请求：

```
[
  {
        "jacType": "JACList",
        "name": "MURACHS",
        "isActive": true,
        "children": [
          {
            "jacType": "JACGroup",
            "name": "MURACHS",
            "isActive": true,
            "children": []
          },
          {
            "jacType": "JACGroup",
```

```
            "name": "TEST",
            "isActive": true,
            "children": []
          }
        ]
  }
]
```

在进行编辑之前，仅名为“MURACHS”的 GROUP 属于名为“MURACHS”的 LIST。通过编辑，我们将名为“TEST”的 GROUP“添加” 到名为“MURACHS”的 LIST 中。

- 返回一个布尔值。如果值为 “true”，则该 LIST 修改已正确保存到底层 JICS 存储中。

删除 LIST

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/deleteList`
- 参数：JSON 有效负载，表示要删除的 JICS LIST。这是 `com.netfective.bluage.jac.entities.JACList` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 LIST 已从底层 JICS 存储中正确删除。

删除 GROUP

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/deleteGroup`
- 参数：JSON 有效负载，表示要删除的 JICS GROUP。这是 `com.netfective.bluage.jac.entities.JACGroup` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 GROUP 已从底层 JICS 存储中正确删除。

删除 TRANSACTION

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/deleteTransaction`

- 参数：JSON 有效负载，表示要删除的 JICS TRANSACTION。这是 `com.netfective.bluage.jac.entities.JACTransaction` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 TRANSACTION 已从底层 JICS 存储中正确删除。

删除 PROGRAM

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/deleteProgram`
- 参数：JSON 有效负载，表示要删除的 JICS PROGRAM。这是 `com.netfective.bluage.jac.entities.JACProgram` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 PROGRAM 已从底层 JICS 存储中正确删除。

删除 FILE

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/deleteFile`
- 参数：JSON 有效负载，表示要删除的 JICS FILE。这是 `com.netfective.bluage.jac.entities.JACFile` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 FILE 已从底层 JICS 存储中正确删除。

删除 TDQUEUE

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/deleteTDQueue`
- 参数：JSON 有效负载，表示要删除的 JICS TDQUEUE。这是“com.netfective.bluage.jac.entities.JACTDQueue”对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 TDQUEUE 已从底层 JICS 存储中正确删除。

删除 TSMODEL

- 支持的方法：POST

- 需要身份验证
- 路径：`/api/services/rest/jicsservice/deleteTSModel`
- 参数：JSON 有效负载，表示要删除的 JICS TSMODEL。这是“com.netfective.bluage.jac.entities.JACTSModel”对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 TSMODEL 已从底层 JICS 存储中正确删除。

创建 LIST

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/createList`
- 参数：JSON 有效负载，表示要创建的 JICS LIST。这是“com.netfective.bluage.jac.entities.JACList”对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 LIST 已在底层 JICS 存储中正确创建。

> **Note**
>
> LIST 创建时始终为空。将 GROUP 附加到 LIST 需要执行另一项操作。

创建 GROUP

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/createGroup`
- 参数：JSON 有效负载，表示要创建的 JICS GROUP。这是 `com.netfective.bluage.jac.entities.JACGroup` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 GROUP 已在底层 JICS 存储中正确创建。

> **Note**
>
> GROUP 创建时始终为空。将 RESOURCE 附加到 GROUP 需要额外的操作（创建资源会自动将其附加到给定的 GROUP）。

创建 RESOURCE 的常见注意事项

以下所有端点均与创建 JICS RESOURCE 有关，并具有一些共同的约束条件：在要发送到端点的请求负载中，必须对 groupName 字段赋值。

GROUP 所有权限制：

创建资源时必须将其附加到现有 GROUP，并且端点会使用 groupName 来检索此资源将要附加到的 GROUP。groupName 必须是现有 GROUP 的名称。如果 groupName 未指向 JICS 底层存储中的现有 GROUP，则将发送一条带有 HTTP 状态代码 400 的错误消息。

GROUP 内的唯一性约束：

具有给定名称的给定资源在指定的 GROUP 内必须唯一。每个资源创建端点均会执行唯一性检查。如果给定的有效负载不符合唯一性约束条件，则端点将发送带有 HTTP 状态代码 400（错误请求）的响应，请参阅下面的示例响应。

负载示例：我们尝试在“TEST”GROUP 中创建事务“ARIT”，但该 GROUP 中已经存在同名的事务。

```
{
    "jacType":"JACTransaction",
    "name":"ARIT",
    "groupName":"TEST",
    "isActive":true
  }
```

我们收到了以下错误响应：

```
{
    "timestamp": 1686759054510,
    "status": 400,
    "error": "Bad Request",
    "path": "/jac/api/services/rest/jicsservice/createTransaction"
  }
```

检查服务器日志可确认问题的根源：

```
2023-06-14 18:10:54 default         TRACE - o.s.w.m.HandlerMethod
        - Arguments: [java.lang.IllegalArgumentException: Transaction already
```

```
 present in the group, org.springframework.security.web.header.HeaderWriterFilter
$HeaderWriterResponse@e34f6b8]
2023-06-14 18:10:54 default         ERROR - c.n.b.j.a.WebConfig                    -
 400
java.lang.IllegalArgumentException: Transaction already present in the group
 at
 com.netfective.bluage.jac.server.services.rest.impl.JicsServiceImpl.createElement(JicsServiceI
```

创建 TRANSACTION

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/createTransaction`
- 参数：JSON 有效负载，表示要创建的 JICS TRANSACTION。这是 `com.netfective.bluage.jac.entities.JACTransaction` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 TRANSACTION 已在底层 JICS 存储中正确创建。

创建 PROGRAM

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/createProgram`
- 参数：JSON 有效负载，表示要创建的 JICS PROGRAM。这是 `com.netfective.bluage.jac.entities.JACProgram` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 PROGRAM 已在底层 JICS 存储中正确创建。

创建 FILE

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/createFile`
- 参数：JSON 有效负载，表示要创建的 JICS FILE。这是“com.netfective.bluage.jac.entities.JACFile”对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 FILE 已在底层 JICS 存储中正确创建。

创建 TDQUEUE

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/createTDQueue`
- 参数：JSON 有效负载，表示要创建的 JICS TDQUEUE。这是“com.netfective.bluage.jac.entities.JACTDQueue”对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 TDQUEUE 已在底层 JICS 存储中正确创建。

创建 TSMODEL

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/createTSModel`
- 参数：JSON 有效负载，表示要创建的 JICS TSMODEL。这是 `com.netfective.bluage.jac.entities.JACTSModel` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 TSMODEL 已在底层 JICS 存储中正确创建。

更新 LIST

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/updateList`
- 参数：JSON 有效负载，表示要更新的 JICS LIST。这是 `com.netfective.bluage.jac.entities.JACList` 对象的 JSON 序列化。无需提供 LIST 中的子项，LIST 更新机制也不会考虑这一点。
- 返回一个布尔值。如果值为“true”，则该 LIST 已在底层 JICS 存储中正确更新。

更新 LIST“isActive”标志将传播到 LIST 中所有拥有的元素，即 LIST 拥有的所有 GROUP 以及这些 GROUP 拥有的所有 RESOURCE。这是一种通过单个操作在多个 GROUP 中停用大量资源的便捷方法。

更新 GROUP

- 支持的方法：POST

- 需要身份验证
- 路径：`/api/services/rest/jicsservice/updateGroup`
- 参数：JSON 有效负载，表示要更新的 JICS GROUP。这是 `com.netfective.bluage.jac.entities.JACGroup` 对象的 JSON 序列化。无需提供 GROUP 中的子项，GROUP 更新机制也不会考虑这一点。
- 返回一个布尔值。如果值为“true”，则该 GROUP 已在底层 JICS 存储中正确更新。

> **Note**
>
> 更新 GROUP“isActive”标志将传播到 GROUP 中所有拥有的元素，即 GROUP 拥有的所有 RESOURCE。这是一种通过单个操作在给定 GROUP 中停用大量资源的便捷方法。

RESOURCE 更新常见注意事项

以下所有端点均与更新 JICS RESOURCE 有关。使用 `groupName` 字段，您可以更改任何 JICS RESOURCE 所属的 GROUP，前提是该字段值指向底层 JICS 存储中的现有 GROUP（否则，您将收到来自端点的错误请求响应（HTTP 状态代码 400））。

更新 TRANSACTION

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/updateTransaction`
- 参数：JSON 有效负载，表示要更新的 JICS TRANSACTION。这是 `com.netfective.bluage.jac.entities.JACTransaction` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 TRANSACTION 已在底层 JICS 存储中正确更新。

更新 PROGRAM

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/updateProgram`
- 参数：JSON 有效负载，表示要更新的 JICS PROGRAM。这是 `com.netfective.bluage.jac.entities.JACProgram` 对象的 JSON 序列化。

- 返回一个布尔值。如果值为“true”，则该 PROGRAM 已在底层 JICS 存储中正确更新。

更新 FILE

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/updateFile`
- 参数：JSON 有效负载，表示要更新的 JICS FILE。这是 `com.netfective.bluage.jac.entities.JACFile` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 FILE 已在底层 JICS 存储中正确更新。

更新 TDQUEUE

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/updateTDQueue`
- 参数：JSON 有效负载，表示要更新的 JICS TDQUEUE。这是 `com.netfective.bluage.jac.entities.JACTDQueue` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 TDQUEUE 已在底层 JICS 存储中正确更新。

更新 TSMODEL

- 支持的方法：POST
- 需要身份验证
- 路径：`/api/services/rest/jicsservice/updateTSModel`
- 参数：JSON 有效负载，表示要更新的 JICS TSMODEL。这是 `com.netfective.bluage.jac.entities.JACTSModel` 对象的 JSON 序列化。
- 返回一个布尔值。如果值为“true”，则该 TSMODEL 已在底层 JICS 存储中正确更新。

JAC 用户管理端点

主题

- 删除用户

- 以用户身份登录
- 测试系统中是否至少存在一个用户
- 记录新用户
- 列出用户
- 列出用户
- 注销当前用户
- Jics 服务器运行状况

删除用户

- 支持的方法：POST
- 需要身份验证和管理员权限
- 路径：`/api/services/security/servicelogin/deleteuser`
- 参数：`com.netfective.bluage.jac.entities.SignOn` 对象的 JSON 序列化，表示要从存储中删除的用户。
- 返回一个布尔值。如果该值为“true”，则该用户已从 JICS 系统中正确删除。

> **Note**
>
> 此操作无法撤消，已删除的用户将无法再次连接到 JAC 应用程序。请谨慎使用。

以用户身份登录

- 支持的方法：POST
- 需要身份验证和管理员权限
- 路径：`/api/services/security/servicelogin/login`
- 返回 `com.netfective.bluage.jac.entities.SignOn` 对象的 JSON 序列化，表示在当前请求中提供凭证的用户。在返回的对象中，密码隐藏不可见，并且会列出赋予用户的角色。

示例响应:

```
{
```

```
    "login": "sadmin",
    "password": null,
    "roles": [
      {
        "id": 0,
        "roleName": "ROLE_SUPER_ADMIN"
      }
    ]
  }
```

测试系统中是否至少存在一个用户

- 支持的方法：GET
- 路径：/api/services/security/servicelogin/hasAccount
- 返回一个布尔值。如果在 JICS 系统中至少创建了一个用户（默认超级管理员用户除外），则该值为 true。

记录新用户

- 支持的方法：POST
- 需要身份验证和管理员权限
- 路径：/api/services/security/servicelogin/recorduser
- 参数：com.netfective.bluage.jac.entities.SignOn 对象的 JSON 序列化，表示要添加到存储中的用户。应定义的用户角色，否则用户可能无法使用 JAC 设施和端点。

示例请求：

```
{
    "login": "simpleuser",
    "password": "simpleuser",
    "roles": [
      {
        "id": 2,
        "roleName": "ROLE_USER"
      }
    ]
  }
```

记录新用户时，仅能使用以下角色：

- ROLE_ADMIN：可以管理 JICS 资源和用户。
- ROLE_USER：可以管理 JICS 资源，但不能管理用户。

列出用户

- 支持的方法：GET
- 需要身份验证和管理员权限
- 路径：`/api/services/security/servicelogin/listusers`
- 返回 `com.netfective.bluage.jac.entities.SignOn` 列表，采用 JSON 序列化。

列出用户

- 支持的方法：GET
- 需要身份验证和管理员权限
- 路径：`/api/services/security/servicelogin/listusers`
- 返回 `com.netfective.bluage.jac.entities.SignOn` 列表，采用 JSON 序列化。

注销当前用户

- 支持的方法：GET
- 路径：`/api/services/security/servicelogout/logout`
- 如果当前用户成功注销（相关的 HTTP 会话将失效），则返回以下 JSON 消息：“{"success":true}”。

Jics 服务器运行状况

- 支持的方法：GET
- 路径：`/api/services/rest/jicsserver/serverIsUp`
- 如果您收到的响应中具有 HTTP 状态代码 200，则表示 JICS 服务器已启动并正在运行。

数据结构

本节介绍各种数据结构的详细信息。

主题

作业执行详细信息消息结构

每个作业执行的详细信息都将包含以下字段：

scriptId

被调用脚本的标识符。

caller

调用方的 IP 地址。

identifier

唯一的作业执行标识符。

startTime

作业执行开始日期和时间。

endTime

作业执行结束日期和时间。

status

作业执行的状态。可能为以下值之一：

- DONE：作业执行正常结束。
- TRIGGERED：作业执行已触发但尚未启动。
- RUNNING：作业执行正在运行。
- KILLED：作业执行已被终止。

- FAILED：作业执行失败。

executionResult

用于汇总作业执行结果的消息。如果作业执行尚未完成，则此消息可以是简单消息，也可以是包含以下字段的 JSON 结构：

- exitCode：数字退出码；负值表示故障情况。
- program：该任务启动的最新程序。
- status：可能为以下值之一：
  - Error：当 exitCode = -1 时；该值表示作业执行期间发生（技术）错误。
  - Failed：当 exitcode = -2 时；该值表示服务程序执行期间发生故障（例如 ABEND 情况）。
  - Succeeded：当 exitCode >= 0 时；
- stepName：作业中最近执行的步骤的名称。

executionMode

可以是 SYNCHRONOUS 或 ASYNCHRONOUS，具体取决于作业的启动方式。

示例输出：

```
{
    "scriptId": "INTCALC",
    "caller": "127.0.0.1",
    "identifier": "97d410be-efa7-4bd3-b7b9-d080e5769771",
    "startTime": "06-09-2023 11:42:41",
    "endTime": "06-09-2023 11:42:42",
    "status": "DONE",
    "executionResult": "{ \"exitCode\": -1, \"stepName\": \"STEP15\", \"program\":
 \"CBACT04C\", \"status\": \"Error\" }",
    "executionMode": "ASYNCHRONOUS"
  }
```

事务启动结果结构

该结构可以包含以下字段：

outCome

代表事务执行结果的字符串。可能的值有：

- Success：事务执行正常结束。

- `Failure`：事务执行未能正常结束，遇到了一些问题。

commarea

表示 COMMAREA 最终值（byte64 编码字节数组）的字符串，可以是空字符串。

containerRecord

（可选）表示容器的记录内容（byte64 编码字节数组）的字符串。

serverDescription

可能包含有关服务于请求的服务器信息（用于调试目的），可以是空字符串。

abendCode

（可选）如果启动的事务所引用的程序中止，则在此字段中将以字符串形式返回中止代码值。

示例响应：

成功

```
{
    "outCome": "Success",
    "commarea": "",
    "serverDescription": ""
  }
```

Failure

```
{
    "outCome": "Failure",
    "commarea": "",
    "serverDescription": "",
    "abendCode": "AEIA"
  }
```

事务启动记录结果结构

该结构可以包含以下字段：

recordContent

表示 COMMAREA 的记录内容（byte64 编码字节数组）的字符串。

containerRecord

表示容器的记录内容（byte64 编码字节数组）的字符串。

serverDescription

可能包含有关服务于请求的服务器信息（用于调试目的），可以是空字符串。

示例响应：

成功

```
{
    "recordContent": "",
    "serverDescription": ""
}
```

队列中作业的可能状态

在队列中，作业可具有以下状态：

ACTIVE

作业目前正在队列中运行。

EXECUTION_WAIT

作业正在等待线程可用。

SCHEDULED

作业计划在特定的日期和时间执行。

HOLD

作业正在等待释放后再运行。

COMPLETED

作业已成功执行。

FAILED

作业执行失败。

UNKNOWN

状态未知。

提交作业输入

提交作业输入是 com.netfective.bluage.gapwalk.rt.jobqueue.SubmitJobMessage 对象的 JSON 序列化。以下示例输入显示了此类 Bean 的所有字段。

示例输入：

```
{
    "messageQueueName":null,
    "scheduleDate":null,
    "scheduleTime":null,
    "programName":"PTA0044",
    "programParams":
     {"wmind":"B"},
    "localDataAreaValue":"",
    "userName":"USER1",
    "jobName":"PTA0044",
    "jobNumber":9,
    "jobPriority":5,
    "executionDate":"20181231",
    "jobQueue":"queue1",
    "jobOnHold":false
}
```

jobNumber

如果 jobnumber 为 0，则将使用作业编号序列中的下一个编号自动生成作业编号。该值应设置为 0（测试目的除外）。

jobPriority

AS400 中的默认作业优先级为 5。有效范围为 0 – 9，0 为最高优先级。

jobOnHold

如果作业被暂停提交，则不会立即执行，而仅在有人将其“释放”时才会执行。可以使用 REST API（/release 或 /releasease-all）释放作业。

scheduleDate 和 scheduleTime

如果这些值不为空，则作业将在指定的日期和时间执行。

Date

可以 MMddyy 或 ddMMyyyy 格式提供（输入的大小将决定所使用的格式）

时间

可以 HHmm 或 HHmmss 格式提供（输入的大小将决定所使用的格式）

programParams

将作为映射传递给程序。

计划作业响应列表

以下是列表作业的作业队列端点的结构。请注意，用于提交该作业的提交作业消息是响应的一部分。此端点结构可用于追踪或测试/重新提交。作业完成后，开始日期和结束日期也会填充。

```
[
    {
      "qrtzJobGroup": "PTA0044-PTA0044",
      "qrtzJobName": "PTA0044-168156109957800",
      "status": "HOLD",
      "qrtzDelay": 0,
      "startDate": null,
      "endDate": null,
      "jobName": "PTA0044",
      "userName": "USER1",
      "jobNumber": 9,
      "jobPriority": 5,
      "jobQueue": "queue1",
      "message": {
        "messageQueueName": null,
        "scheduleDate": null,
        "scheduleTime": null,
        "programName": "PTA0044",
        "programParams": {
          "wmind": "B"
        },
        "localDataAreaValue": "",
        "userName": "USER1",
        "jobName": "PTA0044",
        "jobNumber": 9,
        "jobPriority": 5,
        "executionDate": "20181231",
        "jobQueue": "queue1",
        "jobOnHold": true
      }
```

```
    },
    {
      "qrtzJobGroup": "PTA0044-PTA0044",
      "qrtzJobName": "PTA0044-166196954877600",
      "status": "COMPLETED",
      "qrtzDelay": 0,
      "startDate": "2022-10-13T22:48:34.025+00:00",
      "endDate": "2022-10-13T22:52:54.475+00:00",
      "jobName": "PTA0044",
      "userName": "USER1",
      "jobNumber": 9,
      "jobPriority": 5,
      "jobQueue": "queue1",
      "message": {
        "messageQueueName": null,
        "scheduleDate": null,
        "scheduleTime": null,
        "programName": "PTA0044",
        "programParams": {
          "wmind": "B"
        },
        "localDataAreaValue": "",
        "userName": "USER1",
        "jobName": "PTA0044",
        "jobNumber": 9,
        "jobPriority": 5,
        "executionDate": "20181231",
        "jobQueue": "queue1",
        "jobOnHold": true
      }
    }
  ]
```

“暂停”作业响应列表

该结构与上述结构类似，唯一的不同是所有返回的作业均为“暂停”状态。

```
[
    {
      "qrtzJobGroup": "PTA0044-PTA0044",
      "qrtzJobName": "PTA0044-168156109957800",
      "status": "HOLD",
```

```
      "qrtzDelay": 0,
      "startDate": null,
      "endDate": null,
      "jobName": "PTA0044",
      "userName": "USER1",
      "jobNumber": 9,
      "jobPriority": 5,
      "jobQueue": "queue1",
      "message": {
        "messageQueueName": null,
        "scheduleDate": null,
        "scheduleTime": null,
        "programName": "PTA0044",
        "programParams": {
          "wmind": "B"
        },
        "localDataAreaValue": "",
        "userName": "USER1",
        "jobName": "PTA0044",
        "jobNumber": 9,
        "jobPriority": 5,
        "executionDate": "20181231",
        "jobQueue": "queue1",
        "jobOnHold": true
      }
    }
  ]
```

AWS Blu Age 运行时（非托管）设置

本节介绍在 AWS 基础架构上设置 AWS Blu Age Runtime（非托管）的步骤。

主题

- AWS Blu Age 运行时必
- AWS Blu Age 运行时入门
- AWS Blu Age Runtime（非托管）的基础架构设置要求
- AWS Amazon ECS 上的 Blu Age 运行时部署由 AWS Fargate
- AWS 在 Amazon EC2 上部署 Blu Age 运行时

AWS Blu Age 运行时必

AWS Blu Age Runtime（非托管）有多个发行版本可用。如果您正在进行现代化项目，则可能需要运行时的增量版本用于实施和测试目的。要确定您的需求，请联系您的 AWS Blu Age 配送经理。

在开始 AWS Blu Age Runtime（非托管）入职流程之前，请执行以下操作：

- 请确保您有一个 AWS 帐户。
- 确保你有一个使用 Blu Age 重构的现代化应用程序。 AWS
- 选择一个 AWS 区域和一个计算（开启 Amazon EC2 或 Amazon ECS AWS Fargate）。
- 选择你要使用的 AWS 蓝光时代运行时版本。
- 查看the section called “基础设施设置要求”并验证运行 AWS Blu Age 运行时所需的其他组件（非托管）。

> **Note**
>
> 如果你想测试 AWS Blu Age Runtime（非托管）的功能，你可以使用演示应用程序`Planets Demo`，你可以从 PlanetsDemo-v1.zip 下载该应用程序。

AWS Blu Age 运行时入门

首先，请创建一个 AWS Support 案例，请求入职以访问 AWS Blu Age Runtime。在您的请求中包括您的 AWS 账户 ID、您要使用的 AWS 区域、计算选择和运行时版本。如果您不确定需要哪个版本，请联系您的 AWS Blu Age 交付经理。

AWS 亚马逊 EC2 上的 Blu Age 运行时（非托管）

我们按地区和计算选择将 AWS Blu Age 运行时（非托管）项目存储在不同的 Amazon S3 存储桶中。要在 Amazon EC2 上访问您 AWS 区域 的 AWS blu Age 运行时（非托管）的存储桶，请使用下表中列出的名称。

AWS 区域	存储桶释放	存储桶构建
美国东部（俄亥俄州）	aws-bluage-runtime-artifacts-055777665268-us-east-2	aws-bluage-runtime-artifacts-dev-055777665268-us-east-2

AWS 区域	存储桶释放	存储桶构建
美国东部（弗吉尼亚州北部）	aws-bluage-runtime-artifacts-139023371234-us-east-1	aws-bluage-runtime-artifacts-dev-139023371234-us-east-1
美国西部（北加利福尼亚）	aws-bluage-runtime-artifacts-788454048782-us-west-1	aws-bluage-runtime-artifacts-dev-788454048782-us-west-1
US West（Oregon）	aws-bluage-runtime-artifacts-836771190483-us-west-2	aws-bluage-runtime-artifacts-dev-836771190483-us-west-2
欧洲地区（爱尔兰）	aws-bluage-runtime-artifacts-925278190477-eu-west-1	aws-bluage-runtime-artifacts-dev-925278190477-eu-west-1
欧洲地区（巴黎）	aws-bluage-runtime-artifacts-673009995881-eu-west-3	aws-bluage-runtime-artifacts-dev-673009995881-eu-west-3
欧洲地区（法兰克福）	aws-bluage-runtime-artifacts-485196800481-eu-central-1	aws-bluage-runtime-artifacts-dev-485196800481-eu-central-1
South America（São Paulo）	aws-bluage-runtime-artifacts-737536804457-sa-east-1	aws-bluage-runtime-artifacts-dev-737536804457-sa-east-1
亚太地区（东京)	aws-bluage-runtime-artifacts-445578176276-ap-northeast-1	aws-bluage-runtime-artifacts-dev-445578176276-ap-northeast-1
亚太地区（悉尼）	aws-bluage-runtime-artifacts-726160321909-ap-southeast-2	aws-bluage-runtime-artifacts-dev-726160321909-ap-southeast-2

AWS 由 Fargate 管理的 Amazon ECS 上的 Blu Age Runtime（非托管）

我们按地区和计算选择将 AWS Blu Age 运行时（非托管）项目存储在不同的 Amazon S3 存储桶中。要访问由 Fargate 管理的 AWS 区域 Amazon ECS 上的 AWS Blu Age Runtime（非托管）存储桶，请使用下表中列出的名称。

AWS 区域	存储桶释放	存储桶构建
美国东部（俄亥俄州）	aws-bluage-runtime-fargate-rel-483416914331-us-east-2	aws-bluage-runtime-fargate-dev-483416914331-us-east-2
美国东部（弗吉尼亚州北部）	aws-bluage-runtime-fargate-rel-308472162679-us-east-1	aws-bluage-runtime-fargate-dev-308472162679-us-east-1
美国西部（北加利福尼亚）	aws-bluage-runtime-fargate-rel-343763094578-us-west-1	aws-bluage-runtime-fargate-dev-343763094578-us-west-1
US West（Oregon）	aws-bluage-runtime-fargate-rel-688933007849-us-west-2	aws-bluage-runtime-fargate-dev-688933007849-us-west-2
欧洲地区（爱尔兰）	aws-bluage-runtime-fargate-rel-140138033705-eu-west-1	aws-bluage-runtime-fargate-dev-140138033705-eu-west-1
欧洲地区（巴黎）	aws-bluage-runtime-fargate-rel-339712948211-eu-west-3	aws-bluage-runtime-fargate-dev-339712948211-eu-west-3
欧洲地区（法兰克福）	aws-bluage-runtime-fargate-rel-339712918892-eu-central-1	aws-bluage-runtime-fargate-dev-339712918892-eu-central-1
South America（São Paulo）	aws-bluage-runtime-fargate-rel-767397998881-sa-east-1	aws-bluage-runtime-fargate-dev-767397998881-sa-east-1
亚太地区（东京)	aws-bluage-runtime-fargate-rel-891377400849-ap-northeast-1	aws-bluage-runtime-fargate-dev-891377400849-ap-northeast-1
亚太地区（悉尼）	aws-bluage-runtime-fargate-rel-533267435478-ap-southeast-2	aws-bluage-runtime-fargate-dev-533267435478-ap-southeast-2

使用命令行列出存储桶的内容

登录后，您可以在终端中运行以下命令来列出存储桶的内容。

```
aws s3 ls bucket-name
```

bucket-name替换为上表 AWS 区域 中您的存储桶的名称。

此命令返回与不同版本的 AWS Blu Age Runtime（非托管）运行时相对应的文件夹列表，例如

```
PRE 3.3.0.1/
PRE 3.3.0.2/
```

建议您使用最新可用版本。如果这不可能，则使用在应用程序重构阶段验证过的运行时版本。要列出特定版本的可用框架，请运行以下命令：

```
aws s3 ls s3://bucket-name/version/Framework/
```

bucket-name替换为你的存储桶version名称 AWS 区域 和你想要的版本。以下是示例。

```
aws s3 ls s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/
Framework/
```

该命令将返回框架列表，例如：

```
2023-06-05 10:26:52   97960225 aws-bluage-runtime-3.4.0.tar.gz
2023-06-05 10:27:12         45 aws-bluage-runtime-3.4.0.tar.gz.checksumSHA256
2023-06-05 10:27:14  138497123 aws-bluage-webapps-3.4.0.tar.gz
2023-06-05 10:27:44         45 aws-bluage-webapps-3.4.0.tar.gz.checksumSHA256
```

下载框架

例如，您可以下载框架来升级现有 Amazon EC2 实例上的 Velocity 运行时版本。

```
aws s3 cp s3://bucket-name/version/Framework/ folder-of-your-choice --
recursive
```

其中：

folder-of-your-choice

要下载框架的文件夹路径。

例如：`aws s3 cp s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/Framework/ . --recursive`

此命令将生成以下输出：

```
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/
Framework/aws-bluage-runtime-3.4.0.tar.gz.checksumSHA256 to ./aws-bluage-
runtime-3.4.0.tar.gz.checksumSHA256
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/
Framework/aws-bluage-webapps-3.4.0.tar.gz.checksumSHA256 to ./aws-bluage-
webapps-3.4.0.tar.gz.checksumSHA256
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/Framework/aws-
bluage-webapps-3.4.0.tar.gz to ./aws-bluage-webapps-3.4.0.tar.gz
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/Framework/aws-
bluage-runtime-3.4.0.tar.gz to ./aws-bluage-runtime-3.4.0.tar.gz
```

您可以使用如下命令列出框架文件：

```
ls -l
```

此命令将生成以下输出：

```
total 230928
-rw-rw-r-- 1 cloudshell-user cloudshell-user  97960225 Jun  5 10:26 aws-bluage-
runtime-3.4.0.tar.gz
-rw-rw-r-- 1 cloudshell-user cloudshell-user        45 Jun  5 10:27 aws-bluage-
runtime-3.4.0.tar.gz.checksumSHA256
-rw-rw-r-- 1 cloudshell-user cloudshell-user 138497123 Jun  5 10:27 aws-bluage-
webapps-3.4.0.tar.gz
-rw-rw-r-- 1 cloudshell-user cloudshell-user        45 Jun  5 10:27 aws-bluage-
webapps-3.4.0.tar.gz.checksumSHA256
```

AWS Blu Age Runtime（非托管）的基础架构设置要求

本主题介绍运行 AWS Blu Age Runtime（非托管）所需的最低基础架构配置。以下过程描述了如何在您选择的计算机上设置 AWS Blu Age Runtime（非托管），以便在 AWS Blu Age Runtime 上部署现代化应用程序。您创建的资源必须位于具有专用于您的应用程序域的子网的 Amazon VPC 中。

创建安全组

1. 通过 https://console.aws.amazon.com/vpc/ 打开 Amazon VPC 控制台。
2. 在左侧导航窗格中的安全下，选择安全组。
3. 在中央窗格中，选择创建安全组。
4. 在安全组名称字段中，输入 **M2BluagePrivateLink-SG**。
5. 在入站规则部分中，选择添加规则。

6. 对于类型，选择 HTTPS。
7. 对于来源，请输入您的 VPC CIDR。
8. 在出站规则部分，选择添加规则。
9. 对于类型，选择 HTTPS。
10. 在目标位置字段，输入 **0.0.0.0/0**。
11. 选择创建安全组。

创建 Amazon VPC 端点

1. 通过 https://console.aws.amazon.com/vpc/ 打开 Amazon VPC 控制台。
2. 在左侧导航窗格中的虚拟私有云下，选择端点。
3. 在中央窗格中，选择创建端点。
4. 在 “服务” 部分，在搜索字段**SQS**中输入，然后选择与您所在地区对应的 Amazon SQS 服务。
5. 在 VPC 部分，选择在上一步骤中创建的 Amazon VPC。
6. 在子网部分，选择您为应用程序域创建的子网。
7. 在 “安全组” 部分中，选择 M2-BluagePrivateLink SG。
8. 选择创建端点。

创建 IAM 策略

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中的访问管理下，选择策略。
3. 在中央窗格中，选择创建策略。
4. 在策略编辑器部分，选择 JSON。
5. 将您在编辑器中看到的所有 JSON 替换为以下 JSON。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "sqs:GetQueueUrl",
```

```
                "sqs:ReceiveMessage",
                "sqs:SendMessage"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 如果您需要更多详细信息来定制您的政策，请联系您的 AWS Blu Age 配送经理或客户经理。

6. 选择下一步。
7. 为策略输入名称，然后选择创建策略。

创建 IAM 角色

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中的访问管理下，选择角色。
3. 在中央窗格中，选择创建角色。
4. 在用例部分，根据您的计算选择，选择 EC2 或弹性容器服务（然后选择弹性容器服务任务）。
5. 选择下一步。
6. 在搜索框中，输入前面创建的策略的名称。
7. 选中您的策略左边的复选框。
8. 选择下一步。
9. 输入角色的名称，然后选择创建角色。

在 Amazon EC2 上运行 AWS Blu Age 运行时

创建 Amazon EC2 实例

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 选择启动实例。
3. 对于实例类型，请选择中列出的类型之一the section called “AWS Blu Age 运行时的 Amazon EC2 实例类型（在亚马逊 EC2 上）”。

4. 在密钥对部分中，选择现有密钥对或创建新密钥对。
5. 在网络设置部分，选择选择现有安全组。
6. 对于通用安全组，请选择 M2-BluagePrivateLink SG。
7. 展开高级详细信息部分。
8. 对于 IAM 实例配置文件，选择您前面创建的 IAM 角色。
9. 选择启动实例。

当 Amazon EC2 实例的状态变成正在运行时，连接到该实例并安装以下软件组件：

- Java 运行时环境（JRE）11。
- Apache Tomcat 9。
- AWS Blu Age Runtime（在 Amazon EC2 上）。在 Apache Tomcat 安装文件夹的根目录下安装 AWS Blu Age 运行时（有些文件将被添加，而另一些文件将被覆盖）。

如果要安装其他 Web 应用程序，请将其安装在单独的文件夹中。在这种情况下，请重复安装 Apache Tomcat 的过程，然后在上面安装存档。

在 Amazon ECS 上运行 AWS Blu Age Runtime 由 AWS Fargate

创建 Amazon ECS 集群和任务角色，以便稍后启动 AWS Fargate 任务时使用。对于任务角色，请包括您之前创建的策略。根据您的场景，您可能需要为任务角色附加其他 IAM 策略。

AWS Blu Age 运行时的 Amazon EC2 实例类型（在亚马逊 EC2 上）

以下是您可以用于 AWS Blu Age 运行时（在 Amazon EC2 上）的 Amazon EC2 实例类型列表。

```
t3.xlarge
t3.small
t3.large
t2.small
t2.large
r6i.xlarge
r6i.large
r6i.4xlarge
r6i.2xlarge
r5b.xlarge
r5b.large
r5b.2xlarge
r3.xlarge
```

```
m6i.xlarge
m6i.large
m6i.8xlarge
m6i.4xlarge
m6i.2xlarge
m6i.16xlarge
m5zn.xlarge
m5zn.large
m5zn.3xlarge
m5zn.2xlarge
m5.xlarge
m5.large
m5.8xlarge
m5.4xlarge
m5.2xlarge
m5.16xlarge
m5.12xlarge
c6i.xlarge
c6i.large
c6i.8xlarge
c6i.4xlarge
c6i.2xlarge
c6i.16xlarge
c5.xlarge
c5.large
c5.9xlarge
c5.4xlarge
c5.2xlarge
c5.18xlarge
c5.12xlarge
```

AWS Amazon ECS 上的 Blu Age 运行时部署由 AWS Fargate

本节中的主题介绍如何在由管理的 Amazon ECS 上设置 AWS Blu Age Runtime AWS Fargate、如何更新运行时版本、如何使用 Amazon CloudWatch 警报监控您的部署以及如何添加许可依赖项。

主题

- 在由管理的 Amazon ECS 上设置 AWS Blu Age Runtime AWS Fargate
- 升级由管理的 Amazon ECS 上的 AWS Blu Age 运行时 AWS Fargate
- 亚马逊 CloudWatch ECS 上的 AWS Blu Age 运行时的 Amazon Alarms 由 AWS Fargate
- 在 Amazon ECS 上的 AWS Blu Age Runtime 中设置许可依赖关系 AWS Fargate

在由管理的 Amazon ECS 上设置 AWS Blu Age Runtime AWS Fargate

本主题介绍如何在由管理的 Amazon ECS 上使用 AWS Blu Age Runtime 设置和部署 PlanetsDemo 示例应用程序 AWS Fargate。

AWS 由管理的亚马逊 ECS 上的 Blu Age Runtime 可用 AWS Fargate 于 Linux/X86。

主题

先决条件

在开始之前，请确保满足以下先决条件：

- AWS CLI 按照配置 AWS CLI 中的步骤进行配置。
- 填写 the section called “AWS Blu Age 运行时必” 和 the section called “AWS Blu Age 运行时入门”。
- 在 Amazon ECS 上下载由 AWS Fargate 二进制文件管理的 B AWS lu Age 运行时。有关说明，请参阅the section called “AWS Blu Age 运行时入门”。
- 下载 Apache Tomcat 9 二进制文件。
- 下载PlanetsDemo应用程序档案。
- 为 JICS 创建一个 Amazon Aurora PostgreSQL 数据库，然后对其运行`PlanetsDemo-v1/jics/sql/initJics.sql`查询。有关如何创建 Amazon Aurora PostgreSQL 数据库的信息，请参阅创建和连接 Aurora Post greSQL 数据库集群。

设置

要设置 PlanetsDemo 示例应用程序，请完成以下步骤。

1. 下载 Apache Tomcat 二进制文件后，解压缩内容并转到文件夹。`conf`打开`catalina.properties`文件进行编辑，然后将以下行开头的`common.loader`行替换为下一行。

```
common.loader="${catalina.base}/lib","${catalina.base}/lib/
*.jar","${catalina.home}/lib","${catalina.home}/lib/*.jar","${catalina.home}/
```

```
shared","${catalina.home}/shared/*.jar","${catalina.home}/extra","${catalina.home}/
extra/*.jar"
```

2. 使用 tar 命令压缩 Apache Tomcat 文件夹来构建 “tar.gz” 存档。
3. 准备一个 Dockerfile，以便根据提供的运行时二进制文件和 Apache Tomcat 服务器二进制文件构建您的自定义镜像。请参阅以下 Dockerfile 示例。目标是安装 Apache Tomcat 9，然后安装在 Apache Tomcat 9 安装目录根目录中提取的 AWS Blu Age Runtime（适用于由管理的 Amazon ECS AWS Fargate），然后安装名为的示例现代化应用程序。 PlanetsDemo

> **Note**
>
> 在本示例 Dockerfile 中使用的 install-gapwalk.sh 和 install-app.sh 脚本的内容列在 Dockerfile 之后。

```
FROM --platform=linux/x86_64 amazonlinux:2

RUN mkdir -p /workdir/apps
WORKDIR /workdir
COPY install-gapwalk.sh .
COPY install-app.sh .
RUN chmod +x install-gapwalk.sh
RUN chmod +x install-app.sh

# Install Java and AWS CLI v2-y
RUN yum install sudo java-17-amazon-corretto unzip tar -y
RUN sudo yum remove awscli -y
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
 "awscliv2.zip"
RUN sudo unzip awscliv2.zip
RUN sudo ./aws/install

#·Installation dir
RUN mkdir -p /usr/local/velocity/installation/gapwalk
# Copy PlanetsDemo archive to a dedicated apps dir
COPY PlanetsDemo-v1.zip /workdir/apps/

# Copy resources (tomcat, blu age runtime) to installation dir
COPY tomcat.tar.gz /usr/local/velocity/installation/tomcat.tar.gz
COPY aws-bluage-on-fargate-runtime-3.10.0.15.tar.gz /usr/local/velocity/
installation/gapwalk/gapwalk-bluage-on-fargate.tar.gz
```

```
# run relevant installation scripts
RUN ./install-gapwalk.sh
RUN ./install-app.sh

EXPOSE 8080
EXPOSE 8081
# ...

# Run Command to start Tomcat server
CMD ["sh", "-c", "sudo /bluage-on-fargate/tomcat.gapwalk/velocity/startup.sh
 $ECS_CONTAINER_METADATA_URI_V4 $AWS_CONTAINER_CREDENTIALS_RELATIVE_URI"]
```

以下是 install-gapwalk.sh 的内容。

```
#!/bin/sh

# Vars
TEMP_DIR=/bluage-on-fargate/tomcat.gapwalk/temp

# Install
echo "Installing Gapwalk and Tomcat"
sudo rm -rf /bluage-on-fargate
mkdir -p ${TEMP_DIR}
# Copy Blu Age runtime and tomcat archives to temporary extraction dir
sudo cp /usr/local/velocity/installation/gapwalk/gapwalk-bluage-on-fargate.tar.gz
 ${TEMP_DIR}
sudo cp /usr/local/velocity/installation/tomcat.tar.gz ${TEMP_DIR}
#·Create velocity dir
mkdir -p /bluage-on-fargate/tomcat.gapwalk/velocity
#·Extract tomcat files
tar -xvf ${TEMP_DIR}/tomcat.tar.gz -C ${TEMP_DIR}
# Copy all tomcat files to velocity dir
cp -fr ${TEMP_DIR}/apache-tomcat-9.x.x/* /bluage-on-fargate/tomcat.gapwalk/velocity
# Remove default webapps of Tomcat
rm -f /bluage-on-fargate/tomcat.gapwalk/velocity/webapps/*
# Extract Blu Age runtime at velocity dir
tar -xvf ${TEMP_DIR}/gapwalk-bluage-on-fargate.tar.gz -C /bluage-on-fargate/
tomcat.gapwalk
# Remove temporary extraction dir
sudo rm -rf ${TEMP_DIR}
```

以下是 install-app.sh 的内容。

```
#!/bin/sh

APP_DIR=/workdir/apps
TOMCAT_GAPWALK_DIR=/bluage-on-fargate/tomcat.gapwalk

unzip ${APP_DIR}/PlanetsDemo-v1.zip -d ${APP_DIR}
cp -r ${APP_DIR}/webapps/* ${TOMCAT_GAPWALK_DIR}/velocity/webapps/
cp -r ${APP_DIR}/config/* ${TOMCAT_GAPWALK_DIR}/velocity/config/
```

4. 在文件中（位于application-main.yml文件{TOMCAT_GAPWALK_DIR}/config夹）中提供作为先决条件一部分而创建的数据库的连接信息。有关更多信息，请参阅创建并连接到 Aurora PostgreSQL 数据库集群。

```
datasource:
   jicsDs:
     driver-class-name :
     url:
     username:
     password:
     type :
```

5. 生成镜像并将其推送到您的 Amazon ECR 存储库。有关说明，请参阅《亚马逊弹性容器注册表用户指南》中的推送 Docker 镜像。
6. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。
7. 在左侧导航窗格中，选择 Task definitions（任务定义）。
8. 对于 “启动类型”，选择 AWS Fargate。
9. 选择您作为其一部分创建的任务角色the section called “基础设施设置要求”。
10. 将您的图像附加到容器上。
11. 填写完表单，然后选择 “创建”。
12. 在左侧导航窗格中，选择集群，然后从列表中选择您的集群。
13. 在集群的详细信息页面的服务选项卡上，选择创建。
14. 选择任务定义。
15. 展开 “网络” 部分，配置您在其中the section called “基础设施设置要求”创建的 VPC、子网和安全组。
16. 部署您的亚马逊 ECS 服务。

如果部署失败，请检查日志。要找到它们，请前往 Amazon ECS 中由管理的任务页面 AWS Fargate，然后选择日志选项卡。如果您发现以 C 后跟数字开头的错误代码（例如 CXXXX），请记下错误消息。例如，错误代码 C5102 是一个常见错误，表示基础架构配置不正确。您还可以在正在运行的任务中导航并运行一些命令，类似于 AWS Blu Age Runtime（在 Amazon EC2 上）。有关更多信息，请参阅《亚马逊弹性容器服务开发人员指南》中的 “使用 Amazon ECS Exec 进行调试”。

要打开交互式 shell，请在本地计算机上运行以下命令。

```
aws ecs execute-command --cluster your_cluster_name --container your_container_name --
task task_id --interactive --command /bin/sh
```

测试 PlanetsDemo 应用程序

要检查已部署 PlanetsDemo 应用程序的状态，请在替换`load-balancer-DNS-name`、`listener-port`和之后`web-binary-name`使用正确的设置值运行以下命令。

```
curl http://load-balancer-DNS-name:listener-port/gapwalk-application/
```

如果应用程序正在运行，则会看到以下输出消息：`Jics application is running`。

接下来，运行以下命令。

```
curl http://load-balancer-DNS-name:listener-port/jac/api/services/rest/jicsservice/
```

如果应用程序正在运行，则会看到以下输出消息：`Jics application is running`。

```
Jics application is running
```

如果您已配置 Blusam，则运行以下命令时可能会出现空响应。

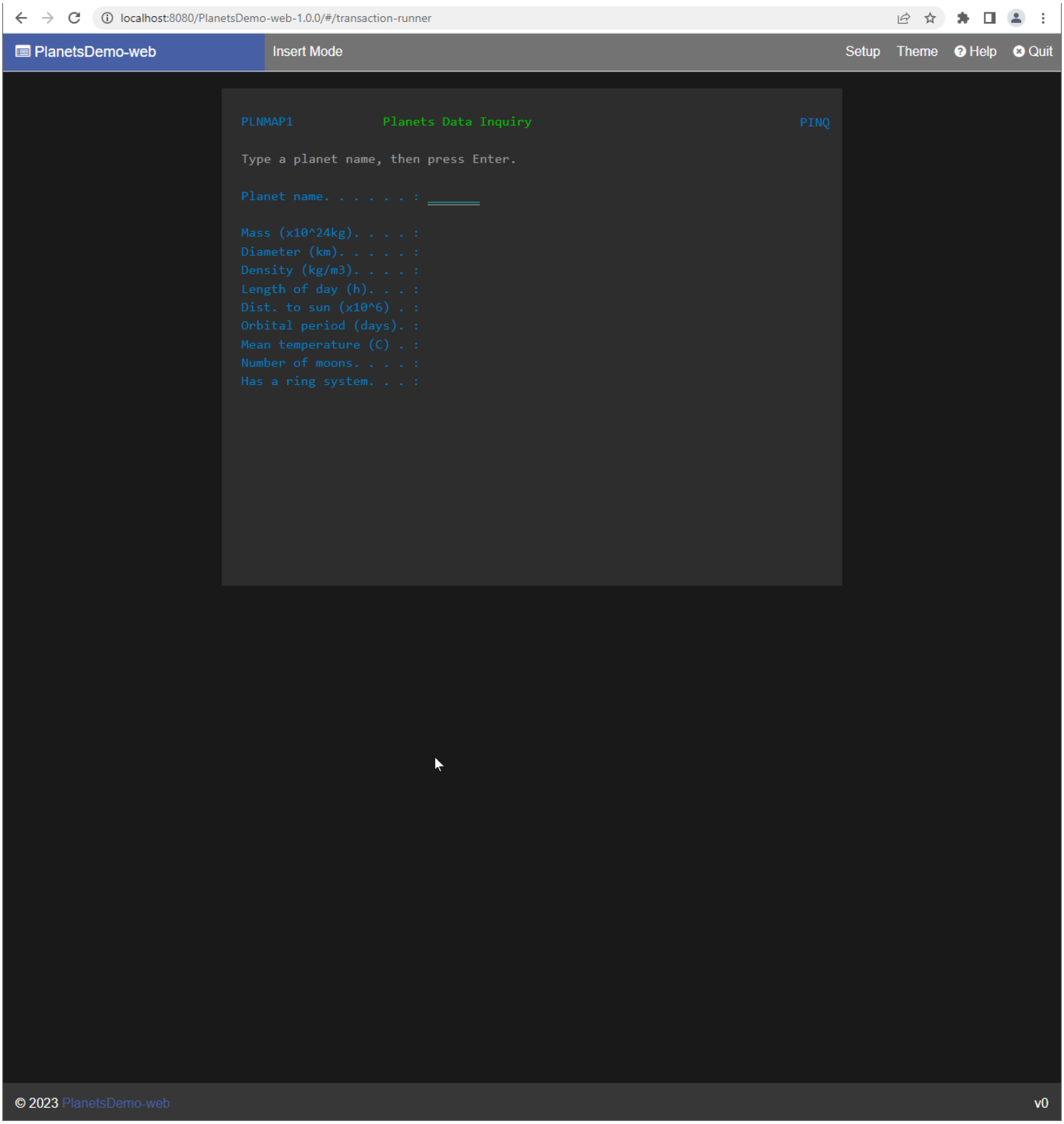
```
curl http://load-balancer-DNS-name:listener-port/bac/api/services/rest/bluesamserver/
serverIsUp
```

注意 Web 二进制文件的名称（PlanetsDemo-web-1.0.0，如果未更改）。要访问 PlanetsDemo应用程序，请使用以下格式的 URL。

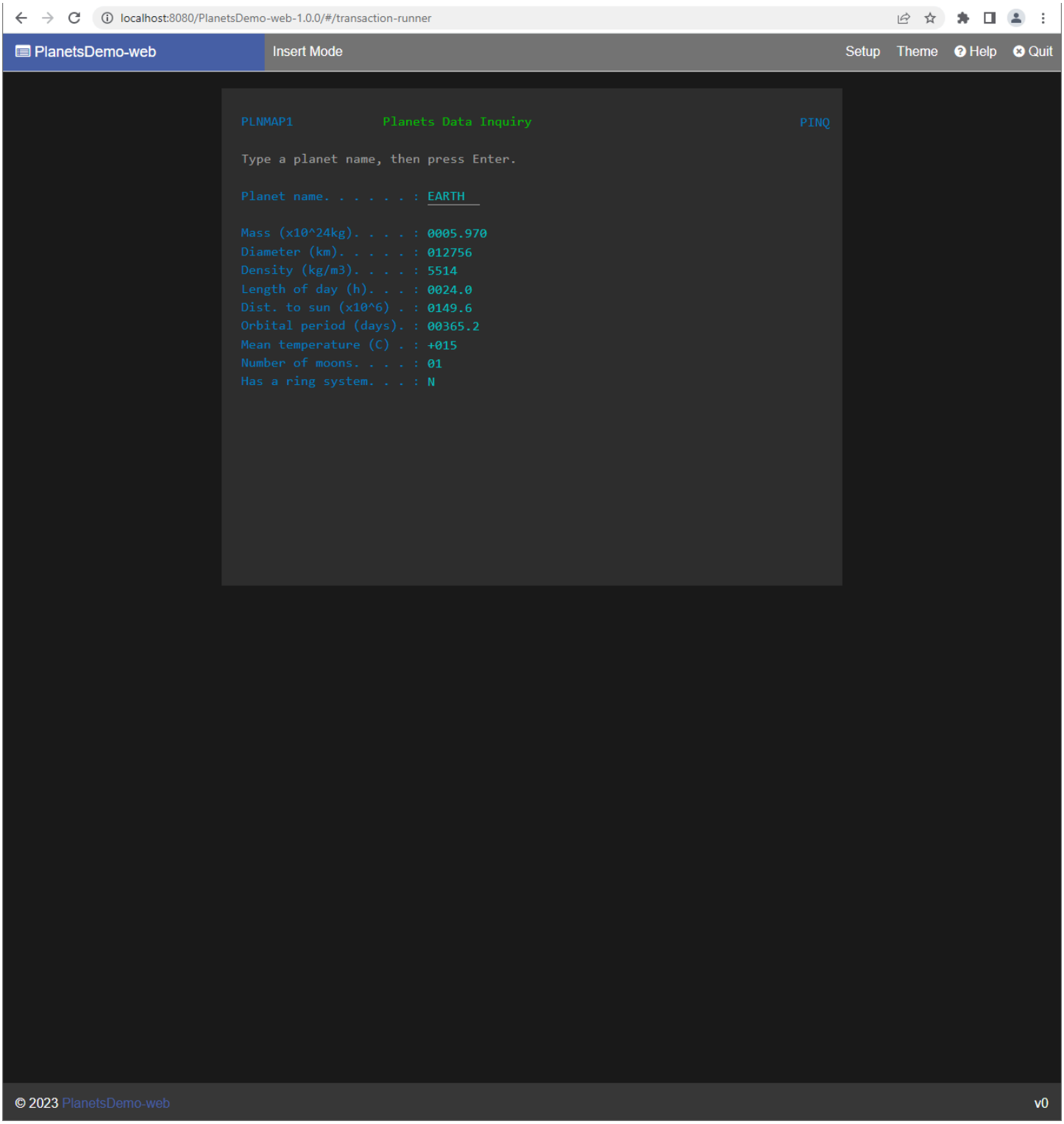
```
https://load-balancer-DNS-name:listener-port/web-binary-name
```

PlanetsDemo 应用程序启动后，将显示主页。

在文本框中，输入 PINQ，然后按 Enter。随后显示数据查询页面。



例如，在 PlanetsDemo 名称字段中输入 EARTH，然后按 Enter。随后将显示您输入的行星页面。



升级由管理的 Amazon ECS 上的 AWS Blu Age 运行时 AWS Fargate

本指南介绍如何升级由管理的 Amazon ECS 上的 AWS Blu Age 运行时 AWS Fargate。

主题

先决条件

在开始之前，确保满足以下先决条件：

- 填写 the section called “AWS Blu Age 运行时必” 和 the section called “AWS Blu Age 运行时入门”。
- 下载你想要升级到的 AWS Blu Age Runtime 版本。有关更多信息，请参阅the section called “AWS Blu Age 运行时入门”。该框架由两个二进制文件组成：`aws-bluage-runtime-x.x.x.x.tar.gz`和`aws-bluage-webapps-x.x.x.x.tar.gz`。

升级 Velocity 运行时

完成以下步骤来升级 Velocity 运行时。

1. 使用所需的 AWS Blu Age 运行时版本重建 Docker 镜像。有关说明，请参阅the section called “在由管理的 Amazon ECS 上设置 AWS Blu Age Runtime AWS Fargate”。
2. 将你的 Docker 镜像推送到你的 Amazon ECR 存储库。
3. 停止并重启您的 Amazon ECS 服务。
4. 验证日志。

AWS 蓝光时代运行时间已成功升级。

亚马逊 CloudWatch ECS 上的 AWS Blu Age 运行时的 Amazon Alarms 由 AWS Fargate

要在部署的应用程序遇到异常时获得更多可见的通知，请设置 CloudWatch 为接收应用程序日志，并添加警报以警告您可能出现的错误。

警报设置

借助 CloudWatch 日志，您可以根据您的应用程序和需求配置任意数量的指标和警报。

具体而言，您可以在创建 Amazon ECS 集群期间直接为使用情况警报设置主动警报，以便在出现错误时收到通知。要突出显示与 AWS Blu Age 控制系统连接中的错误，请在日志中添加与字符串 “Error C” 相关的指标。然后，您可以定义一个对此指标做出反应的警报。

在 Amazon ECS 上的 AWS Blu Age Runtime 中设置许可依赖关系 AWS Fargate

本主题介绍如何设置其他许可依赖项，这些依赖项可与管理的 Amazon ECS 上的 AWS Blu Age Runtime 一起使用 AWS Fargate。

主题

先决条件

在开始之前，请确保满足以下先决条件：

- 填写 the section called “AWS Blu Age 运行时必” 和 the section called “AWS Blu Age 运行时入门”。
- 从相应源中获取以下依赖项。

Oracle 数据库

提供 Oracle 数据库驱动程序。例如，o jdbc8-19.8.0.0.jar。

IBM MQ 连接

提供一个 IBM MQ 客户端。例如，com.ibm.mq.allclient-9.3.0.15.jar。

使用此依赖项版本时，还要提供以下传递依赖项：

- javax.jms-api-2.0.1.jar
- json-20080701.jar

DDS 打印机文件

提供 Jasper 报告库。例如，jasperreports-6.16.0.jar，但更新的版本可能兼容。

使用此依赖项版本时，还要提供以下传递依赖项：

- castor-core-1.4.1.jar
- castor-xml-1.4.1.jar
- commons-digester-2.1.jar
- ecj-3.21.0.jar

- itext-2.1.7.js8.jar
- javax.inject-1.jar
- jcommon-1.0.23.jar
- jfreechart-1.0.19.jar

概述

要安装依赖项，请完成以下步骤。

1. 根据需要将上述任何依赖项复制到您的 Docker 镜像构建文件夹。
2. 如果您的 JICS 或 Blusam 数据库托管在 Oracle 上，请在中提供 Oracle 数据库驱动程序。*your-tomcat-path*/extra
3. 在你的 Dockerfile 上，将这些依赖项复制到。*your-tomcat-path*/extra
4. 构建 Docker 镜像，然后将其推送到亚马逊 ECR。
5. 停止并重启您的 Amazon ECS 服务。
6. 检查日志。

AWS 在 Amazon EC2 上部署 Blu Age 运行时

本节中的主题介绍如何在 Amazon EC2 上设置 AWS Blu Age Runtime（非托管）、如何更新运行时版本、如何使用 Amazon CloudWatch 警报监控您的部署以及如何添加许可依赖项。

主题

在 Amazon AWS EC2 上设置 Blu Age 运行时（非托管）

本主题介绍如何在 Amazon EC2 上使用 AWS Blu Age 运行时（非托管）设置和部署 PlanetsDemo 示例应用程序。

主题

- 设置
- 测试 PlanetsDemo 应用程序

先决条件

在开始之前，请确保满足以下先决条件：

- AWS CLI 按照配置 AWS CLI 中的步骤进行配置。
- 填写 the section called “AWS Blu Age 运行时必” 和 the section called “AWS Blu Age 运行时入门”。
- 创建包含最新 AWS Blu Age 运行时的 Amazon EC2 实例（在 Amazon EC2 上）。有关更多信息，请参阅 Amazon EC2 Linux 实例入门。
- 确保您可以成功连接到 Amazon EC2 实例，例如使用 SSM。
- 下载并解压 AWS Blu Age Runtime（在 Amazon EC2 上），网址为*your-tomcat-path*/*。有关说明，请参阅the section called “AWS Blu Age 运行时入门”。
- 下载PlanetsDemo应用程序档案。
- 解压缩存档文件，并将应用程序上传到您选择的 Amazon S3 存储桶。
- 为 JICS 创建一个 Amazon Aurora PostgreSQL 数据库，然后对其运行`PlanetsDemo-v1/jics/sql/initJics.sql`查询。有关如何创建 Amazon Aurora PostgreSQL 数据库的信息，请参阅创建和连接 Aurora Post greSQL 数据库集群。

设置

要设置 PlanetsDemo 示例应用程序，请完成以下步骤。

1. 连接到您的亚马逊 EC2 实例，然后转到 Apache Tomcat 9 安装`conf`文件夹下的文件夹。打开`catalina.properties`文件进行编辑，然后将以下行开头的`common.loader`行替换为下一行。

   ```
   common.loader="${catalina.base}/lib","${catalina.base}/lib/
   *.jar","${catalina.home}/lib","${catalina.home}/lib/*.jar","${catalina.home}/
   shared","${catalina.home}/shared/*.jar","${catalina.home}/extra","${catalina.home}/
   extra/*.jar"
   ```

2. 导航到 *<your-tomcat-path>*/webapps文件夹。
3. 使用以下命令 PlanetsDemo从 Amazon S3 存储桶中复制-v1/webapps/ 文件夹中可用的 PlanetsDemo 二进制文件。

```
aws s3 cp s3://path-to-demo-app-webapps/ . --recursive
```

> **Note**
>
> path-to-demo-app-webapps替换为之前解压缩存档的存储桶的正确 Amazon S3 URI。 PlanetsDemo

4. 将 PlanetsDemo-v1/config/ 文件夹的内容复制到 *<your-tomcat-path>*/config/。
5. 在application-main.yml文件中的以下代码段中，提供作为先决条件一部分创建的数据库的连接信息。有关更多信息，请参阅创建并连接到 Aurora PostgreSQL 数据库集群。

```
datasource:
   jicsDs:
     driver-class-name :
     url:
     username:
     password:
     type :
```

6. 启动你的 Apache Tomcat 服务器并验证日志。

```
your-tomcat-path/startup.sh

tail -f your-tomcat-path/logs/catalina.log
```

如果您发现以 C 后跟数字开头的错误代码（例如 CXXXX），请记下错误消息。例如，错误代码 C5102 是一个常见错误，表示基础架构配置不正确。

测试 PlanetsDemo 应用程序

要检查已部署 PlanetsDemo 应用程序的状态，请在替换load-balancer-DNS-namelistener-port、和之后web-binary-name使用正确的设置值运行以下命令。

```
curl http://load-balancer-DNS-name:listener-port/gapwalk-application/
```

如果应用程序正在运行，则会看到以下输出消息：Jics application is running。

接下来，运行以下命令。

```
curl http://load-balancer-DNS-name:listener-port/jac/api/services/rest/jicsservice/
```

如果应用程序正在运行，则会看到以下输出消息：`Jics application is running`。

```
Jics application is running
```

如果您已配置 Blusam，则运行以下命令时可能会出现空响应。

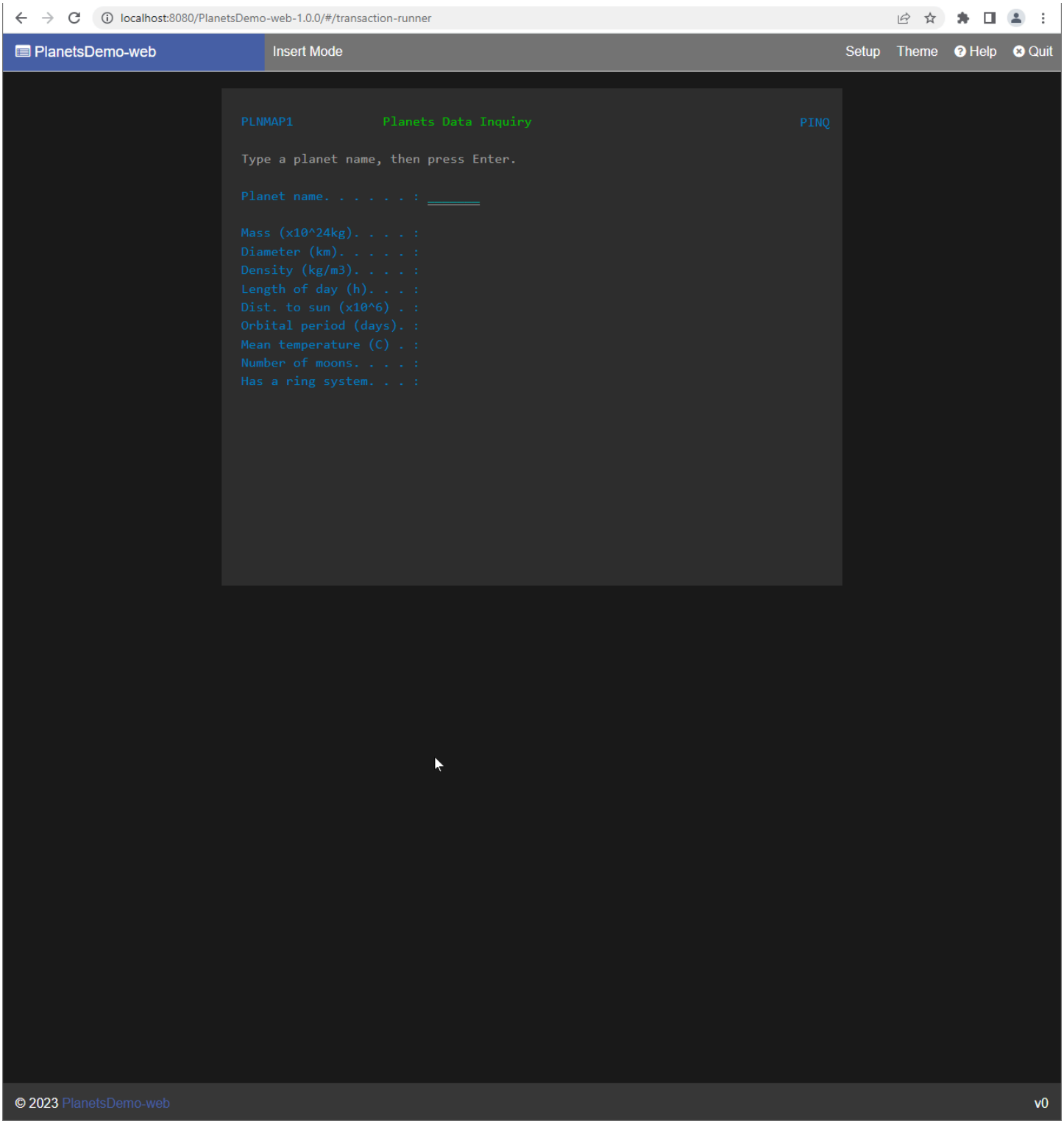
```
curl http://load-balancer-DNS-name:listener-port/bac/api/services/rest/bluesamserver/
serverIsUp
```

注意 Web 二进制文件的名称（PlanetsDemo-web-1.0.0，如果未更改）。要访问 PlanetsDemo 应用程序，请使用以下格式的 URL。

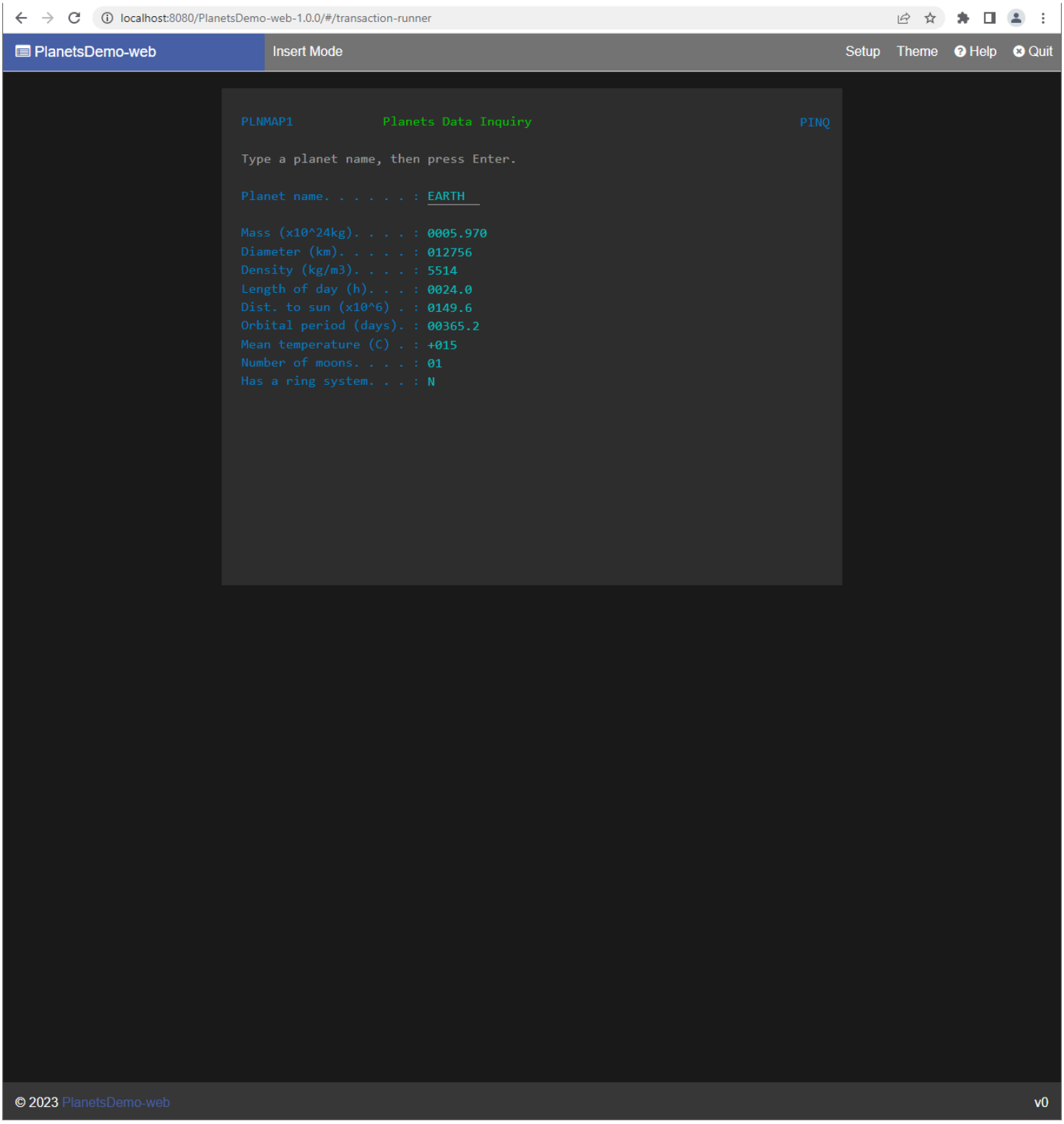
```
https://load-balancer-DNS-name:listener-port/web-binary-name
```

PlanetsDemo 应用程序启动后，将显示主页。

在文本框中，输入 PINQ，然后按 Enter。随后显示数据查询页面。



例如，在 PlanetsDemo 名称字段中输入 EARTH，然后按 Enter。随后将显示您输入的行星页面。



升级 Amazon EC2 上的 AWS Blu Age 运行时

本指南介绍如何在 Amazon EC2 上升级 AWS Blu Age 运行时。

主题

先决条件

在开始之前，确保满足以下先决条件：

- 填写 the section called “AWS Blu Age 运行时必” 和 the section called “AWS Blu Age 运行时入门”。
- 确保您拥有包含最新 AWS 蓝光时代运行时的 Amazon EC2 实例。有关更多信息，请参阅 Amazon EC2 Linux 实例入门。
- 确保您可以成功连接到 Amazon EC2 实例，例如，使用 SSM。
- 下载您想要升级到的 AWS Blu Age 运行时版本（在 Amazon EC2 上）。有关更多信息，请参阅 the section called “AWS Blu Age 运行时（非托管）设置”。该框架包含两个二进制文件：`aws-bluage-runtime-x.x.x.x.tar.gz` 和 `aws-bluage-webapps-x.x.x.x.tar.gz`。

概述

完成以下步骤来升级 Velocity 运行时。

1. 通过运行以下命令连接到您的 Amazon EC2 实例并将用户更改为 su。

   ```
   sudo su
   ```

   您需要超级用户权限才能运行本教程中的命令。
2. 创建两个文件夹，每个二进制文件一个文件夹。
3. 将每个文件夹命名为与相应二进制文件相同的名称。
4. 将每个二进制文件复制到相应的文件夹。

   > **Warning**
   >
   > 提取每个二进制文件会生成一个同名的文件夹。因此，如果将两个二进制文件一个接一个地提取到同一位置，则会覆盖其内容。

5. 使用以下命令提取二进制文件。运行每个文件夹中的命令。

```
tar xvf aws-bluage-runtime-x.x.x.x.tar.gz
tar xvf aws-bluage-webapps-x.x.x.x.tar.gz
```

6. 使用以下命令停止 Apache Tomcat 服务。

```
systemctl stop tomcat.service
systemctl stop tomcat-webapps.service
```

7. 将 <your-tomcat-path>/shared/ 的内容替换为 aws-bluage-runtime-x.x.x.x/velocity/shared/ 的内容。
8. 将 <your-tomcat-path>/webapps/gapwalk-application.war替换为 aws-bluage-runtime-x.x.x.x/velocity/webapps/gapwalk-application.war。
9. 将 <your-tomcat-path>/webapps/ 中的 war 文件（即 bac.war 和 jac.war）替换为 aws-bluage-webapps-x.x.x.x/velocity/webapps/ 中的相同文件。
10. 通过运行以下命令启动 Apache Tomcat 服务。

```
systemctl start tomcat.service
systemctl start tomcat-webapps.service
```

11. 检查日志。

运行以下命令来查看部署的应用程序的状态。

```
curl http://localhost:8080/gapwalk-application/
```

将会出现以下消息。

```
Jics application is running
```

```
curl http://localhost:8181/jac/api/services/rest/jicsservice/
```

将会出现以下消息。

```
Jics application is running
```

```
curl http://localhost:8181/bac/api/services/rest/bluesamserver/serverIsUp
```

响应应为空。

AWS Blu Age 运行时已成功升级。

AWS Blu Age Runtime（在亚马逊 EC2 上）Amazon Alarm CloudWatch s

为了在部署的应用程序遇到会使您的应用程序处于宽限期的异常时获得更明显的通知，您可以设置 CloudWatch为接收应用程序日志，并添加警报以警告您可能出现的错误。

部署 CloudWatch 日志记录

默认情况下，`application-main.yml` 文件包含对另一个名为 `logback-cloudwatch.yml` 的日志记录配置文件的引用。

```
logging:
  config: classpath:logback-cloudwatch.xml
```

两个文件都在 config 文件夹中，这就是配置 CloudWatch 日志的方式，如以下各节所述。

CloudWatch 日志配置

默认的 `logback-cloudwatch.xml` 文件具有以下内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration>
<configuration>

    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%date{yyyy-MM-dd HH:mm:ss.SSS,UTC}  %level --- [%thread{15}]
 %logger{40} : %msg%n%xThrowable</pattern>
        </encoder>
    </appender>

    <appender name="cloudwatch"
 class="com.netfective.bluage.runtime.cloudwatchlogger.CloudWatchAppender">
        <logGroup>BluAgeRuntimeOnEC2-Logs</logGroup>
        <logStream>%date{yyyy-MM-dd,UTC}.%instanceId.%uuid</logStream>
        <layout>
            <pattern>%date{yyyy-MM-dd HH:mm:ss.SSS,UTC}  %level --- [%thread{15}]
 %logger{40} : %msg%n%xThrowable</pattern>
        </layout>
```

```
        <appender-ref ref="console" />
    </appender>

    <root level="INFO">
        <appender-ref ref="cloudwatch" />
    </root>
</configuration>
```

<appender name="cloudwatch"/> 元素之外的所有内容均为标准 logback 配置。此文件中有两个附加程序：用于向控制台发送日志的控制台附加程序和用于向其发送日志的 CloudWatch 附加程序。CloudWatch

`root` 元素中的 `level` 属性指定了整个应用程序的日志记录级别。

标签内的必填值<appender name="cloudwatch"/>为：

- <logGroup/>:在中设置日志组的 CloudWatch名称。如果未指定，则默认值为 `BluAgeRuntimeOnEC2-Logs`。如果日志组不存在，系统将自动创建。可以通过配置来更改此行为，具体内容将在下文中介绍。
- <logStream/>：在中设置日志流（在日志组内）的名称。 CloudWatch

可选值：

- <region/>：覆盖将写入日志流的区域。默认情况下，日志会被写入 EC2 实例所在区域。
- <layout/>：日志消息将使用的模式。
- <maxbatchsize/>：每个操作要发送到的最大日志消息 CloudWatch 数。
- <maxbatchtimemillis/>：允许写入 CloudWatch 日志的时间（以毫秒为单位）。
- <maxqueuewaittimemillis/>：尝试在内部日志队列中插入请求的时间（以毫秒为单位）。
- <internalqueuesize/>：内部队列的最大大小。
- <createlogdests/>：创建日志组和日志流（如果不存在）。
- <initialwaittimemillis/>: 您希望线程在启动时处于睡眠状态的时间。该初始等待时间允许日志的初始累积。
- <maxeventmessagesize/>：日志事件的最大大小。超过此大小的日志将不会被发送。
- <truncateeventmessages/>：截断过长的消息。
- <printrejectedevents/>：启用紧急 Appender。

CloudWatch 设置

为了使上述配置能够正确推送日志 CloudWatch，请更新您的 Amazon EC2 IAM 实例配置文件角色，以授予其对 “BluAgeRuntimeOnec2-Logs” 日志组及其日志流的额外权限：

- `logs:CreateLogStream`
- `logs:DescribeLogStreams`
- `logs:CreateLogGroup`
- `logs:PutLogEvents`
- `logs:DescribeLogGroups`

警报设置

借助 CloudWatch 日志，您可以根据应用程序和需求配置不同的指标和警报。具体而言，您可以为使用情况设置主动警报，以便在出现可能使您的应用程序处于宽限期（最终使其完全无法运行）的错误时收到警告。为此，您可以在日志中添加与 “Error C5001” 字符串相关的指标，该指标会突出显示与 AWS Blu Age 控制系统连接中的错误。然后，您可以定义一个对此指标做出反应的警报。

在 Amazon EC2 的 AWS Blu Age 运行时中设置许可的依赖关系

本指南介绍如何在 Amazon EC2 上设置可与 AWS Blu Age Runtime 一起使用的其他许可依赖项。

主题

先决条件

在开始之前，请确保满足以下先决条件：

- 填写 the section called “AWS Blu Age 运行时必” 和 the section called “AWS Blu Age 运行时入门”。
- 确保您有一个包含最新 AWS Blu Age 运行时的 Amazon EC2 实例（在 Amazon EC2 上）。有关更多信息，请参阅 Amazon EC2 Linux 实例入门。
- 确保您可以成功连接到 Amazon EC2 实例，例如，使用 SSM。
- 从其来源获取以下依赖关系。

Oracle 数据库

提供 Oracle 数据库驱动程序。我们使用版本 ojdbc8-19.8.0.0.jar 测试了 B AWS lu Age Runtime（在 Amazon EC2 上）功能，但可能兼容更新的版本。

IBM MQ 连接

提供一个 IBM MQ 客户端。我们使用版本 com.ibm.mq.allclient-9.3.0.15.jar 测试了 BI AWS u Age Runtime（在亚马逊 EC2 上）功能，但可能兼容更新的版本。

使用此依赖项版本时，还要提供以下传递依赖项：

- javax.jms-api-2.0.1.jar
- json-20080701.jar

DDS 打印机文件

提供 Jasper 报告库。我们使用 j asperreports-6.16.0.jar 测试了 AWS Blu Age Runtime（在亚马逊 EC2 上）功能，但可能兼容更新的版本。

使用此依赖项版本时，还要提供以下传递依赖项：

- castor-core-1.4.1.jar
- castor-xml-1.4.1.jar
- commons-digester-2.1.jar
- ecj-3.21.0.jar
- itext-2.1.7.js8.jar
- javax.inject-1.jar
- jcommon-1.0.23.jar
- jfreechart-1.0.19.jar

概述

要安装依赖项，请完成以下步骤。

1. 通过运行以下命令连接到您的 Amazon EC2 实例并将用户更改为 su。

```
sudo su
```

您需要超级用户权限才能运行本教程中的命令。

2. 导航到 <your-tomcat-path>/extra/文件夹。

```
cd <your-tomcat-path>/extra/
```

3. 根据需要将上述任何依赖项复制到此文件夹。
4. 通过运行以下命令停止和启动 tomcat.service。

```
systemctl stop tomcat.service
```

```
systemctl start tomcat.service
```

5. 检查该服务的状态，确保其正在运行。

```
systemctl status tomcat.service
```

6. 验证日志。

为 JAC 和 BAC webapps 设置依赖项

1. 如果您的 JICS 或 Blusam 数据库托管在 Oracle 上，则需要在 <your-tomcat-path>/extra 中提供 Oracle 数据库驱动程序。
2. 如果该文件夹不存在，请创建该文件夹。
3. 停止并重新启动你的 Apache Tomcat 服务器。
4. 验证日志。

使用 Blu Age Developer IDE 修改源代码

如果您使用的是 AWS托管的 AWS Blu Age 运行时引擎，则可以使用 Blu Age Developer 来修改生成的源代码。在以下情况下可能需要修改源代码：出于某种原因需要更新现代化代码，或者部分旧源代码无法进行现代化。你可以通过 Amazon AppStream 2.0 访问 Blu Age 开发者。本节介绍如何在 AppStream 2.0 上设置 Blu Age Developer。它还说明了如何使用示例应用程序使用 Blu Age Developer 更新源代码 PlanetsDemo。

主题

- 教程：为 AWS Blu Age 开发者 IDE 设置 AppStream 2.0

- 教程：在 AppStream 2.0 上使用 AWS Blu Age 开发者

教程：为 AWS Blu Age 开发者 IDE 设置 AppStream 2.0

AWS 大型机现代化通过 Amazon AppStream 2.0 提供了多种工具。 AppStream 2.0 是一项完全托管的安全应用程序流服务，允许您在不重写应用程序的情况下将桌面应用程序流式传输给用户。AppStream 2.0 使用户能够即时访问他们需要的应用程序，并在他们选择的设备上提供响应灵敏、流畅的用户体验。使用 AppStream 2.0 托管特定于运行时引擎的工具，使客户应用程序团队能够直接从其 Web 浏览器使用这些工具，与存储在 Amazon S3 存储桶或存储库中的应用程序文件进行交互。CodeCommit

有关 AppStream 2.0 中浏览器支持的信息，请参阅《Amazon AppStream 2.0 管理指南》中的 “系统要求和功能支持（Web 浏览器）”。如果您在使用 AppStream 2.0 时遇到问题，请参阅 Amazon AppStream 2.0 管理指南中的 AppStream 2.0 用户问题疑难解答。

本文档介绍如何在 AppStream 2.0 舰队上设置 AWS Blu Age 开发者 IDE。

主题

先决条件

下载存档文件，其中包含在 AppStream 2.0 下设置 AWS Blu Age Developer IDE 所需的工件。

> **Note**
>
> 这是一个大文件。如果您遇到操作超时问题，我们建议您使用 Amazon EC2 实例来提高上传和下载性能。

步骤 1：创建 Amazon S3 存储桶

创建与您将要创建的 AppStream 2.0 队列 AWS 区域 相同的 Amazon S3 存储桶。该存储桶将包含完成本教程所需的构件。

步骤 2：将策略附加到 S3 存储桶

将以下策略附加到您为本教程创建的存储桶。请务必将 MYBUCKET 替换为您创建的存储桶的实际名称。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "AllowAppStream2.0ToRetrieveObjects",
        "Effect": "Allow",
        "Principal": {
            "Service": "appstream.amazonaws.com"
        },
        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::MYBUCKET/*"
    }]
}
```

步骤 3：将文件上传到 Amazon S3 存储桶

解压缩您在先决条件中下载的文件，然后将该 appstream 文件夹上传到您的存储桶。上传此文件夹会在您的存储桶中创建正确的结构。有关更多信息，请参阅《Amazon S3 用户指南》中的上传对象。

第 4 步：下载 AWS CloudFormation 模板

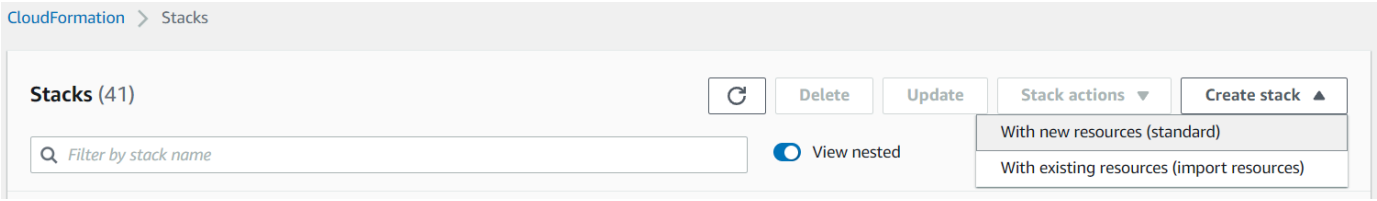
下载以下 AWS CloudFormation 模板。您需要这些模板来创建和填充 AppStream 2.0 队列。

- cfn-m2-.yaml appstream-elastic-fleet-linux
- cfn-m2--linux.yaml appstream-bluage-dev-tools
- cfn-m2-.yaml appstream-bluage-shared-linux
- cfn-m2-.yaml appstream-chrome-linux
- cfn-m2-.yaml appstream-eclipse-jee-linux
- cfn-m2-.yaml appstream-pgadmin-linux

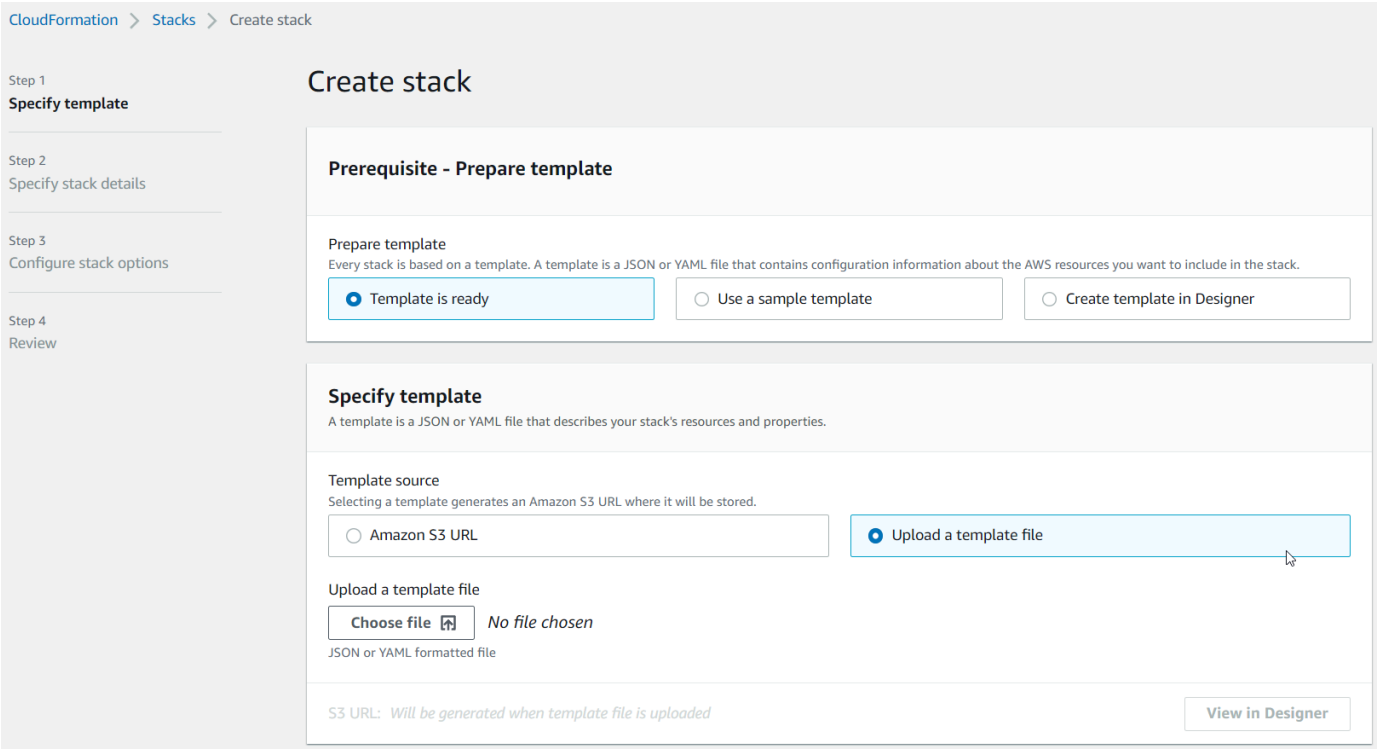
第 5 步：使用创建舰队 AWS CloudFormation

在此步骤中，您将使用`cfn-m2-appstream-elastic-fleet-linux.yaml` AWS CloudFormation 模板创建 AppStream 2.0 舰队和堆栈来托管 AWS Blu Age Developer IDE。创建队列和堆栈后，您将运行在上一步中下载的其他 AWS CloudFormation 模板来安装开发者 IDE 和其他必需的工具。

1. 在 AWS 管理控制台 AWS CloudFormation 中导航到，然后选择堆栈。
2. 在堆栈中，选择创建堆栈和使用新资源(标准)。



3. 在创建堆栈中，选择模板准备就绪和上传模板文件：



4. 选择选择文件，并导航到文件 `cfn-m2-appstream-elastic-fleet-linux.yaml`。选择下一步。
5. 在指定堆栈集详细信息页面上，提供以下信息：
   - 堆栈的名称。
   - 您的默认安全组和该安全组的两个子网。

> **Note**
>
> 安全组的两个子网必须位于不同的可用区中。

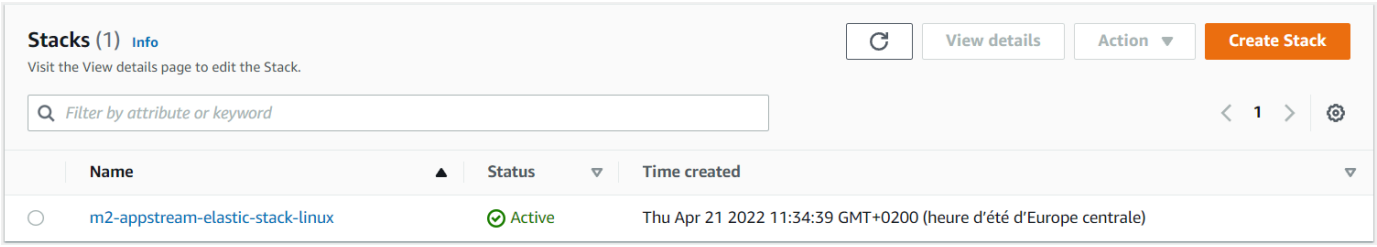
6. 选择下一步，然后再次选择下一步。
7. 选择我确认 AWS CloudFormation 可能会使用自定义名称创建 IAM 资源。 ，然后选择 “提交”。
8. 创建队列后，使用其他下载的模板创建 CloudFormation 堆栈以完成应用程序的设置。确保BucketName每次都更新以指向正确的 S3 存储桶。你可以在 CloudFormation控制台BucketName中编辑。或者，您可以直接编辑模板文件并更新 `S3Bucket` 属性。

> **Note**
>
> 下载的模板预期使用名为 `appstream/bluage/developer-ide/` 的文件夹结构在 S3 存储桶中查找资产。存储桶必须与您创建的队列 AWS 区域 相同。

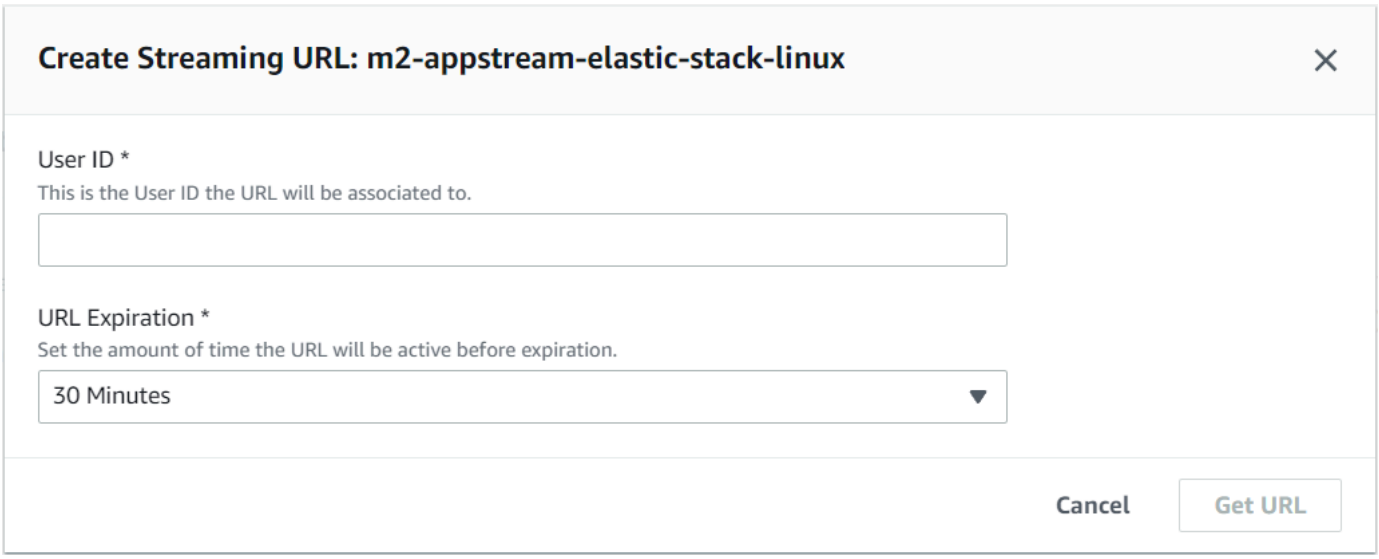
步骤 6：访问实例

创建并启动实例集后，您可以创建一个临时链接，以便通过本机客户端访问实例集。

1. 在中导航到 AppStream 2.0， AWS Management Console 然后选择之前创建的堆栈：



2. 在堆栈详细信息页面上，依次选择操作和创建流式传输 URL：



3. 在创建流式传输 URL 中，输入任意“用户 ID”和“URL 过期时间”，然后选择获取 URL。您获取的 URL 可用于流式传输到浏览器或本机客户端。我们建议您流式传输到本机客户端。

清理资源

有关清理已创建堆栈和队列的过程，请参阅创建 AppStream 2.0 队列和堆栈。

删除 AppStream 2.0 对象后，您或账户管理员还可以清理 S3 存储桶中的应用程序设置和主文件夹。

> **Note**
>
> 给定用户的主文件夹在所有队列中都是唯一的，因此，如果同一个账户中的其他 AppStream 2.0 堆栈处于活动状态，则可能需要保留该文件夹。

您无法使用 AppStream 2.0 控制台删除用户。而是必须使用服务 API 和 AWS CLI进行删除。有关更多信息，请参阅 Amazon AppStream 2.0 管理指南中的用户池管理。

教程：在 AppStream 2.0 上使用 AWS Blu Age 开发者

本教程向您展示了如何在 AppStream 2.0 上访问 AWS Blu Age Developer 并将其与示例应用程序一起使用，以便您可以试用这些功能。完成本教程后，您可以在使用自己的应用程序的情况下，使用相同的步骤进行操作。

主题

步骤 1：创建数据库

在此步骤中，使用 Amazon RDS 创建托管 PostgreSQL 数据库，演示应用程序使用该数据库来存储配置信息。

1. 打开 Amazon RDS 控制台。
2. 选择数据库 > 创建数据库。
3. 选择标准创建 > PostgreSQL，保留默认版本，然后选择免费套餐。
4. 选择数据库实例标识符。
5. 对于凭证设置，选择管理 AWS Secrets Manager 中的主凭证。有关更多信息，请参阅《Amazon RDS 用户指南》中的使用 Amazon RDS 和 AWS Secrets Manager 管理密码。
6. 确保 VPC 与您用于 AppStream 2.0 实例的 VPC 相同。您可以向管理员询问此值。
7. 对于 VPC 安全组，选择新建。
8. 将公共访问设置为是。
9. 保留所有其他默认值。审核这些值。
10. 选择创建数据库。

要使数据库服务器可以从您的实例访问，请在 Amazon RDS 中选择数据库服务器。在连接和安全性下，为数据库服务器选择 VPC 安全组。此安全组是之前创建的，其描述应与 RDS 管理控制台创建的安全组的描述类似。选择操作 > 编辑入站规则，选择添加规则，然后创建 PostgreSQL 类型的规则。对于规则来源，请使用安全组默认值。您可以开始在来源字段中输入来源名称，然后接受建议的 ID。最后，选择保存规则。

步骤 2：访问环境

在此步骤中，您将在 AppStream 2.0 上访问 AWS Blu Age 开发环境。

1. 请联系您的管理员以获取访问您的 AppStream 2.0 实例的正确方法。有关可能的客户端和配置的一般信息，请参阅 Amazon AppStream 2.0 管理指南中的 AppStream 2.0 访问方法和客户端。请考虑使用本机客户端以获得最佳体验。
2. 在 AppStream 2.0 中选择 “桌面”。

步骤 3：设置运行时

在此步骤中，设置 AWS Blu Age 运行时。您必须在首次启动时设置运行时，如果收到运行时升级的通知，则必须重新设置运行时。此步骤将填充您的 `.m2` 文件夹。

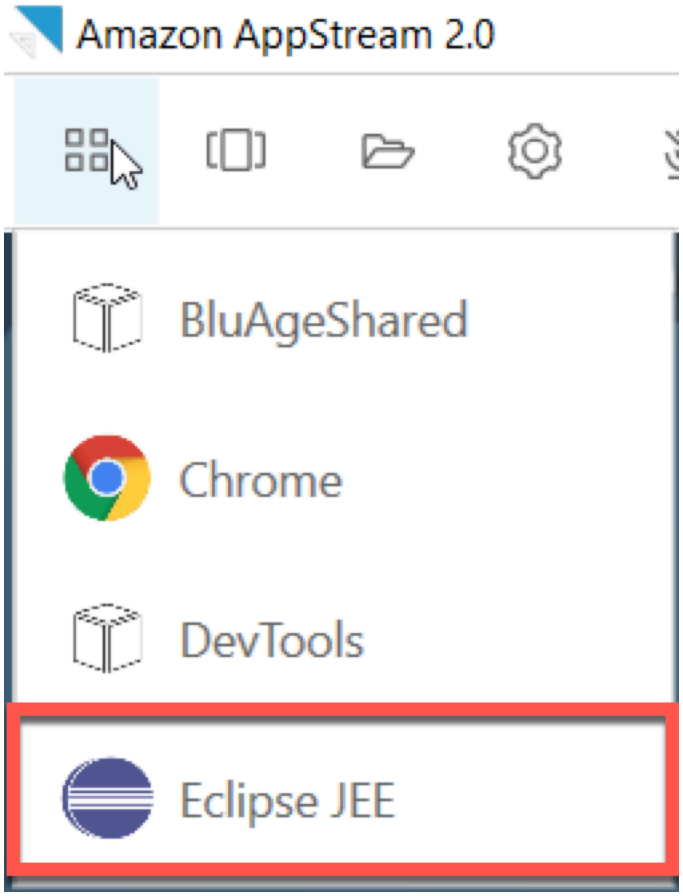
1. 从菜单栏中选择应用程序，然后选择终端。
2. 输入以下 命令：

```
~/_install-velocity-runtime.sh
```

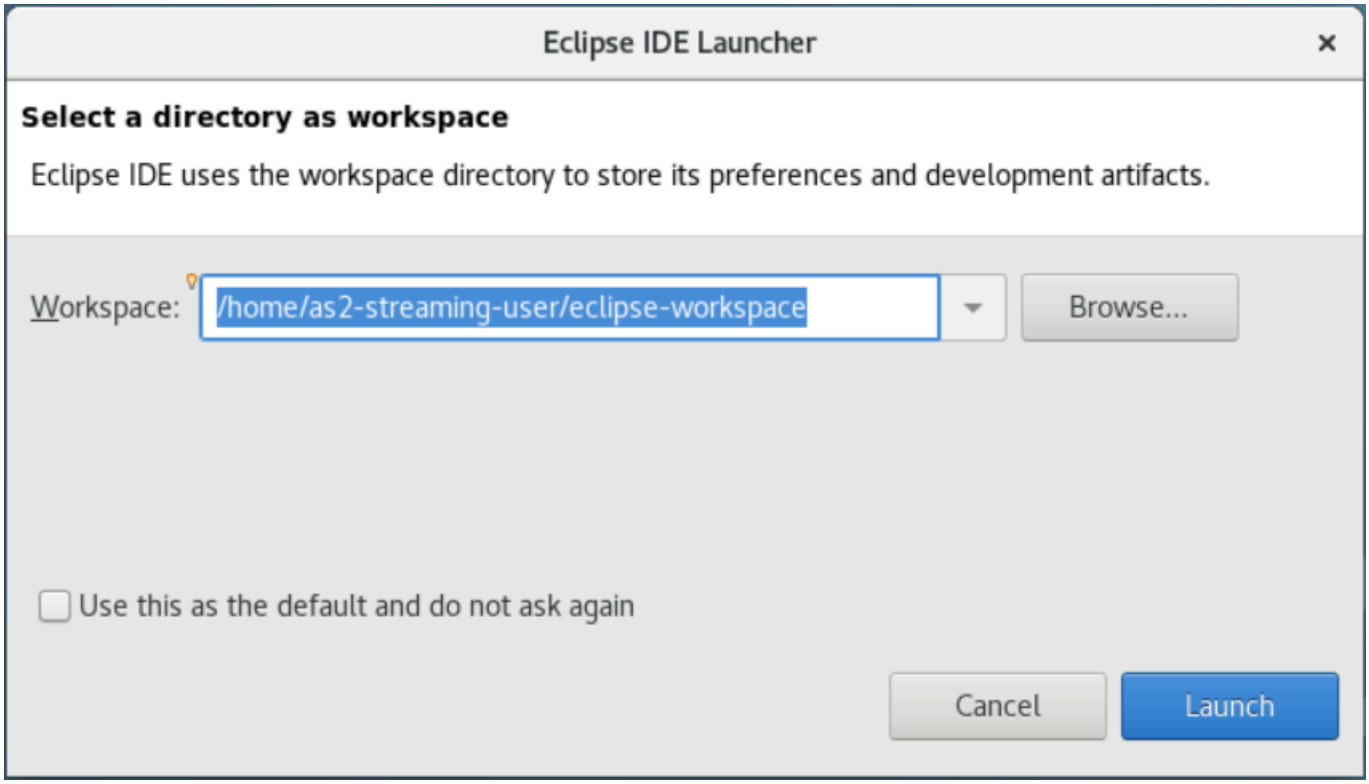
步骤 4：启动 Eclipse IDE

在此步骤中，启动 Eclipse IDE 并选择要创建工作区的位置。

1. 在 AppStream 2.0 中，选择工具栏上的 “启动应用程序” 图标，然后选择 Eclipse JEE。



2. 启动器打开后，输入要创建工作区的位置，然后选择启动。



您也可以选择通过命令行启动 Eclipse，如下所示：

```
~/eclipse &
```

步骤 5：设置 Maven 项目

在此步骤中，为 Planets 演示应用程序导入 Maven 项目。

1. PlanetsDemo将-pom.zip 上传到你的主文件夹。您可以使用本机客户端“我的文件”功能来执行此操作。
2. 使用 `unzip` 命令行工具提取文件。
3. 在解压缩后的文件夹中导航，然后在文本编辑器中打开项目的根 `pom.xml`。
4. 编辑 `gapwalk.version` 属性，使其与已安装的 AWS Blu Age 运行时相匹配。

   如果不确定安装的版本，请在终端中发出以下命令：

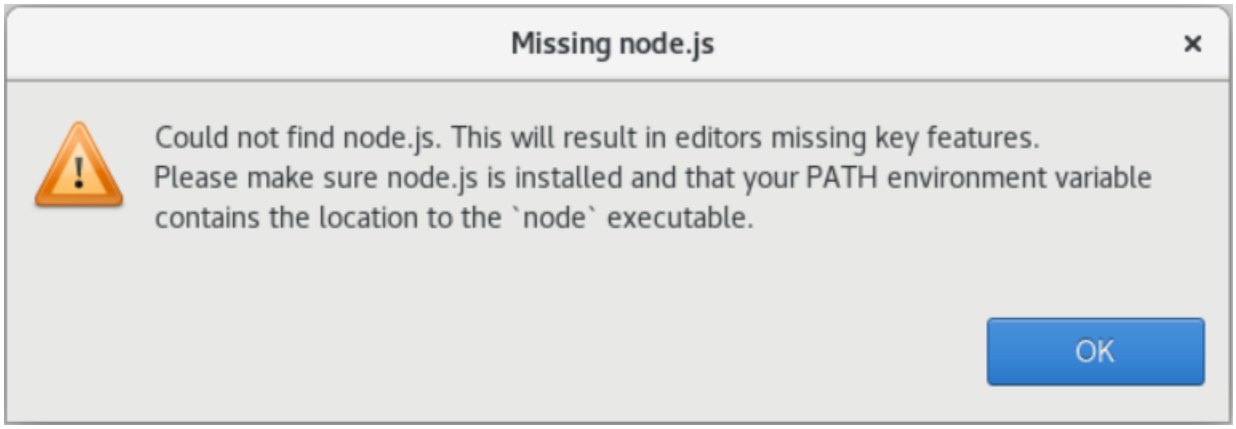
   ```
   cat ~/runtime-version.txt
   ```

   此命令打印当前可用的运行时版本，例如 `3.1.0-b3257-dev`。

> **Note**
>
> 请勿在 gapwalk.version 中包含 -dev 后缀。例如，有效的值可以是 <gapwalk.version>3.1.0-b3257</gapwalk.version>。

5. 在 Eclipse 中，选择文件，然后选择导入。在导入对话框窗口中，展开 Maven，然后选择现有的 Maven 项目。选择下一步。
6. 在导入 Maven 项目中，提供提取的文件的位置，然后选择完成。

   您可以放心地忽略以下弹出窗口。Maven 下载 node.js 本地副本来构建项目的 Angular (*-web) 部分：

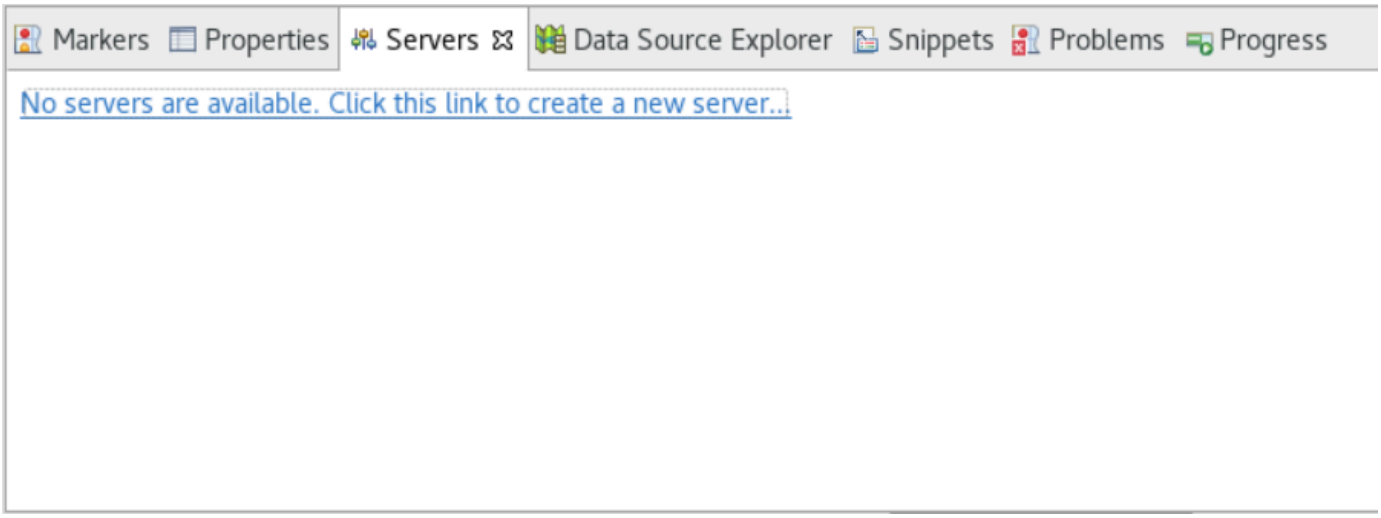


   等到构建结束。您可以在进度视图中查看构建进度。
7. 在 Eclipse 中，选择项目并选择运行方式。然后选择 安装。Maven 安装成功后，会在 PlanetsDemoPom/PlanetsDemo-web/target/PlanetsDemo-web-1.0.0.war 下创建 war 文件。

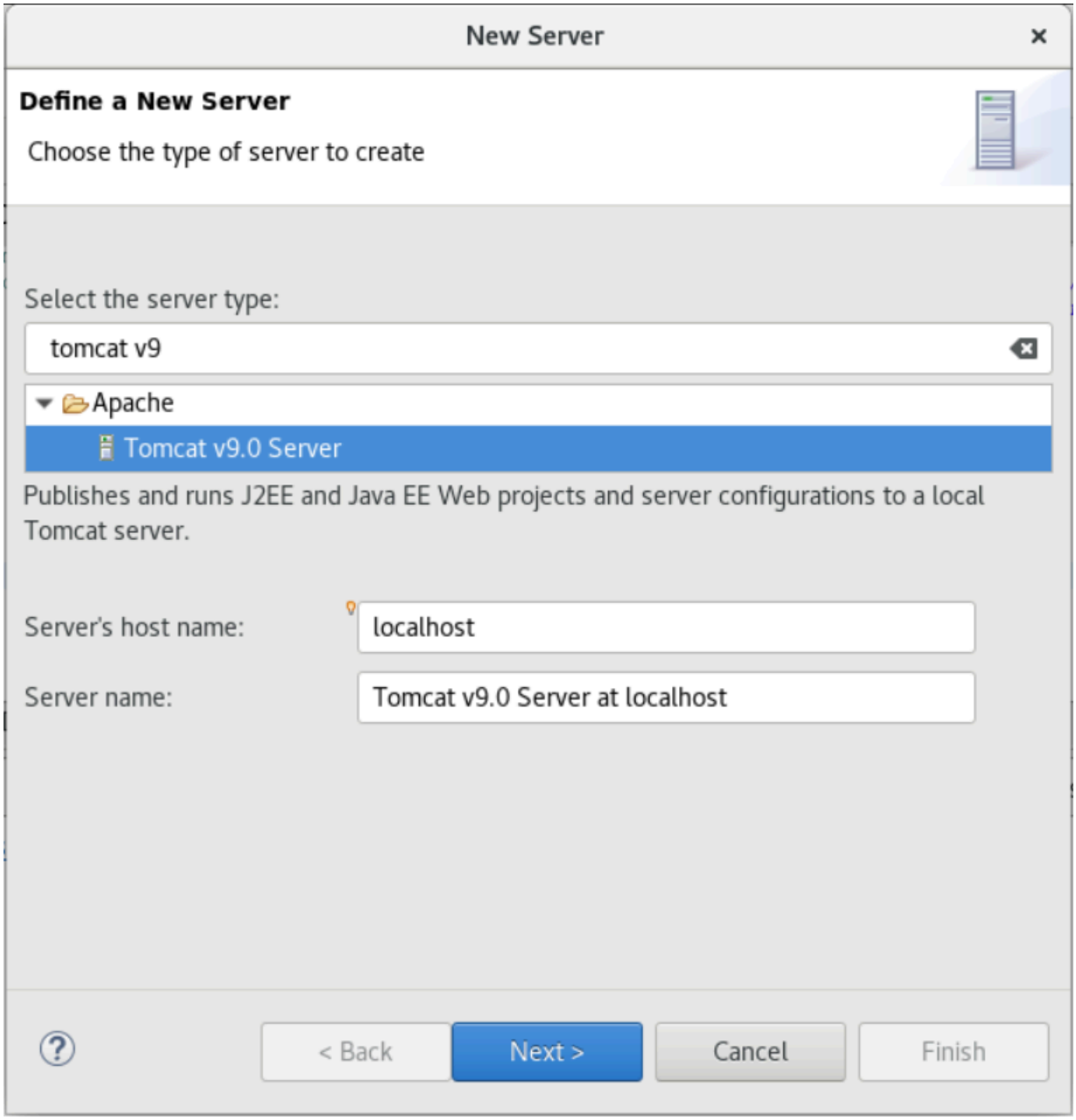
步骤 6：配置 Tomcat 服务器

在此步骤中，配置一个 Tomcat 服务器，用于部署和启动已编译的应用程序。

1. 在 Eclipse 中，选择窗口 > 显示视图 > 服务器以显示服务器视图：

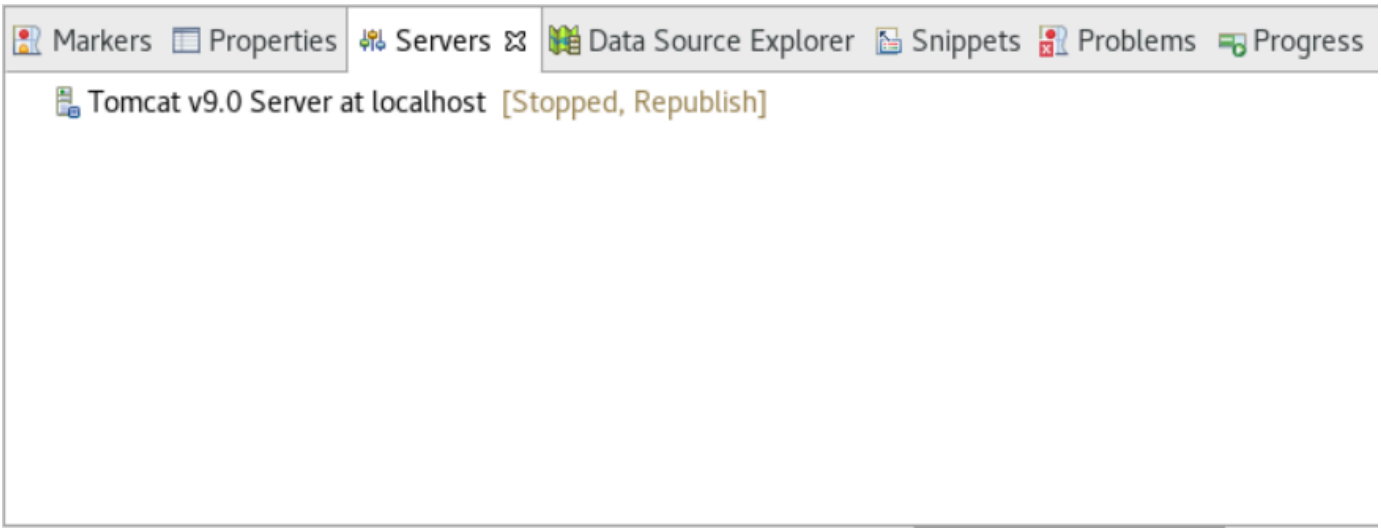


2. 选择没有可用的服务器。请单击此链接创建新服务器...。随后将出现新建服务器向导。在向导的选择服务器类型字段中，输入 tomcat v9，并选择 Tomcat v9.0 服务器。然后选择下一步。



3. 选择浏览，然后选择主文件夹根目录下的 tomcat 文件夹。保持 JRE 的默认值，然后选择完成。

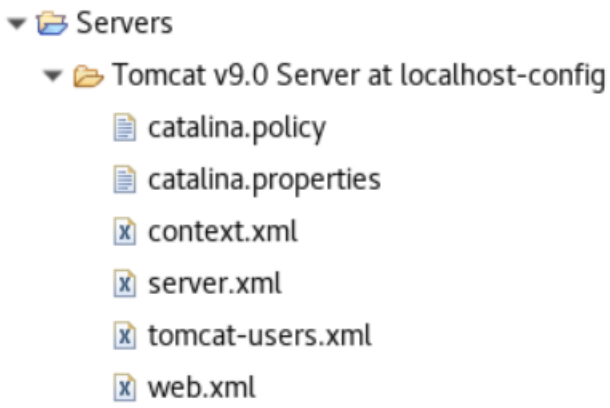
   服务器项目已在工作区中创建，Tomcat v9.0 服务器现在在服务器视图中可用。已编译应用程序将在此处部署和启动：



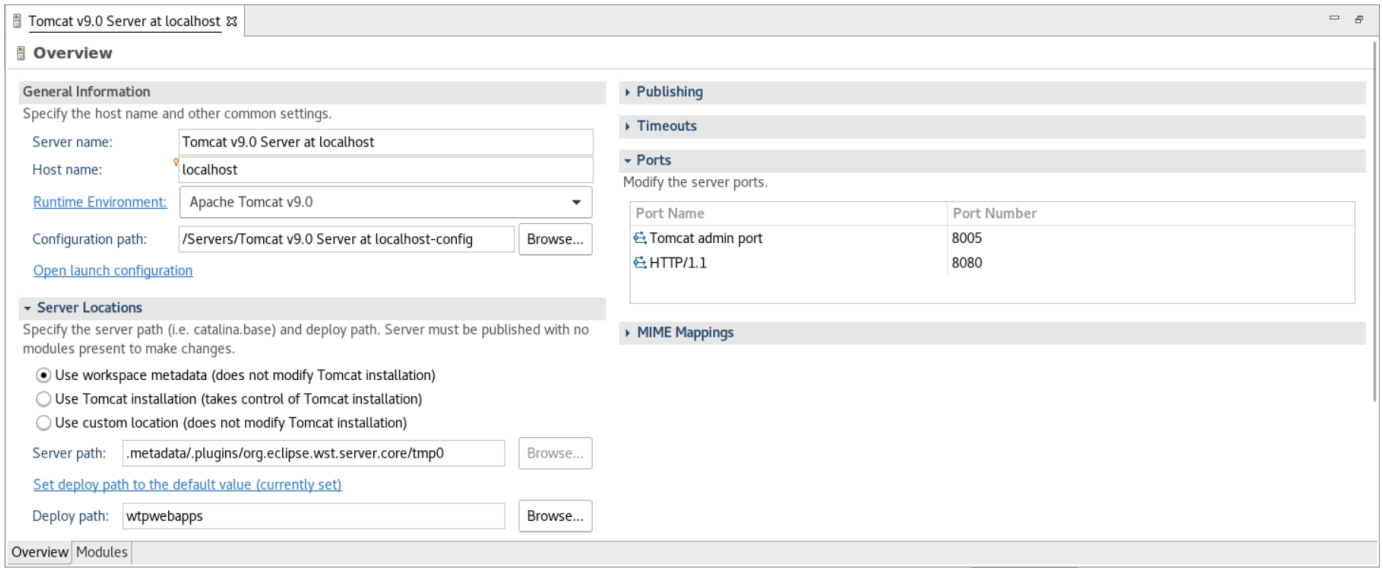
步骤 7：部署到 Tomcat

在此步骤中，将 Planets 演示应用程序部署到 Tomcat服务器，以便运行该应用程序。

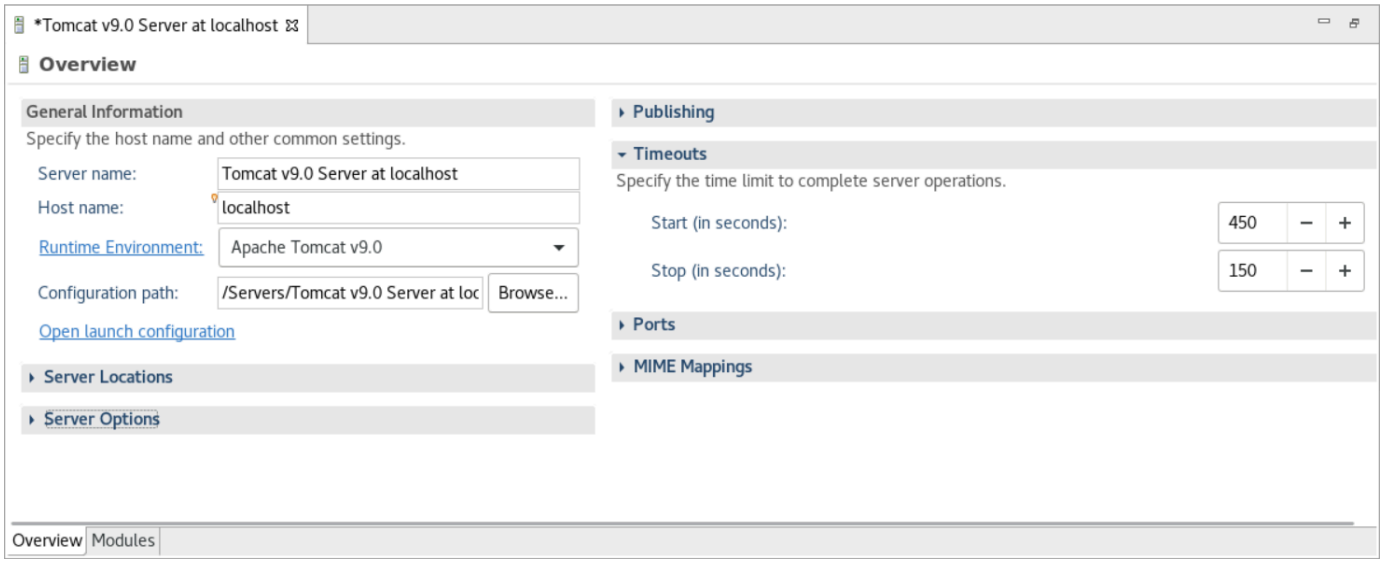
1. 选择 `PlanetsDemo-web` 文件并选择运行方式 > Maven 安装。再次选择 `PlanetsDemo-web` 并选择刷新，确保将经 npm 编译的前端正确编译为 .war 并被 Eclipse 注意到。
2. 将 PlanetsDemo-runtime.zip 上传到实例，然后将文件解压缩到可访问的位置。这样可以确保演示应用程序可以访问所需的配置文件夹和文件。
3. 将 `PlanetsDemo-runtime/tomcat-config` 的内容复制到您为 Tomcat 服务器创建的 `Servers/Tomcat v9.0...` 子文件夹中：



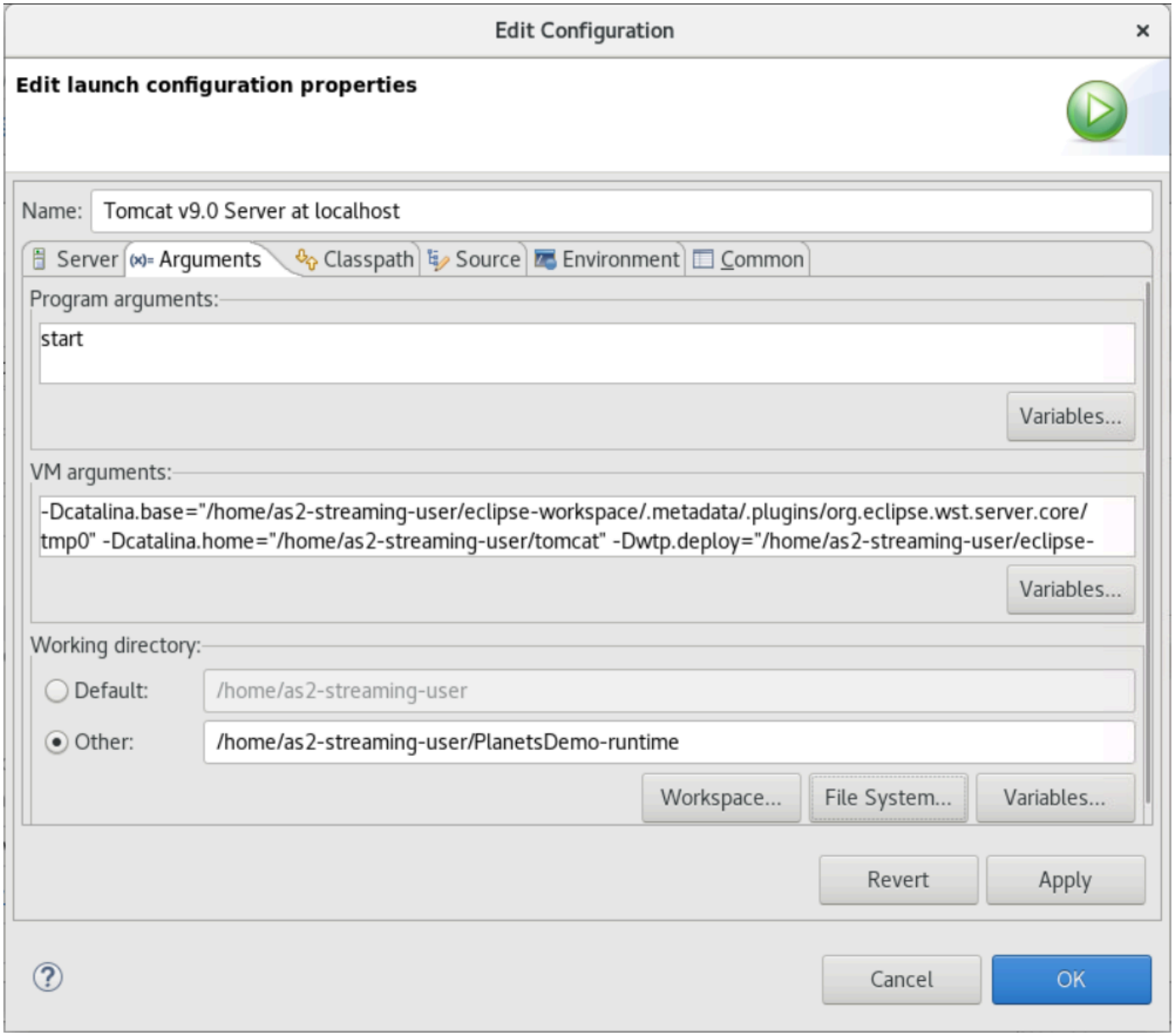
4. 在“服务器”视图中打开 `tomcat v9.0` 服务器条目。此时会显示服务器属性编辑器：



5. 在概览选项卡中，将“启动”的超时值增加到 450 秒，将“停止”的“超时”值增加到 150 秒，如下所示：



6. 选择打开启动配置。此时将显示向导。在向导中，导航到参数文件夹，然后在工作目录中选择其他。选择文件系统，然后导航到之前解压缩的 `PlanetsDemo-runtime` 文件夹。此文件夹应包含一个名为 config 的直接子文件夹。

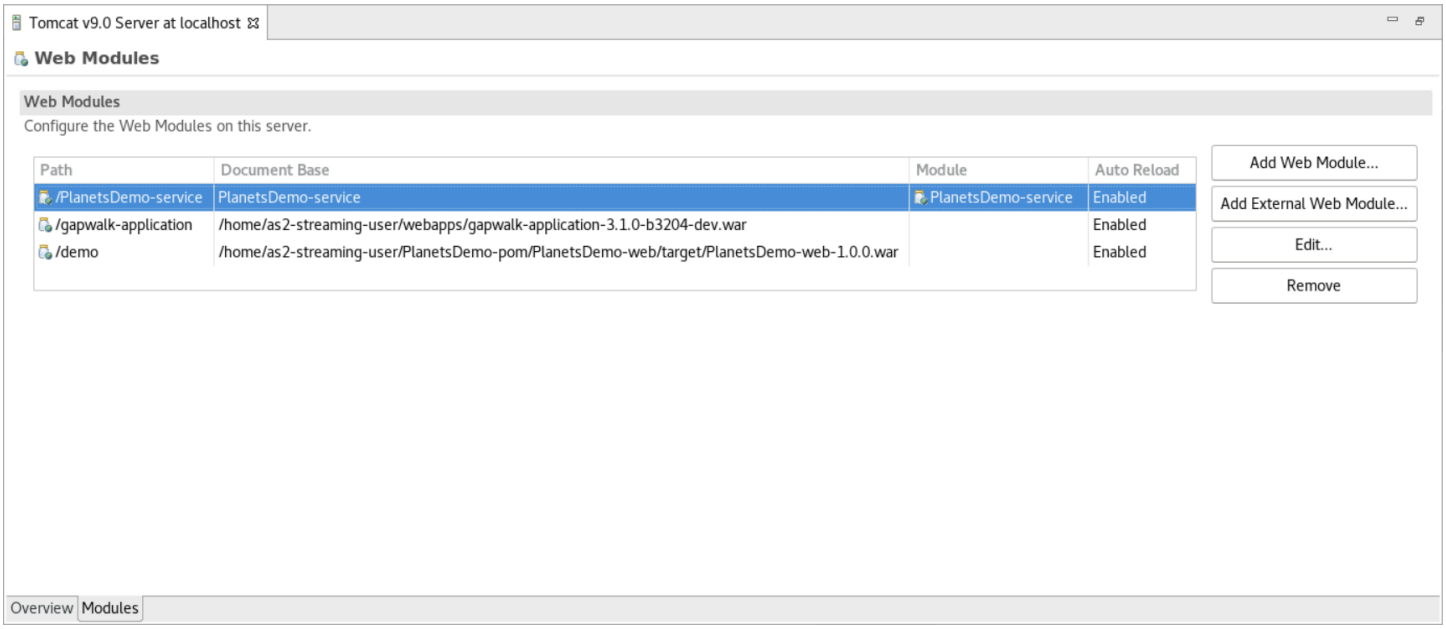


7. 选择服务器属性编辑器的模块选项卡并进行以下更改：

   - 选择添加 Web 模块并添加 `PlanetsDemo-service`。
   - 选择添加外部 Web 模块。随后显示添加 Web 模块对话框窗口。进行以下更改：
     - 在文档库中，选择浏览并导航至 `~/webapps/gapwalk-application...war`
     - 在路径中，输入 `/gapwalk-application`。
   - 选择“确定”。
   - 再次选择添加外部 Web 模块并进行以下更改：
     - 对于文档库，输入前端 .war 的路径（在 `PlanetsDemo-web/target` 中）

  - 在路径，输入 /demo
- 选择“确定”。
- 保存编辑器修改（Ctrl + S）。

编辑器内容应类似于以下示例所示。



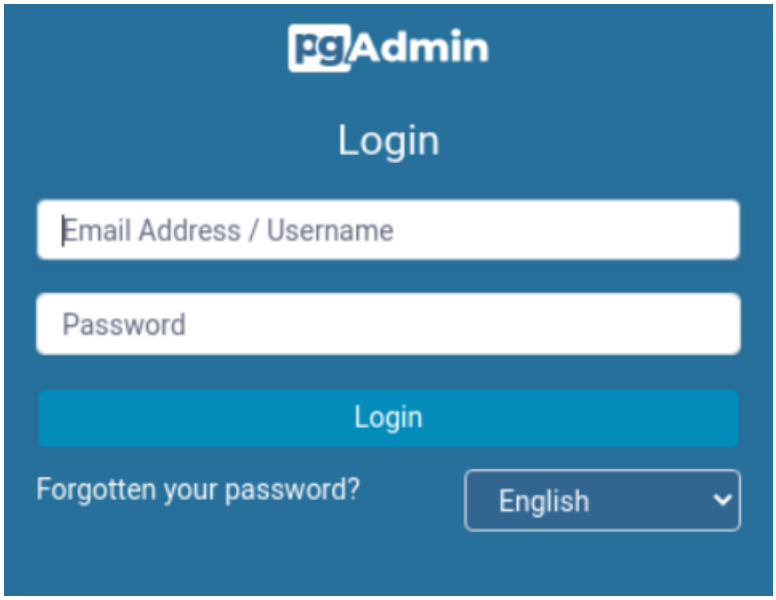
步骤 8：创建 JICS 数据库

在此步骤中，连接到您在 步骤 1：创建数据库 中创建的数据库。

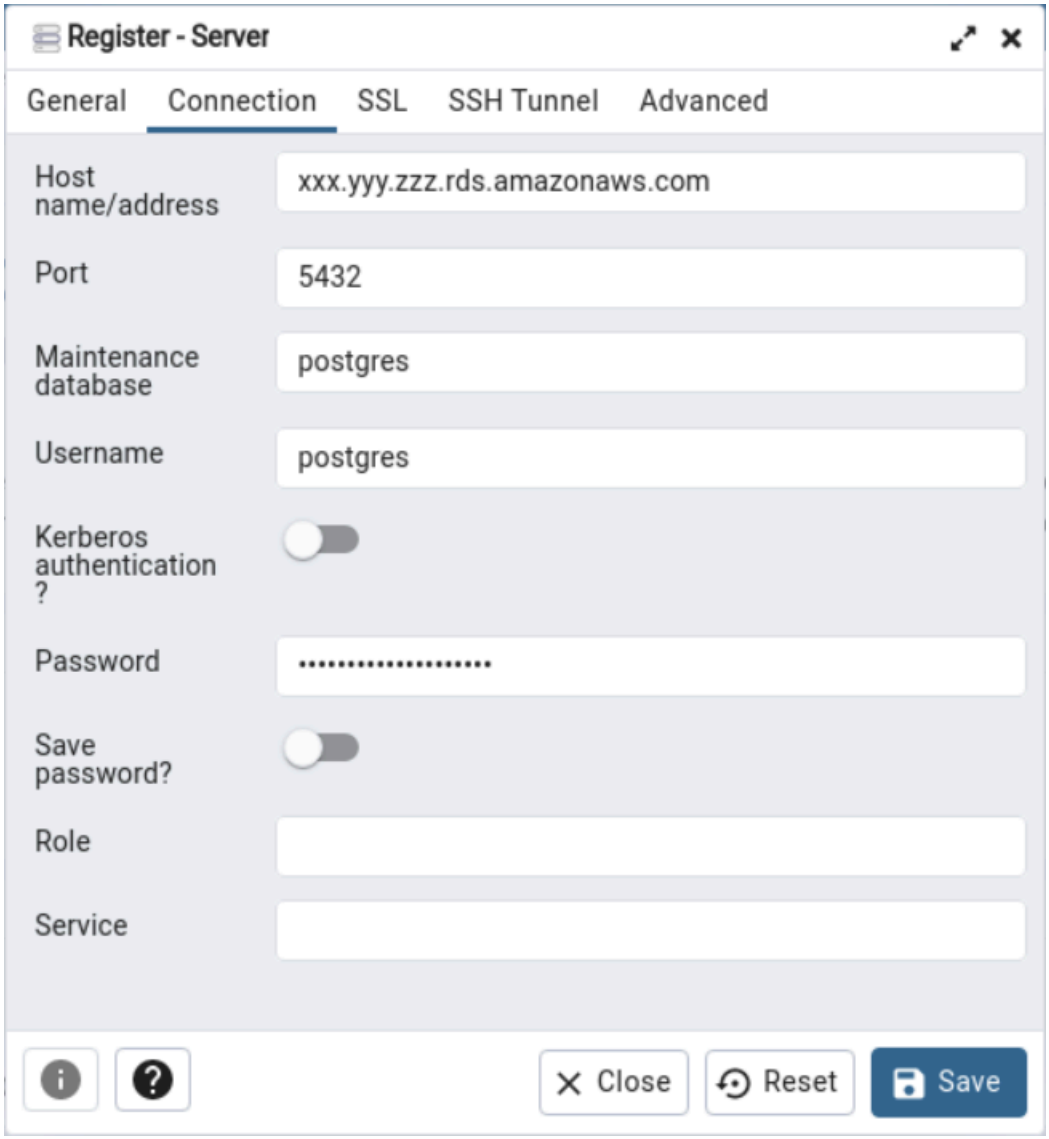
1. 在 AppStream 2.0 实例中，在终端中发出以下命令进行启动pgAdmin：

```
./pgadmin-start.sh
```

2. 选择电子邮件地址和密码作为登录标识符。记录提供的 URL（通常是 http://127.0.0.1:5050）。在实例中启动 Google Chrome，将此 URL 复制并粘贴到浏览器中，然后使用您的标识符登录。



3. 登录后，选择添加新服务器，然后按如下方式输入先前创建的数据库的连接信息。



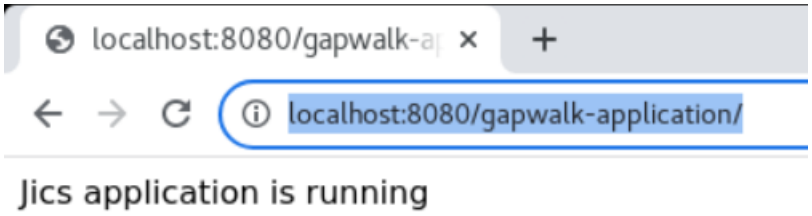
4. 连接到数据库服务器时，使用对象 > 创建 > 数据库并创建一个名为 jics 的新数据库。
5. 编辑演示应用程序使用的数据库连接信息。此信息在 `PlanetsDemo-runtime/config/application-main.yml` 中定义。搜索 `jicsDs` 条目。要检索 `username` 和 `password` 的值，请在 Amazon RDS 控制台中导航到数据库。在配置选项卡的主凭证 ARN 下，选择在 Secrets Manager 中管理。然后，在 Secrets Manager 控制台中，在密钥中选择检索密钥值。

步骤 9：启动和测试应用程序

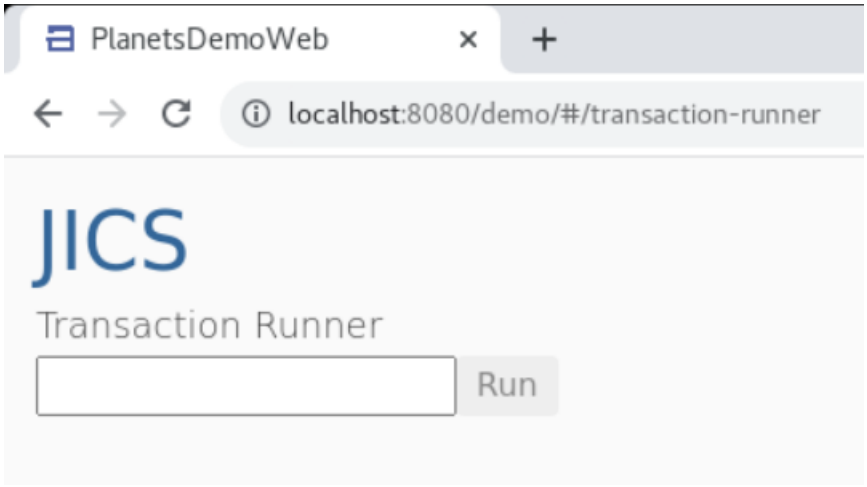
在此步骤中，启动 Tomcat 服务器和演示应用程序，以便对其进行测试。

1. 要启动 Tomcat 服务器和先前部署的应用程序，请在“服务器”视图中选择服务器条目，然后选择启动。随后将显示控制台，控制台上显示启动日志。

2. 在“服务器”视图中查看服务器状态，或者在控制台中等待服务器在 [xxx] 毫秒后启动消息。服务器启动后，检查 gapwalk-application 是否已正确部署。为此，请在 Google Chrome 浏览器中访问 http://localhost:8080/gapwalk-application URL。您应该看到以下内容。



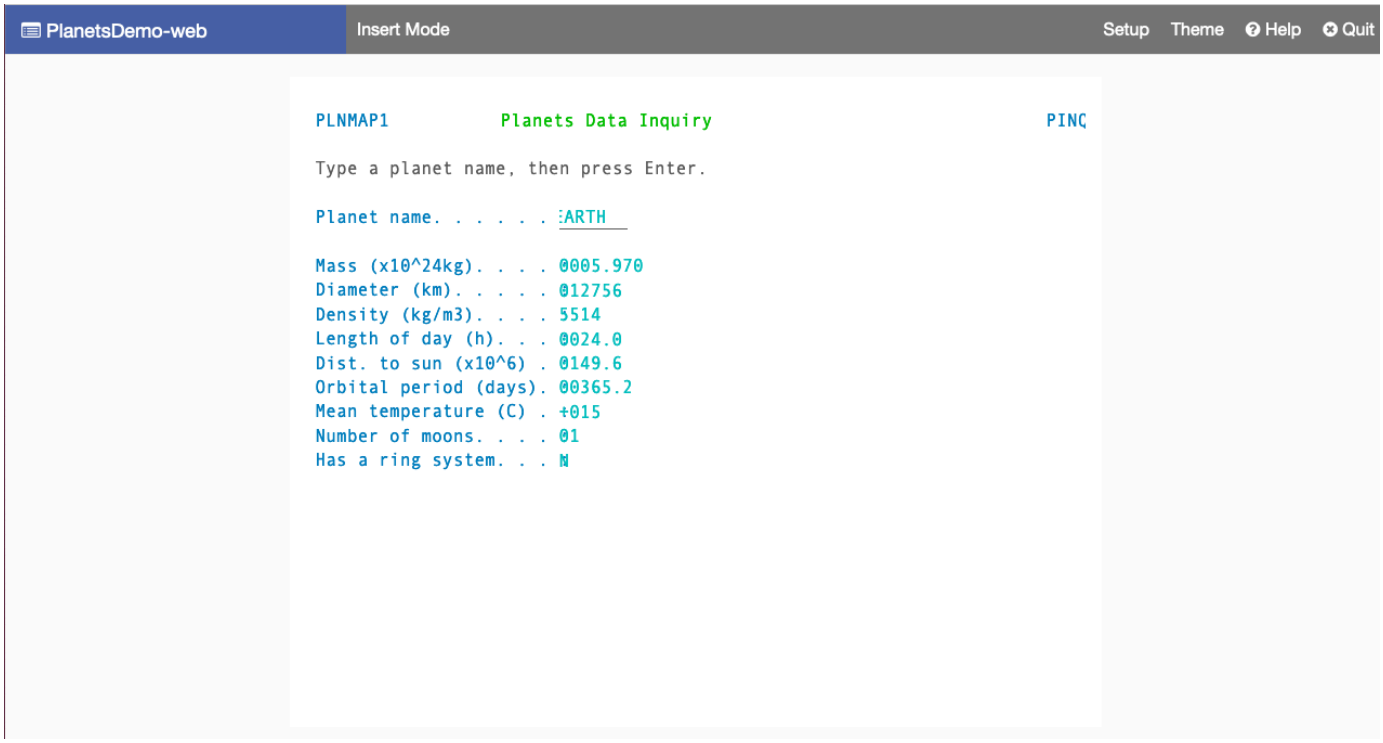
3. 从 Google Chrome 访问已部署的应用程序前端，网址为 http://localhost:8080/demo。随后会出现以下事务启动器页面。



4. 要启动应用程序事务，请在输入字段中输入 PINQ，然后选择运行（或按 Enter）。

   随后会显示演示应用程序屏幕。

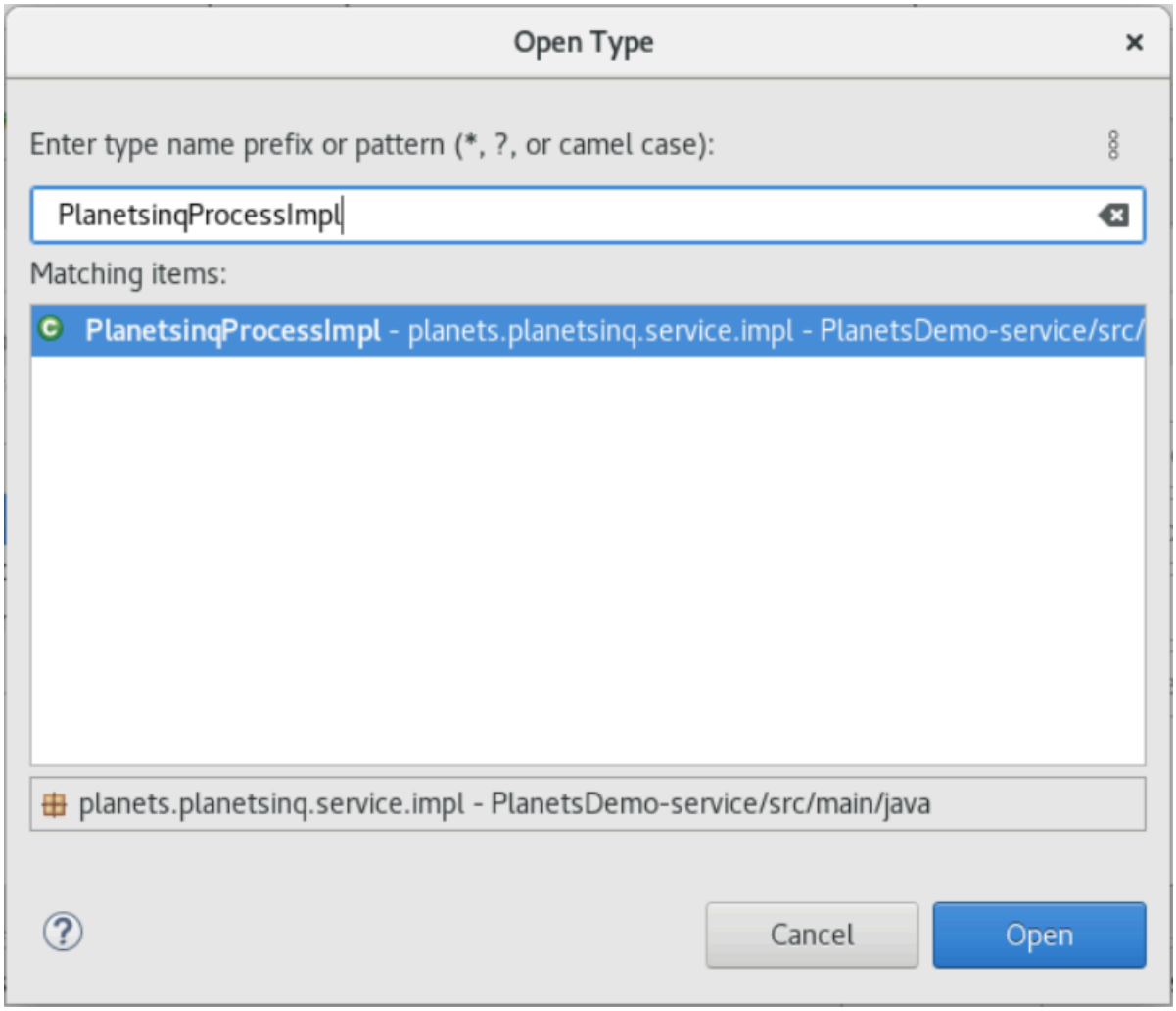
5. 在相应的字段中键入行星名称，然后按 Enter。



步骤 10：部署应用程序

在此步骤中，使用标准 Eclipse 调试功能进行测试。当您使用现代化应用程序时，这些功能可用。

1. 要打开主服务类，请按 Ctrl + Shift + T。然后输入 `PlanetsinqProcessImpl`。



2. 导航到 `searchPlanet` 方法，并在此处放置一个断点。
3. 选择服务器名称并选择在调试中重新启动。
4. 重复前述步骤，即访问应用程序，输入行星名称，然后按 Enter。

   Eclipse 将在 `searchPlanet` 方法中停止该应用程序。现在，您可以进行检查。

清理资源

如果您不再需要为本教程创建的资源，请将其删除，以免产生额外费用。完成以下步骤：

- 如果 Planets 应用程序仍在运行，请将其停止。
- 删除您在 步骤 1：创建数据库 中创建的数据库。有关更多信息，请参阅删除数据库实例。

使用 Micro Focus 为应用程序更换平台

本节介绍更换平台过程中的每个步骤，它描述了所有任务，并包括有关在 Amazon EC2 上配置和操作 AWS 大型机现代化运行时的信息。

主题

Micro Focus 运行时（在 Amazon EC2 上）设置

AWS 大型机现代化提供了多个亚马逊系统映像 (AMI)，其中包括 Micro Focus 许可产品。借助这些 AMI，您可以快速配置 Amazon Elastic Compute Cloud (Amazon EC2) 实例来支持您控制和管理的 Micro Focus 环境。本主题提供访问和启动这些 AMI 所需的步骤。使用这些 AMI 是完全可选的，不是完成本用户指南中的教程所必需的。

主题

- 订阅亚马逊机器映像
- 启动 AWS 大型机现代化 Micro Focus 实例
- 无法访问 Internet 的子网或 VPC
- 对许可证问题进行故障排除

先决条件

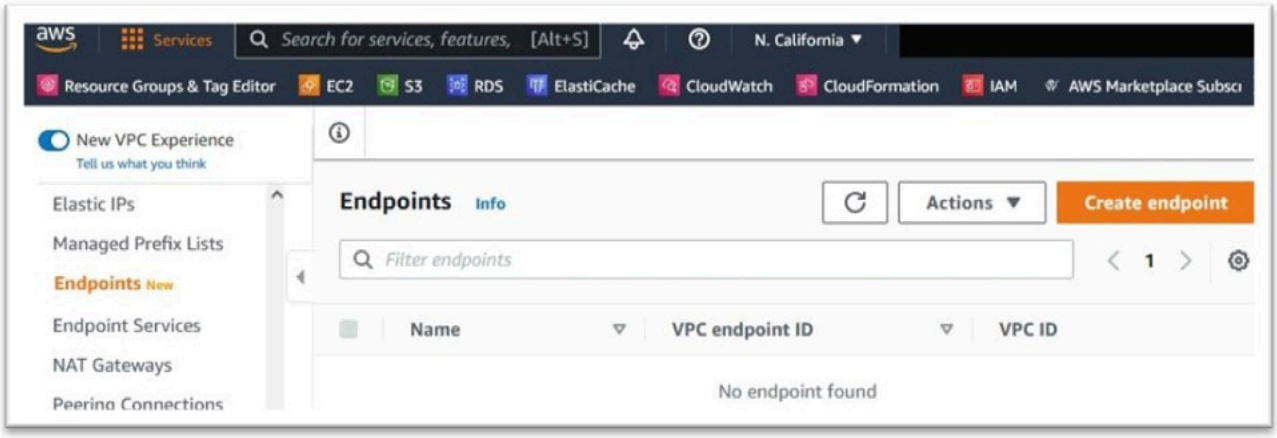
请确保满足以下先决条件：

- 对要在其中创建 Amazon EC2 实例的账户具有管理员访问权限。
- 确定 Amazon EC2 实例将在 AWS 区域 何处创建，并验证 AWS 大型机现代化服务是否可用。请参阅按区域划分的AWS 服务。请务必选择可提供该服务的区域。
- 确定要在其中创建 Amazon EC2 实例的 Amazon Virtual Private Cloud (Amazon VPC)。

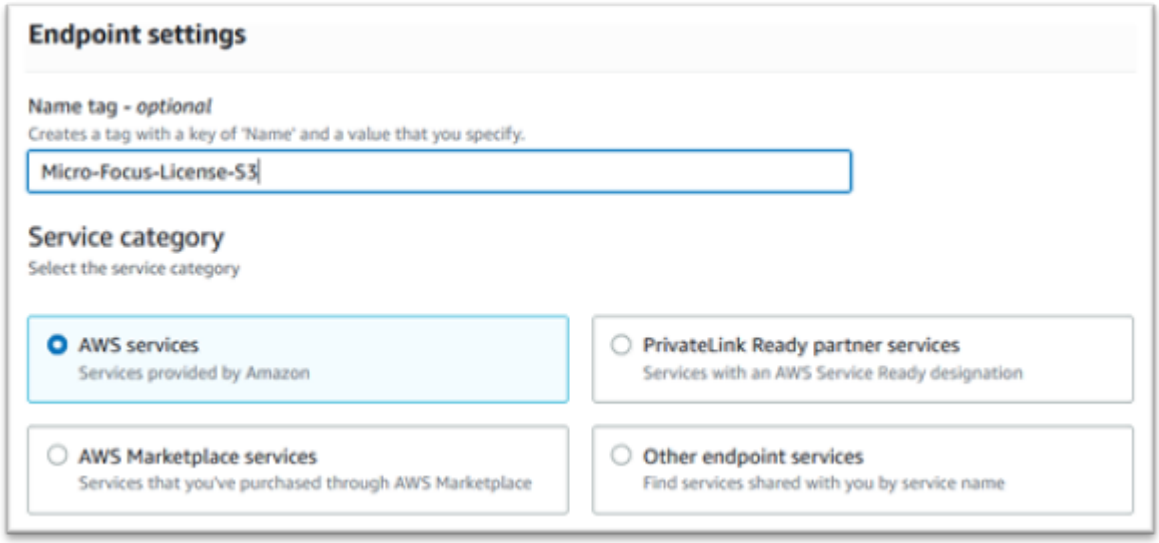
为 Amazon S3 创建 Amazon VPC 端点：

在本节中，创建一个供 Amazon S3 使用的 Amazon VPC 端点。

1. 在 AWS Management Console中导航到 Amazon VPC。
2. 在导航窗格中，选择端点。
3. 选择 创建端点。



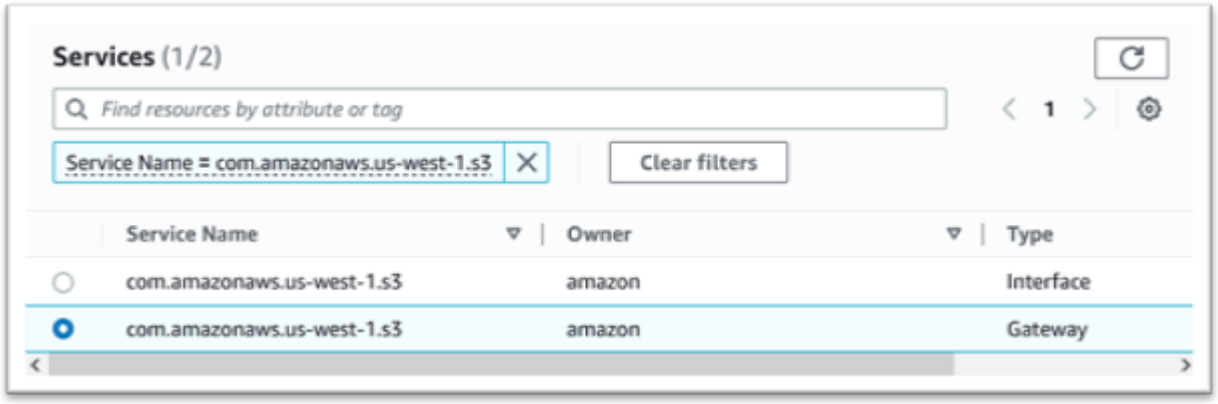
4. 输入一个有意义的名称标签，例如：“Micro-Focus-License-S3”。
5. 对于“服务类别”，选择 AWS 服务。



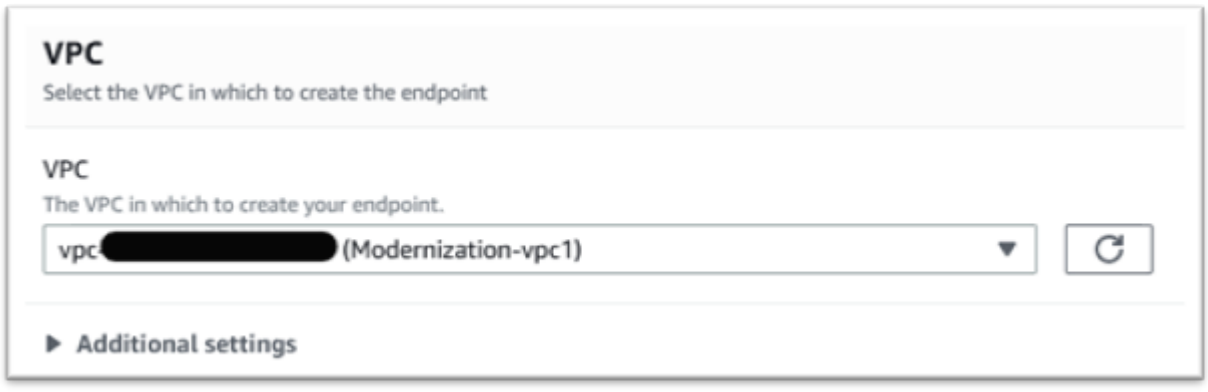
6. 在服务下搜索 Amazon S3 网关服务：com.amazonaws.[region].s3。

   对于 `us-west-1`，请使用：`com.amazonaws.us-west-1.s3`。

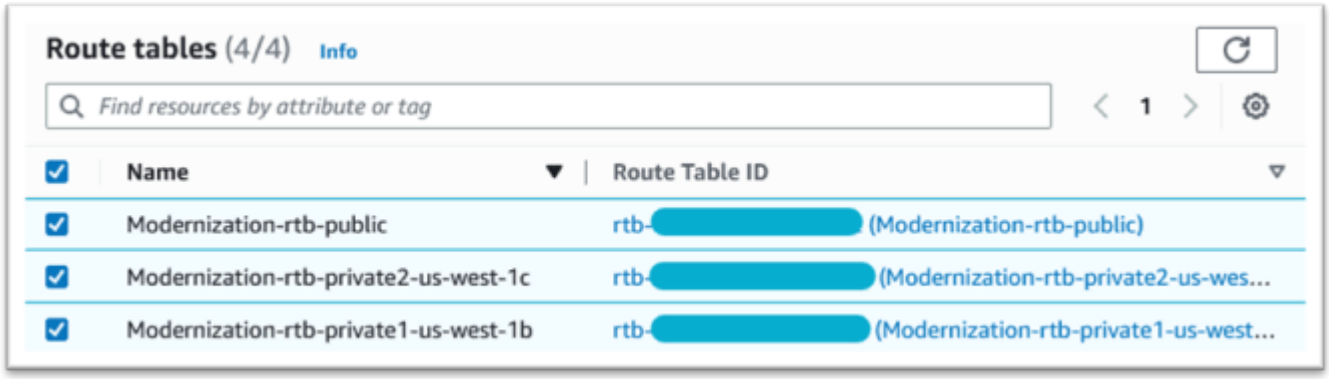
7. 选择网关服务。



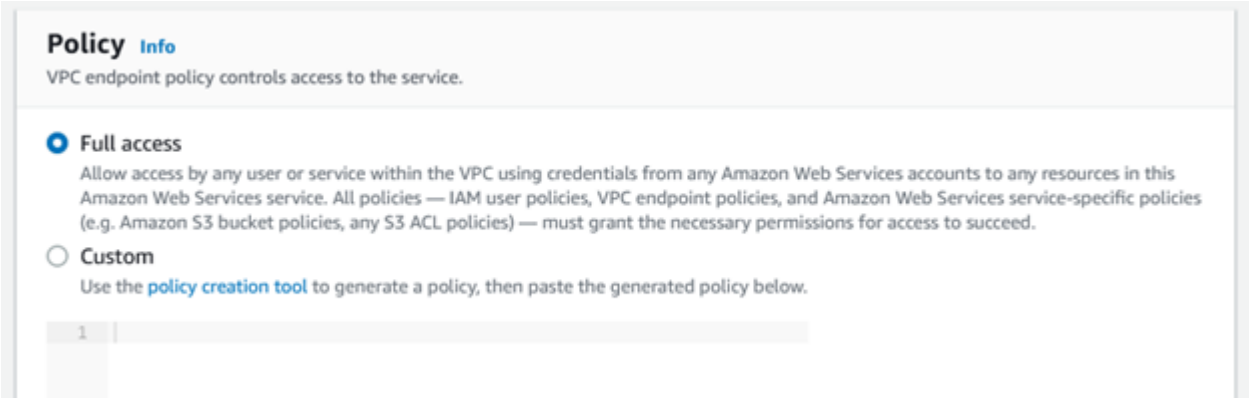
8. 对于 VPC，请选择您要使用的 VPC。



9. 选择 VPC 的所有路由表。



10. 在策略下，选择完全访问权限。



11. 选择创建端点。

申请更新账户的允许列表

请与您的 AWS 代表合作，将您的帐户列入 AWS 大型机现代化 AMI 的许可名单。并提供以下信息：

- AWS 账户 身份证。
- 创建 Amazon VPC 终端节点 AWS 区域 的位置。
- 在为 Amazon S3 创建 Amazon VPC 端点： 中创建的 Amazon VPC Amazon S3 端点 ID。此处是为 com.amazonaws.[region].s3 Gateway 端点创建的 vpce-xxxxxxxxxxxxxxxxx ID。
- 所有 Micro Focus Enterprise Suite AMI Amazon EC2 实例所需的许可证数量。

  每个 CPU 内核（对于大多数 Amazon EC2 实例，每 2 个 vCPU）需要一个许可证。

  有关更多信息，请参阅优化 CPU 选项。

  将来，可以通过以下方式调整所请求的号码 AWS。

> **Note**
>
> AWS 代表必须为许可名单请求打开支持票。无法直接申请，并且申请可能需要几天时间才能完成。

创建 AWS Identity and Access Management 角色

创建供 AWS 大型机现代化 Amazon EC2 实例使用的 AWS Identity and Access Management 策略和角色。通过 IAM 控制台创建角色会创建具有相同名称的关联实例配置文件。将此实例配置文件分配给 Amazon EC2 实例允许分配 Micro Focus 许可证。有关实例配置文件的更多信息，请参阅使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

创建 IAM 策略

首先创建一个 IAM 策略并将其附加到角色。

1. 导航 AWS Identity and Access Management 到 AWS Management Console。
2. 选择策略，然后选择创建策略。

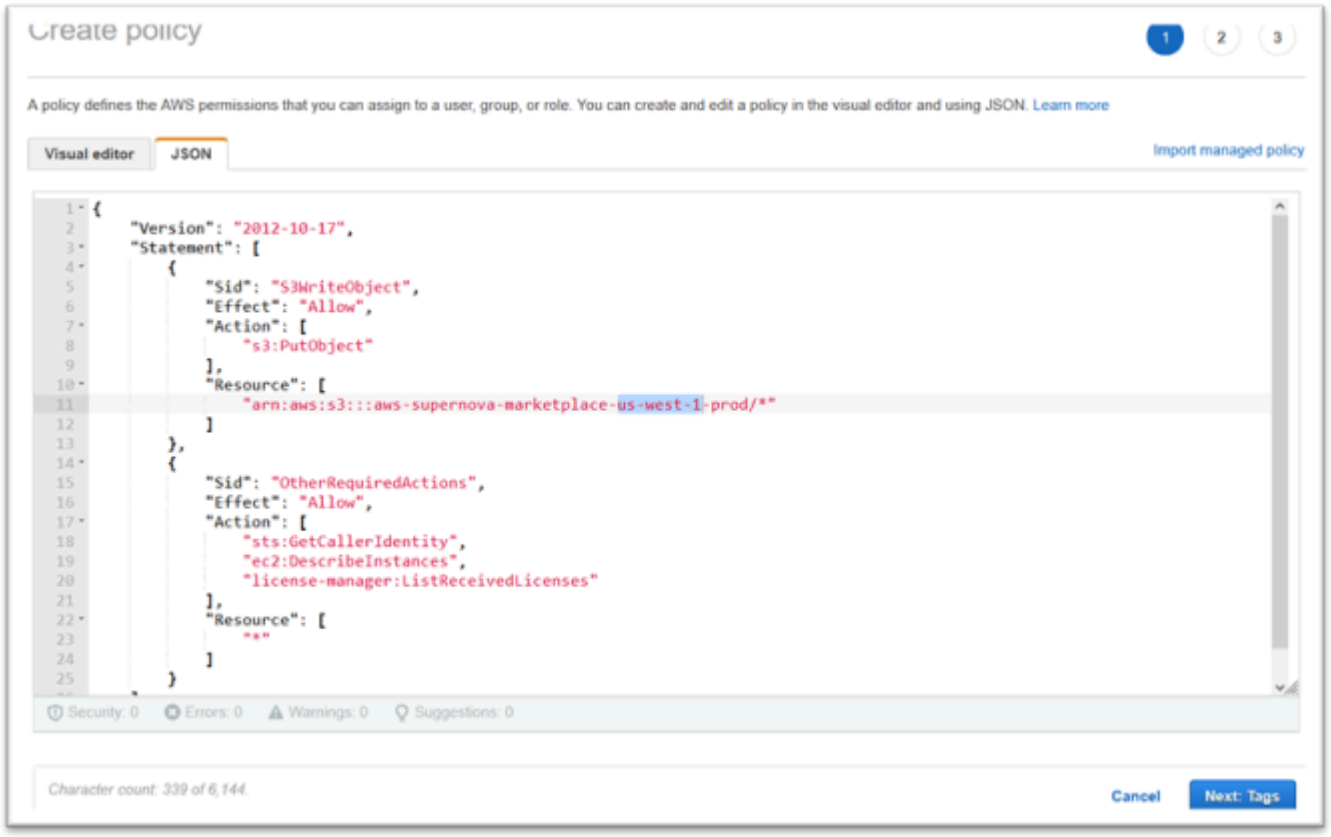
3. 选择 JSON 选项卡。

4. 将以下 JSON us-west-1 中的替换为定义 Amazon S3 终端节点 AWS 区域 的位置，然后将 JSON 复制并粘贴到策略编辑器中。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3WriteObject",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::aws-supernova-marketplace-us-west-1-prod/*"
            ]
        },
        {
            "Sid": "OtherRequiredActions",
            "Effect": "Allow",
            "Action": [
                "sts:GetCallerIdentity",
                "ec2:DescribeInstances",
                "license-manager:ListReceivedLicenses"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
```

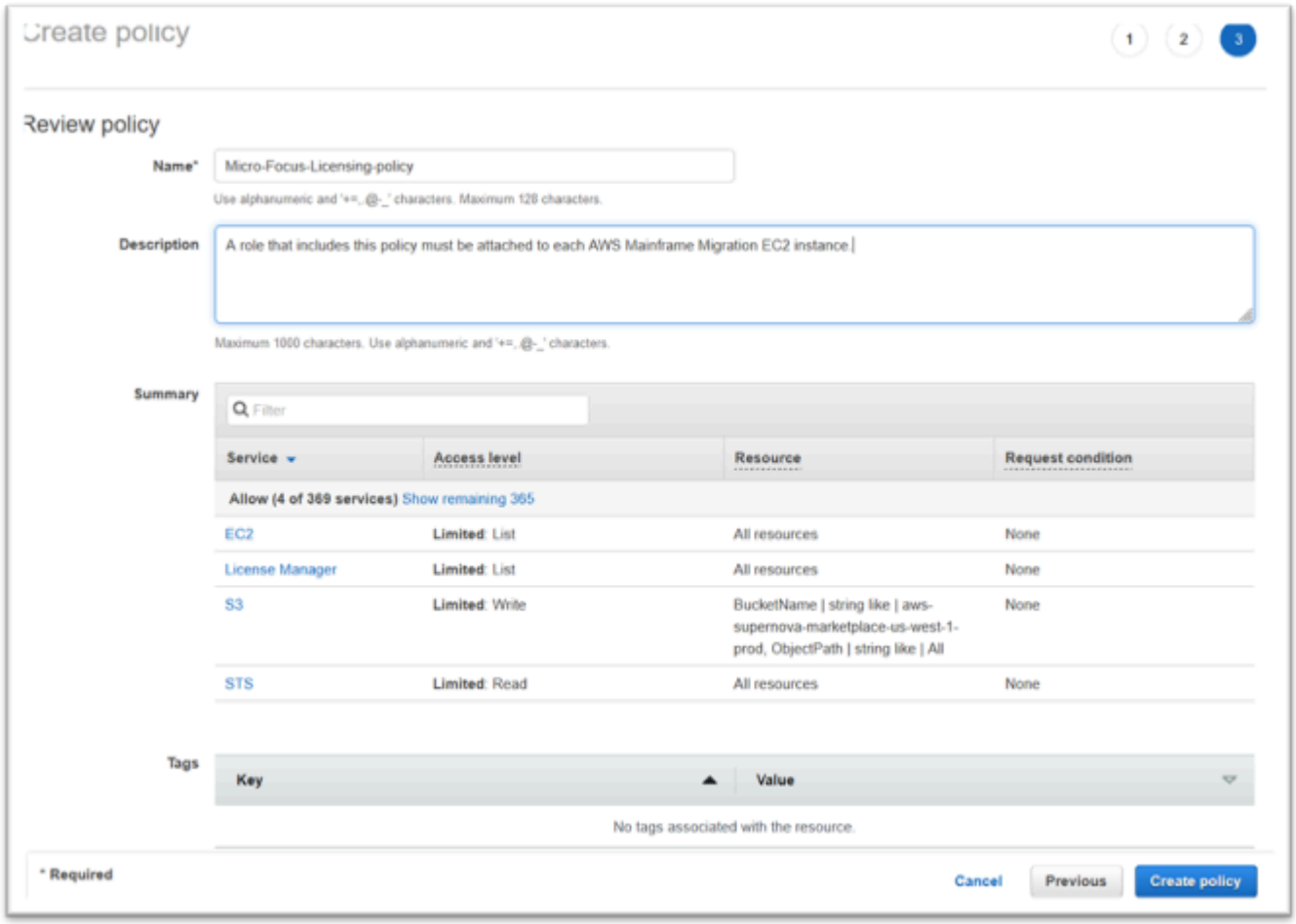
```
}
```

> **Note**
>
> Sid `OtherRequiredActions` 下的操作不支持资源级别的权限，必须在资源元素中指定 `*`。



5. 选择下一步：标签。

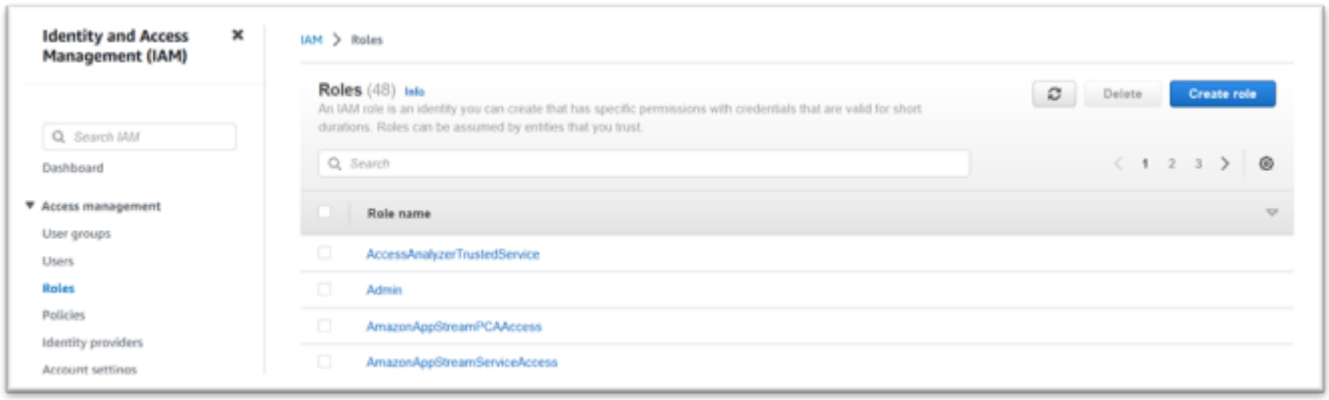
6. （可选）输入任何标签，然后选择下一步：审核。
7. 输入策略的名称，例如“Micro-Focus-Licensing-policy”。（可选）输入描述，例如 “必须将包含此策略的角色附加到每个 AWS 大型机现代化 Amazon EC2 实例”。



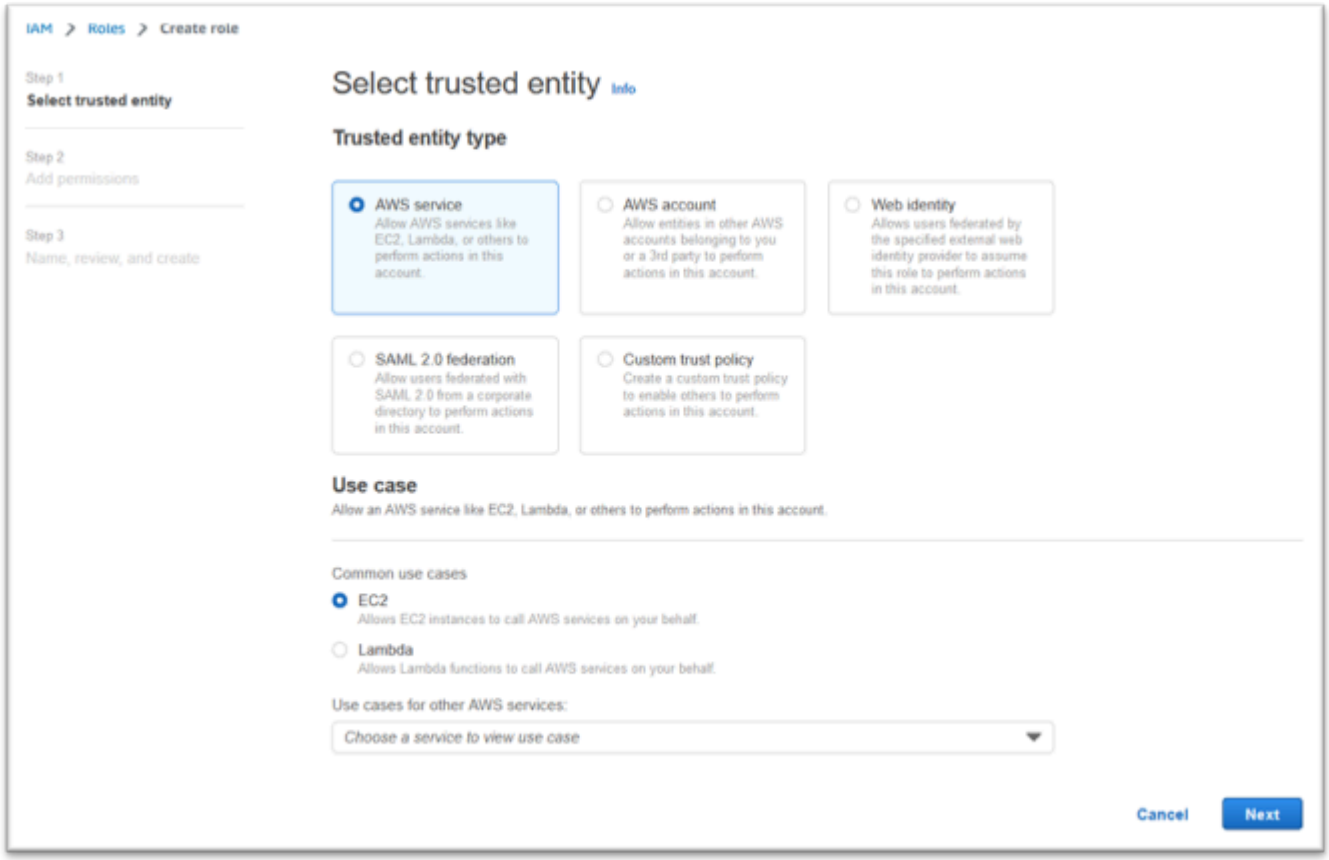
8. 选择创建策略。

创建 IAM 角色

1. 在 AWS Management Console中，导航到 IAM。
2. 选择角色，然后选择创建角色。

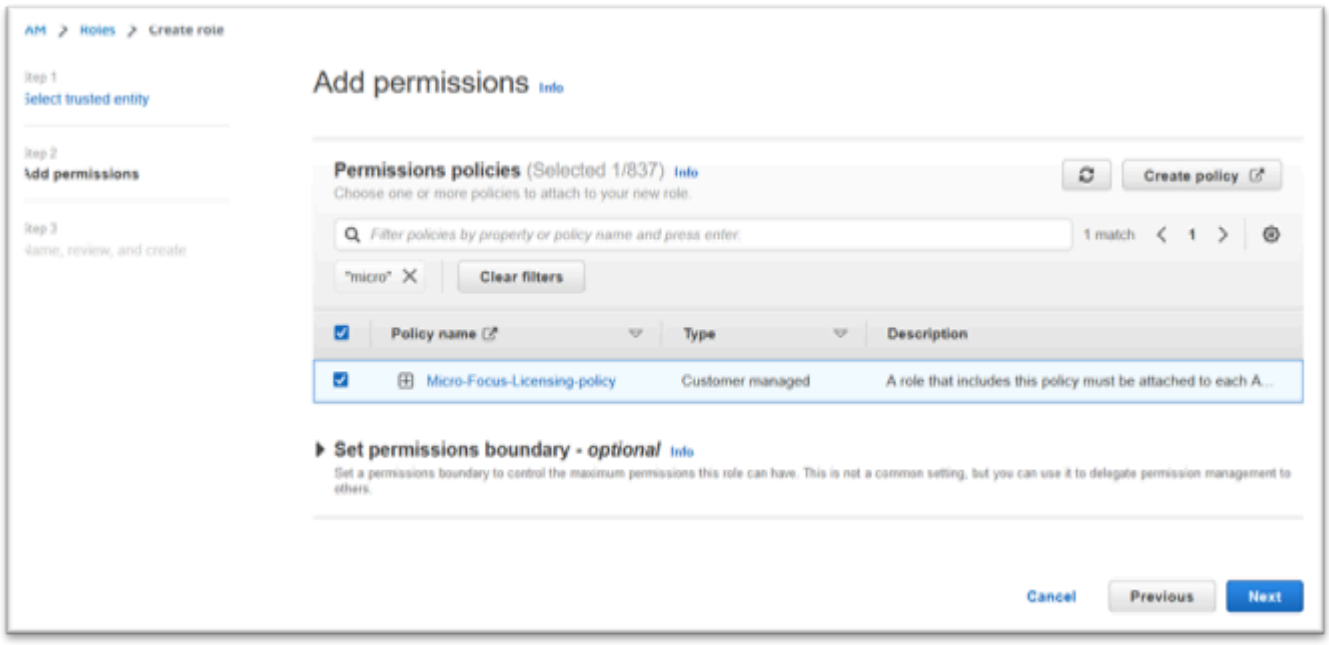


3. 保持可信实体类型为 AWS 服务，然后选择 EC2 常见用例。

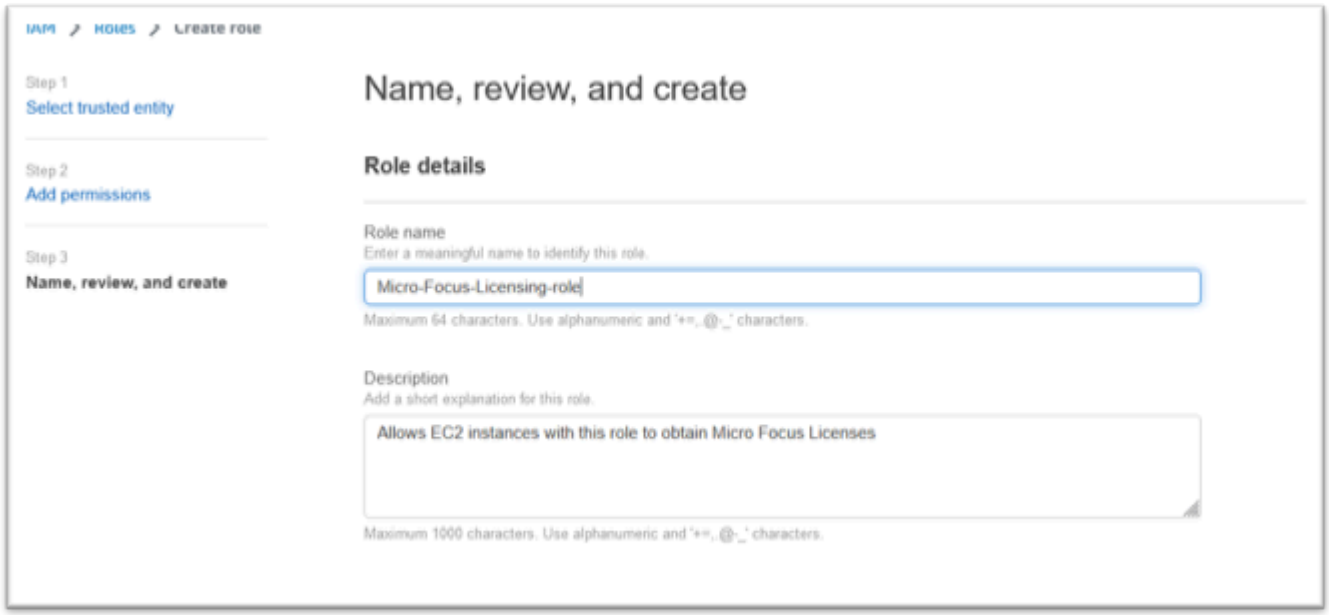


4. 选择下一步。
5. 在过滤条件中输入“Micro”，然后按 Enter 来应用该过滤条件。
6. 选择刚刚创建的策略，例如“Micro-Focus-Licensing-policy”。

7. 选择下一步。



8. 输入角色名称，例如“Micro-Focus-Licensing-role”。
9. 将描述替换为您自己的描述，例如“允许具有此角色的 Amazon EC2 实例获取 Micro Focus 许可证”。



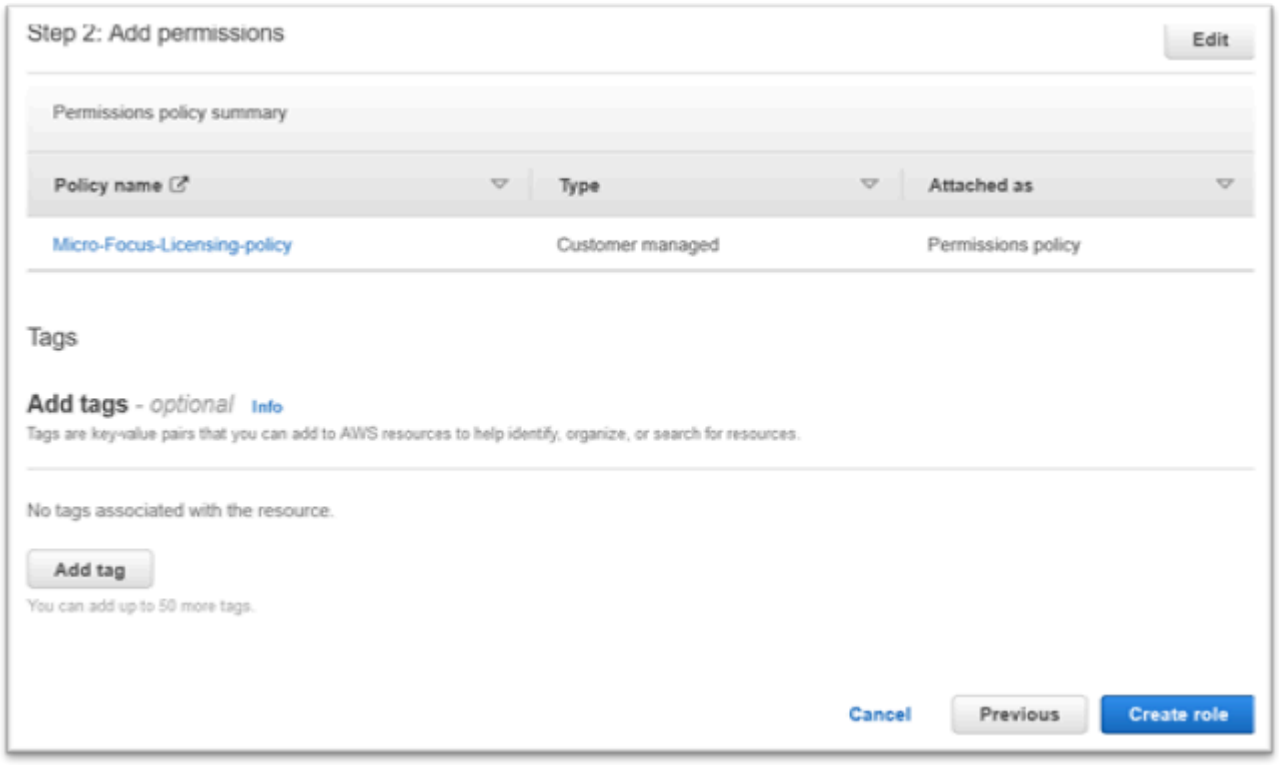
10. 在步骤 1：选择可信实体下，查看 JSON 并确认其具有以下值：

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

> **Note**
>
> Effect、Action 和 Principal 的顺序不重要。

11. 确认步骤 2：添加权限显示了您的许可策略。

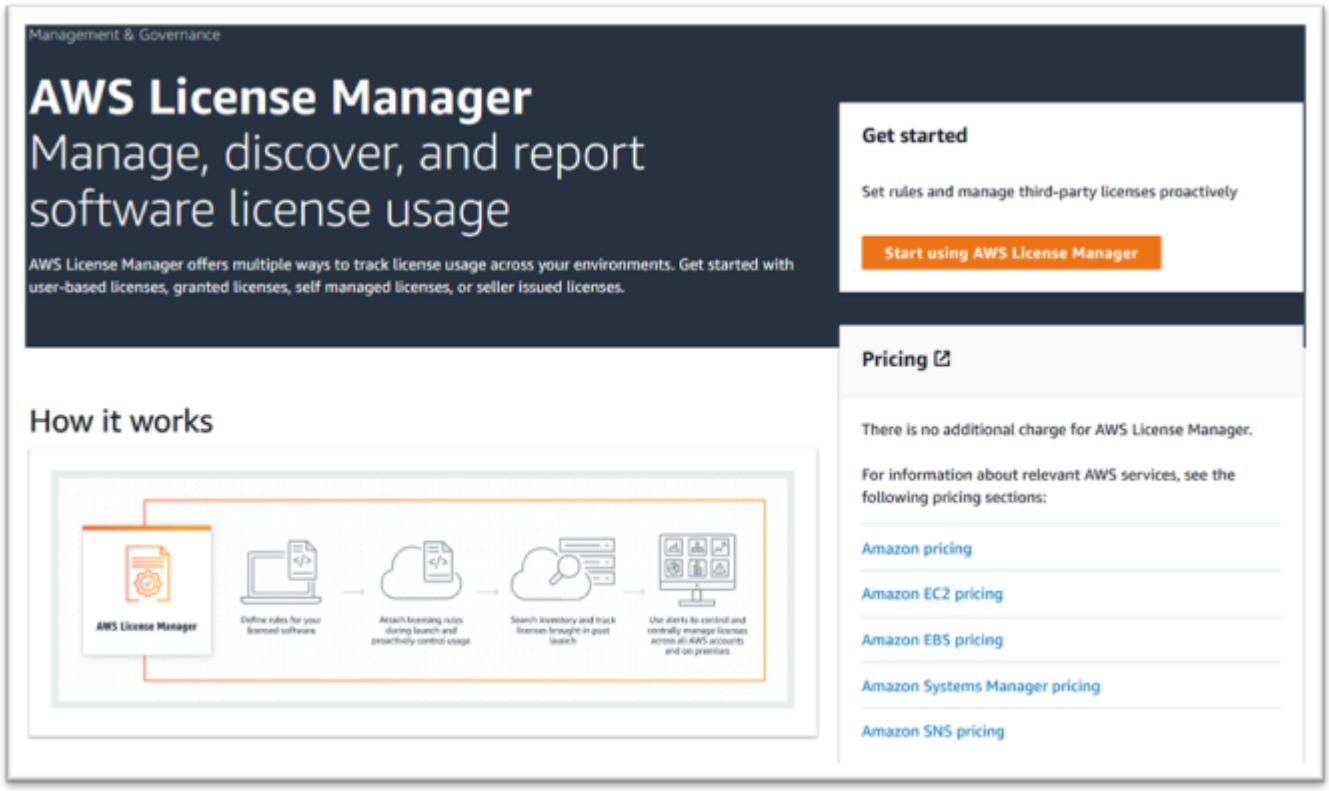


12. 选择创建角色。

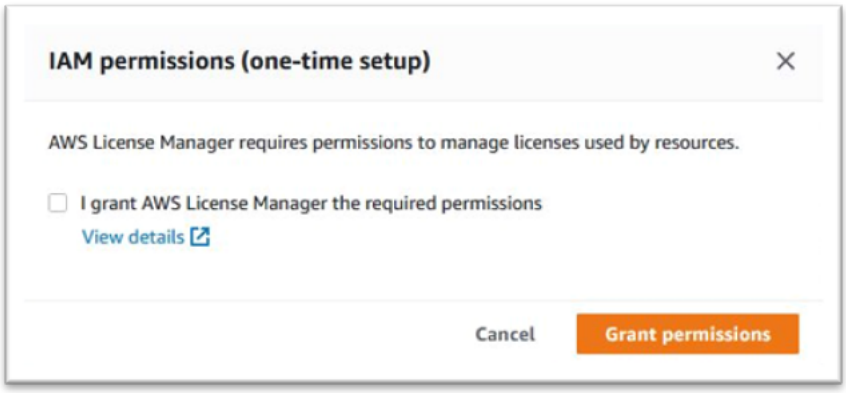
在完成允许列表申请之后，继续执行后续步骤。

授予 License Manager 所需权限

1. 导航 AWS License Manager 到 AWS Management Console。



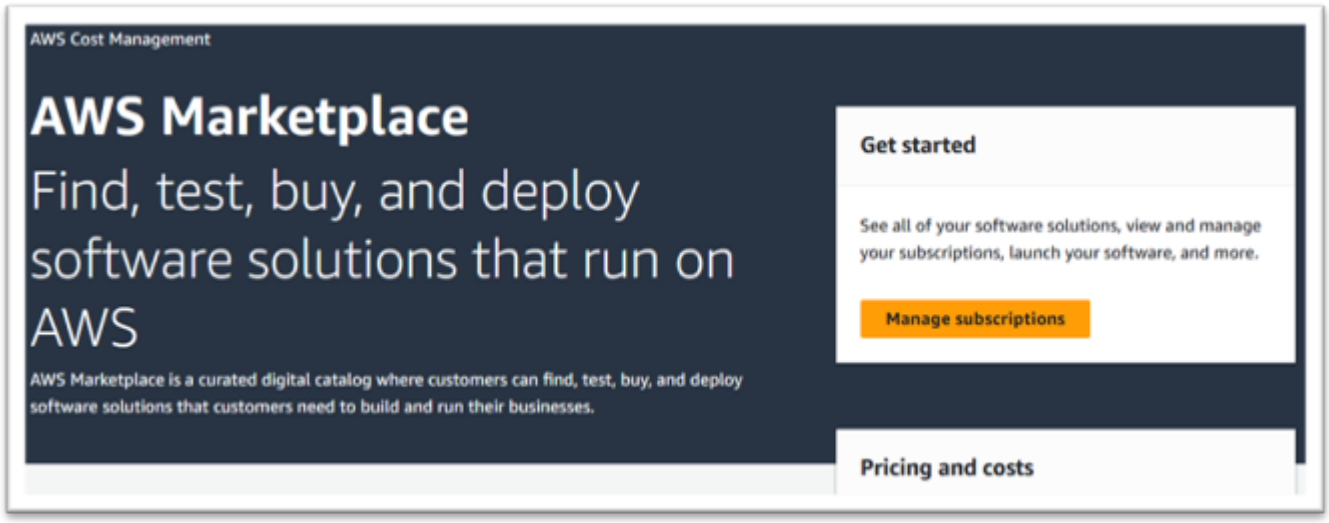
2. 选择开始使用 AWS License Manager。
3. 如果您看到以下弹出窗口，请查看详细信息，然后选中复选框并按授予权限。



订阅亚马逊机器映像

订阅产品后，您可以从该 AWS Marketplace 产品的 AMI 启动实例。

1. 导航到中的 “ AWS Marketplace 订阅” AWS Management Console。
2. 选择管理订阅。



3. 复制以下链接之一并将其粘贴到浏览器地址栏中。

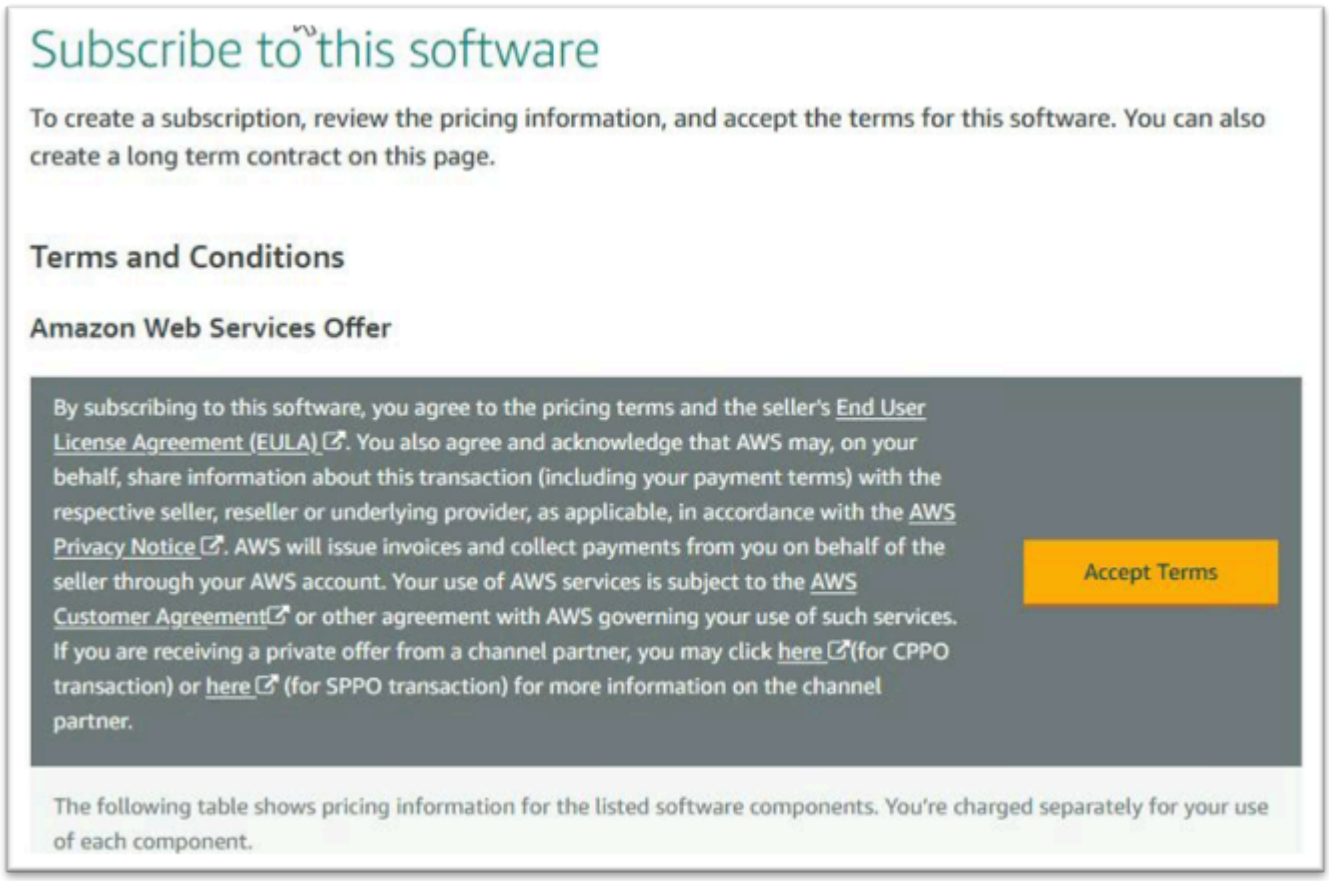
> **Note**
>
> 仅选择您已获授权使用的产品之一的链接。

- Enterprise Server：https://aws.amazon.com/marketplace/pp/prodview-g5emev63l7blc
- Enterprise Server for Windows：https://aws.amazon.com/marketplace/pp/prodview-lwybsiyikbhc2
- Enterprise Developer：https://aws.amazon.com/marketplace/pp/prodview-77qmpr42yzxwk
- Enterprise Developer with Visual Studio 2022：https://aws.amazon.com/marketplace/pp/prodview-m4l3lqiszo6cm
- Enterprise Analyzer：https://aws.amazon.com/marketplace/pp/prodview-tttheylcmcihm
- Enterprise Build Tools for Windows：https://aws.amazon.com/marketplace/pp/prodview-2rw35bbt6uozi
- Enterprise Stored Procedures：https://aws.amazon.com/marketplace/pp/prodview-zoeyqnsdsj6ha

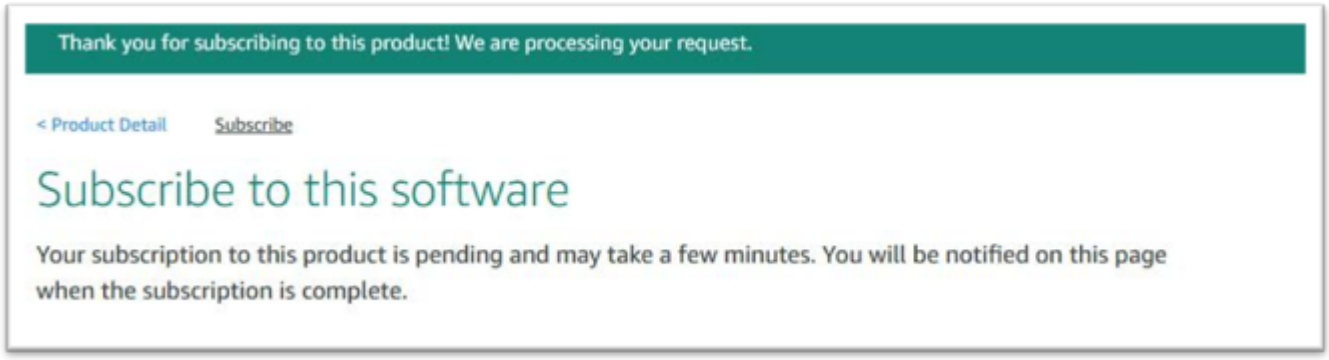
- Enterprise Stored Procedures with SQL Server 2019：https://aws.amazon.com/marketplace/pp/prodview-ynfklquwubnz4

4. 选择继续订阅。

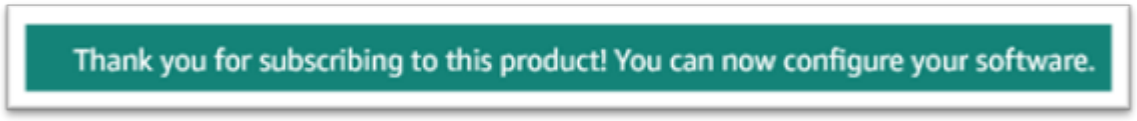
5. 如果可以接受“条款和条件”，请选择接受条款。



6. 处理此订阅可能需要几分钟时间。

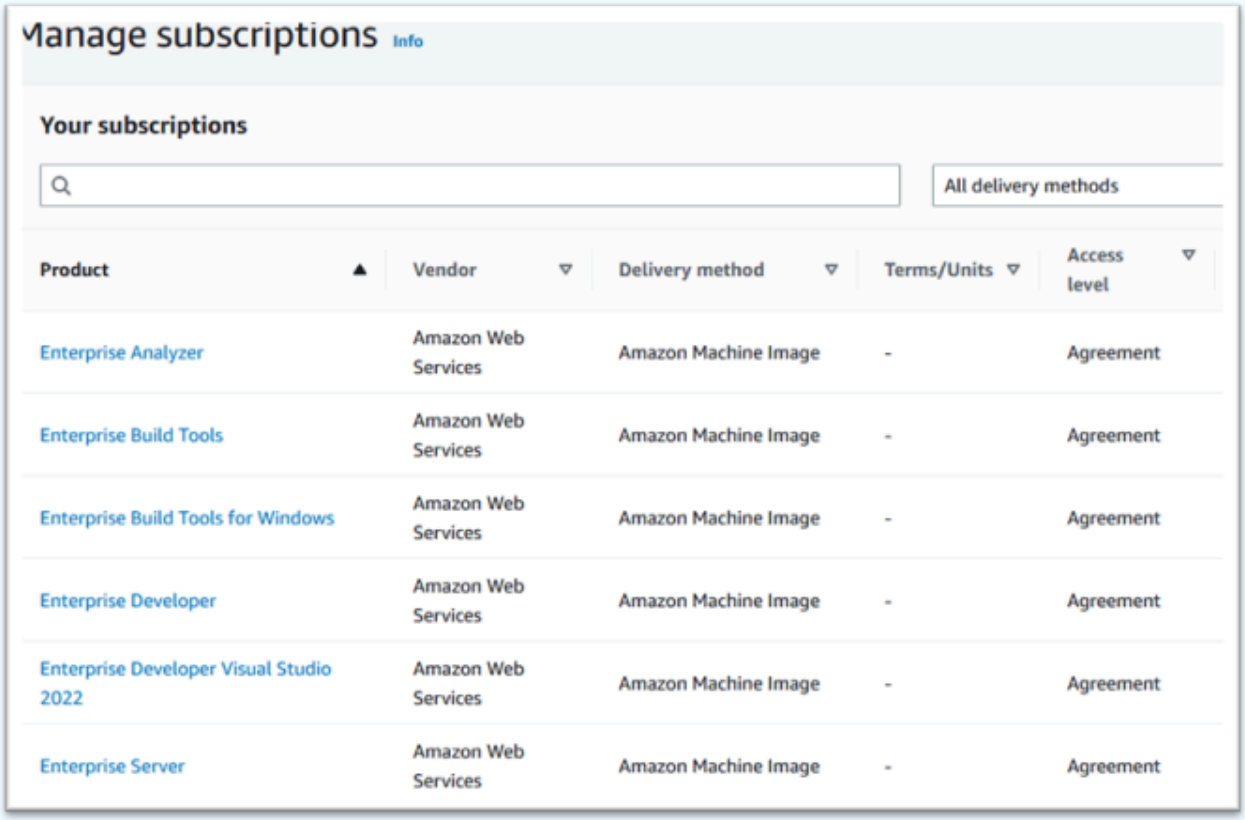


7. 显示感谢消息后，复制并粘贴步骤 3 中的下一个链接以继续添加订阅。



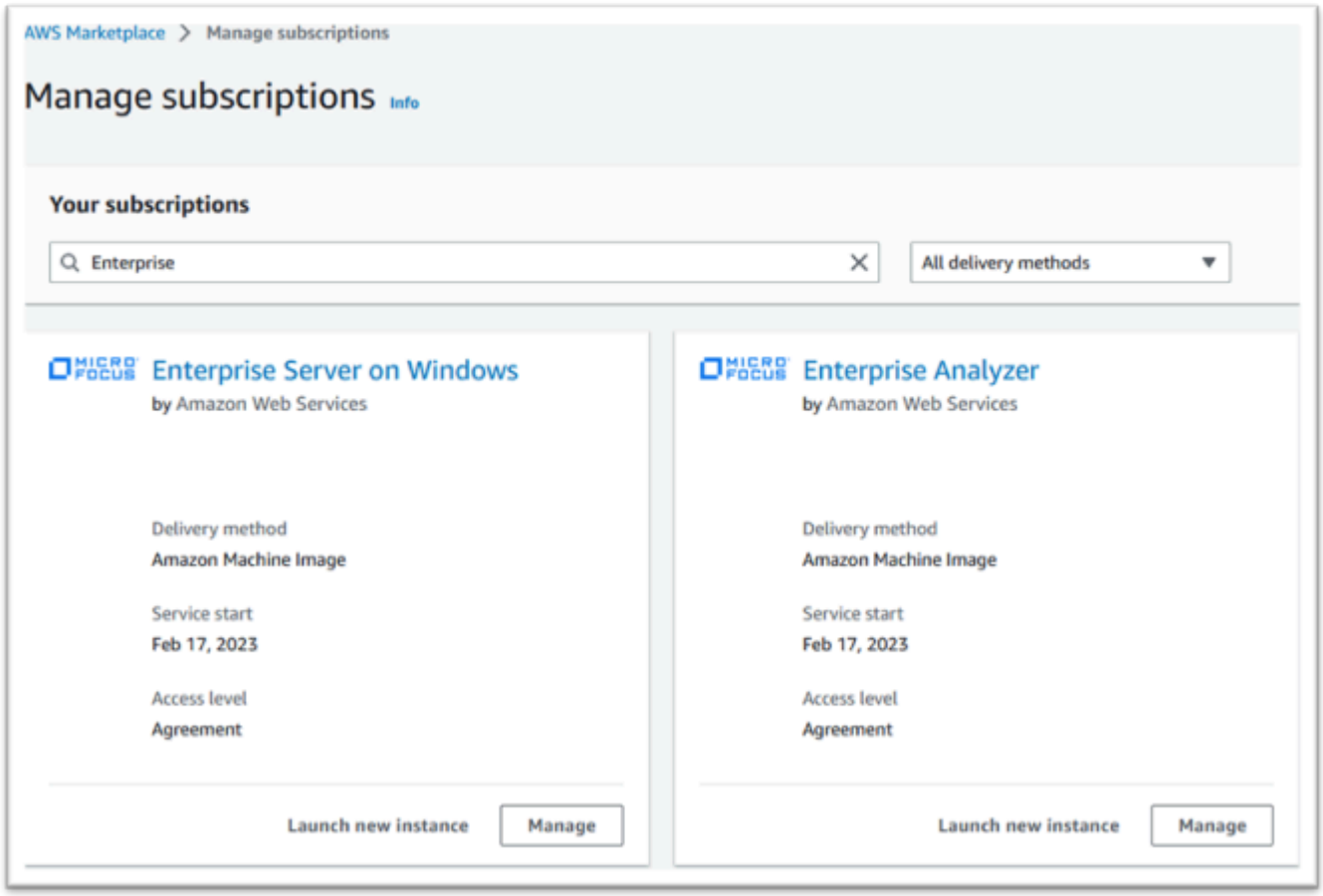
8. 当管理订阅显示所有已订阅的 AMI 时停止添加。

> **Note**
>
> 面板首选项（齿轮图标）设置为以表格形式显示。



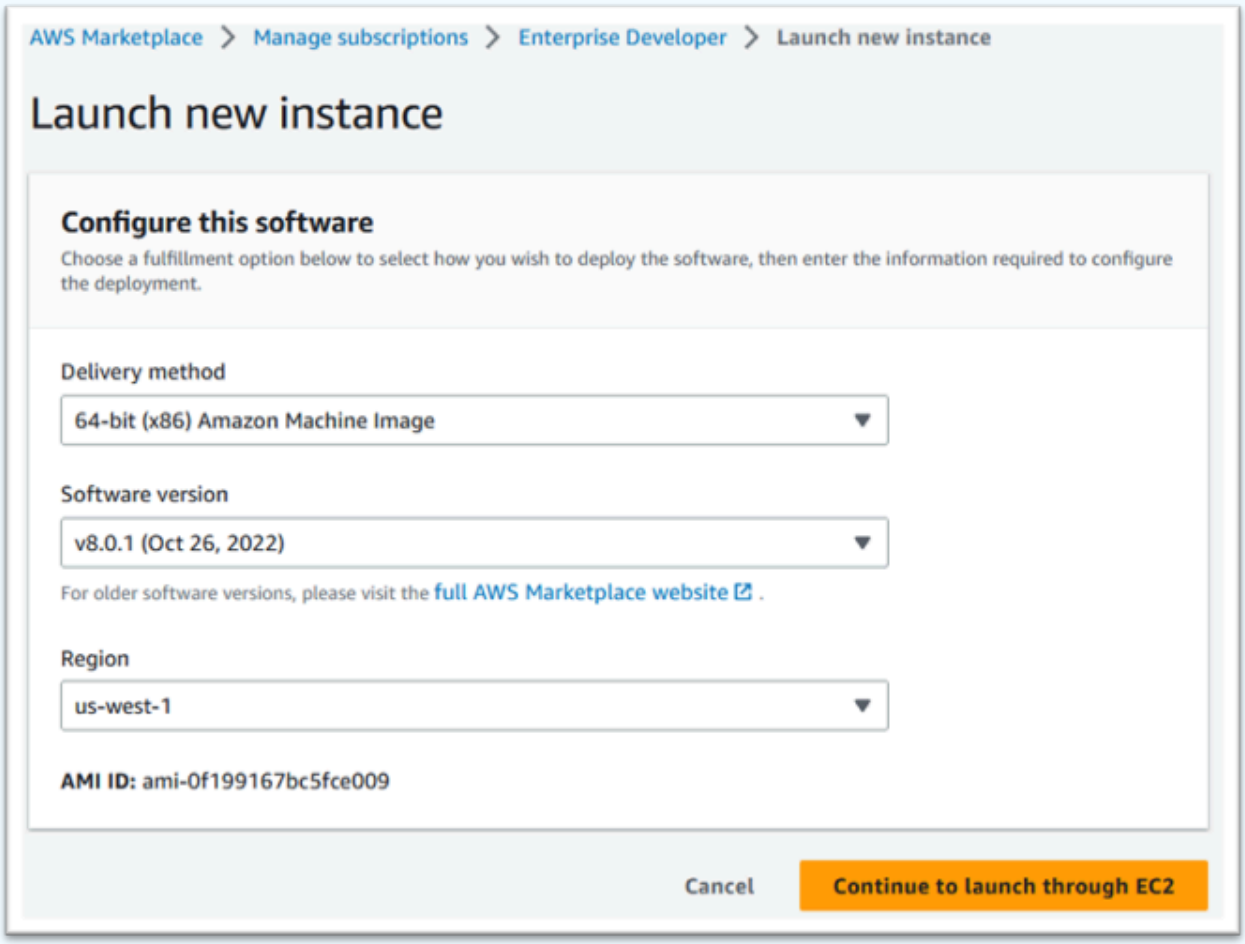
启动 AWS 大型机现代化 Micro Focus 实例

1. 导航到中的 “ AWS Marketplace 订阅” AWS Management Console。
2. 找到要启动的 AMI，然后选择启动新实例。



3. 在“启动新实例”对话框中，确保选择允许列表中的区域。
4. 按继续通过 EC2 启动。

> **Note**
>
> 以下示例显示了企业开发者 AMI 的启动，但所有 AWS 大型机现代化 AMI 的流程都是一样的。

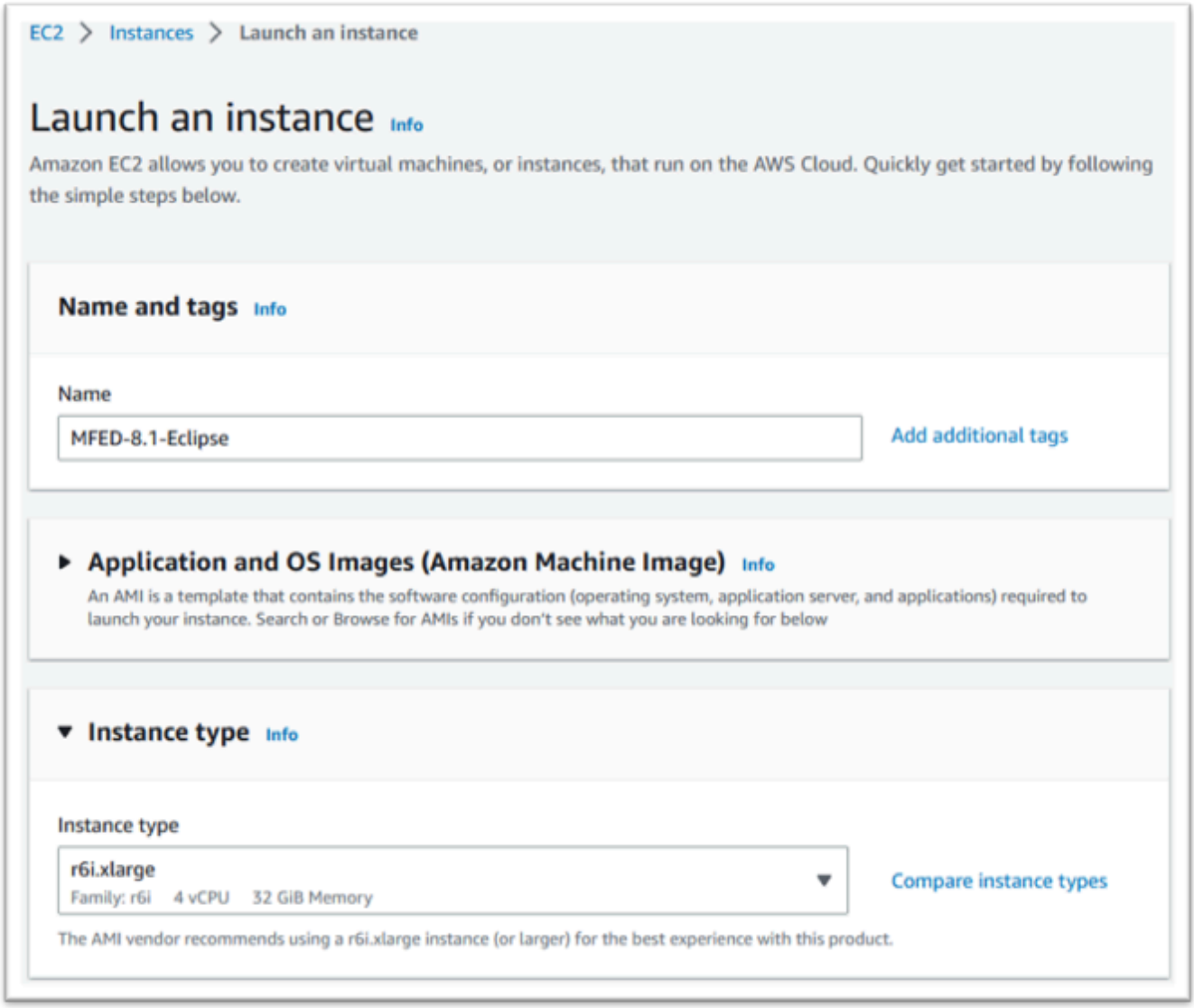


5. 输入服务器的名称。
6. 选择一个实例类型。

   所选的实例类型应由项目性能和成本要求决定。以下是建议的起点：

   - 对于 Enterprise Analyzer，选择 r6i.xlarge
   - 对于 Enterprise Developer，选择 r6i.large
   - 对于 Enterprise Server 的独立实例，选择 r6i.xlarge
   - 对于具有横向扩展功能的 Micro Focus Performance Availability Cluster (PAC)，选择 r6i.large

   > **Note**
   >
   > 屏幕截图中的“应用程序和操作系统映像”部分已折叠。

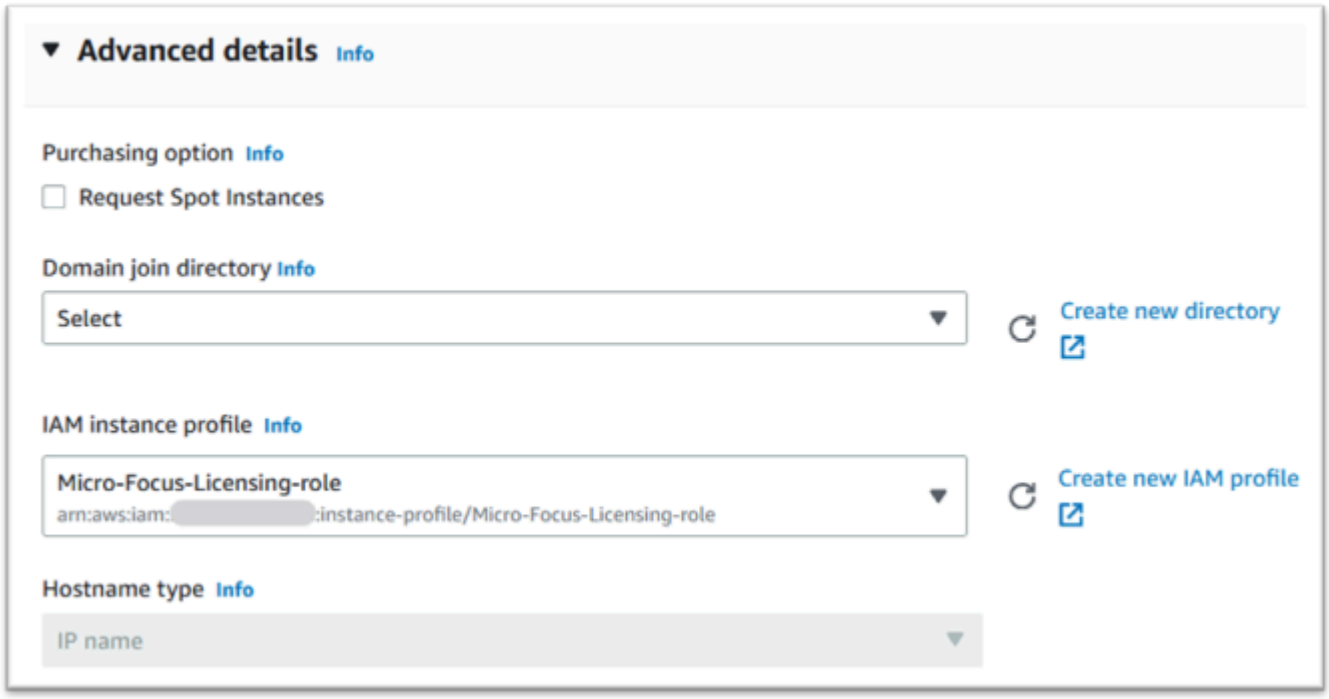


7. 选择或创建（并保存）密钥对（未显示）。

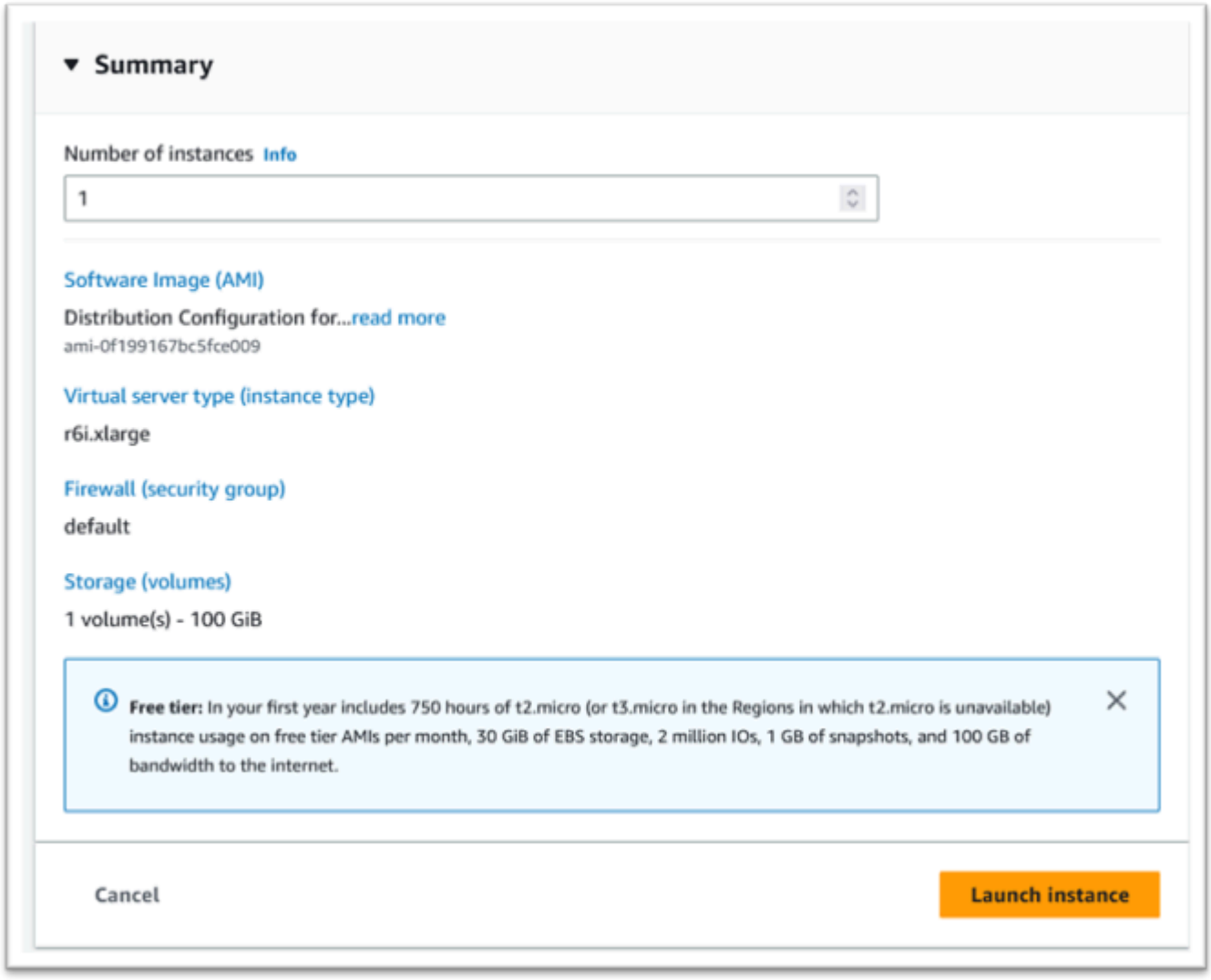
   有关密钥对和 Linux 实例的更多信息，请参阅 Amazon EC2 密钥对和 Linux 实例。

   有关密钥对和 Windows 实例的更多信息，请参阅 Amazon EC2 密钥对和 Windows 实例。
8. 编辑“网络设置”并选择允许列表中的 VPC 和相应的子网。
9. 选择或创建安全组。如果使用的是 Enterprise Server EC2 实例，则通常允许 TCP 流量进入端口 86 和 10086 来管理 Micro Focus 配置。
10. （可选）为 Amazon EC2 实例配置存储。
11. 重要提示 – 展开“高级详细信息”，然后在“IAM 实例配置文件”下选择之前创建的许可角色，例如“Micro-Focus-Licensing-role”。

> **Note**
>
> 如果遗漏此步骤，您可以在创建实例后，从 EC2 实例“操作”菜单的“安全”选项中修改 IAM 角色。

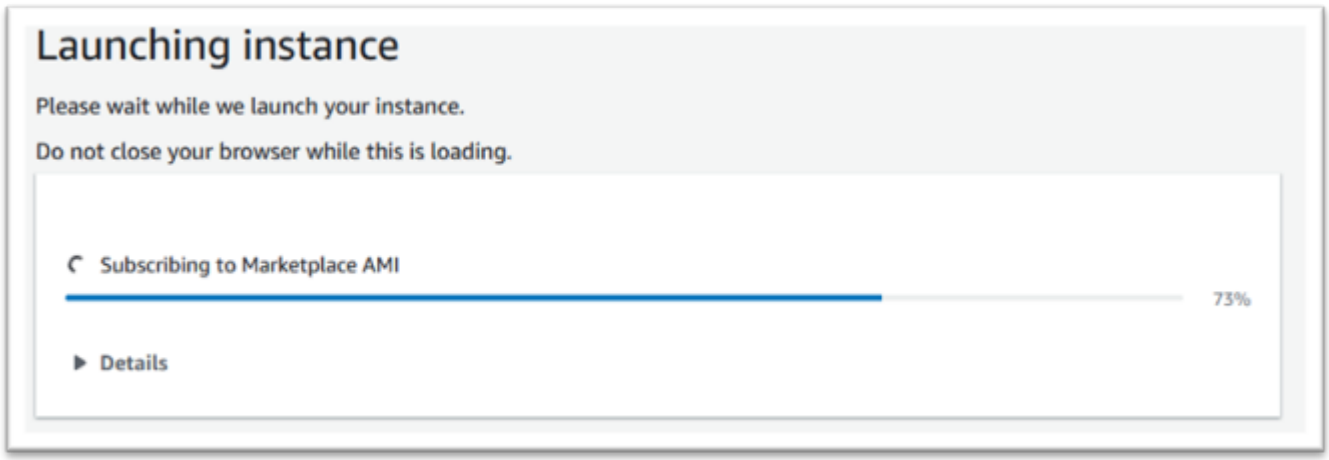


12. 审核“摘要”并按启动实例。

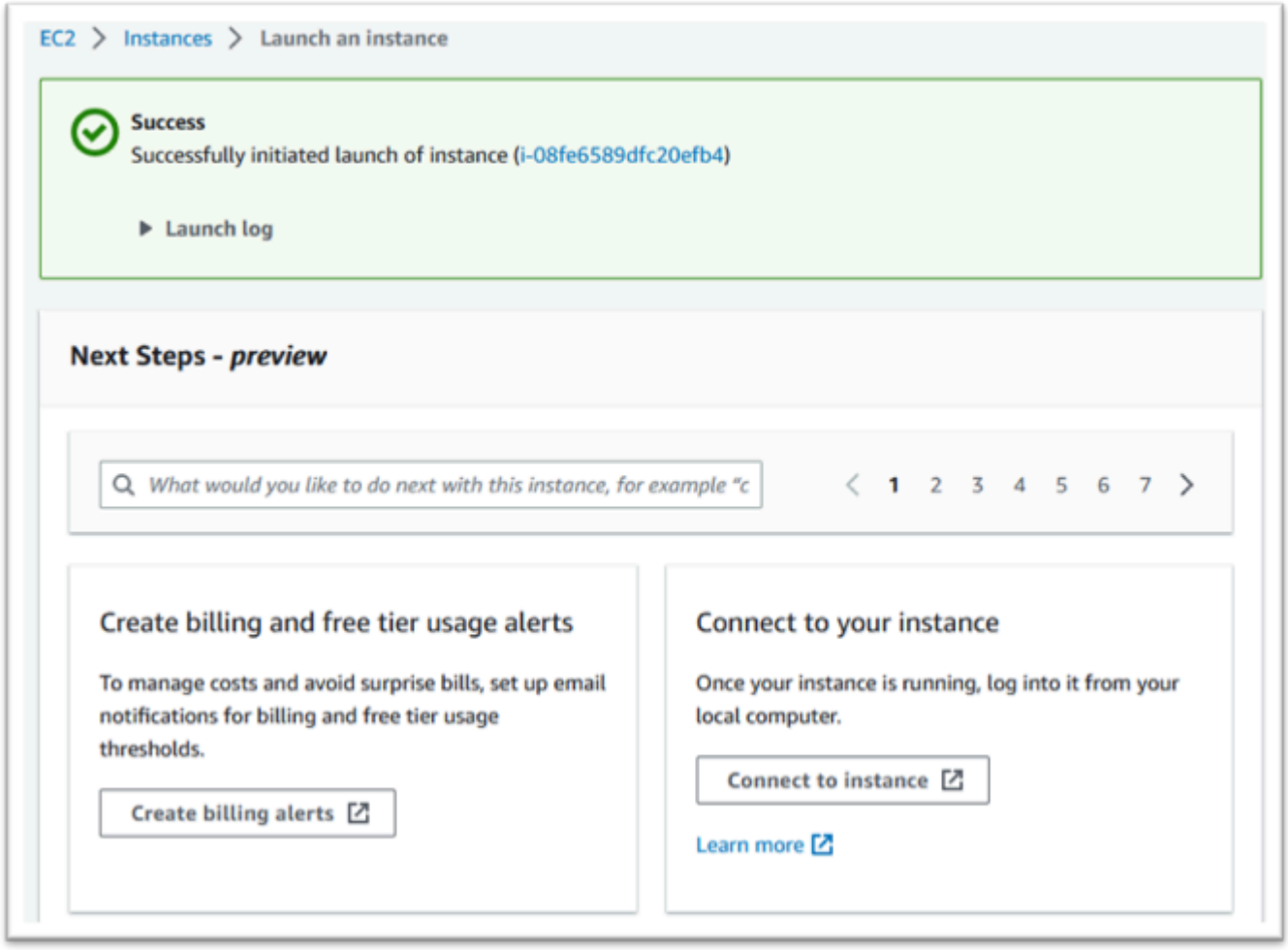


13. 如果选择的虚拟服务器类型无效，则实例启动将失败。

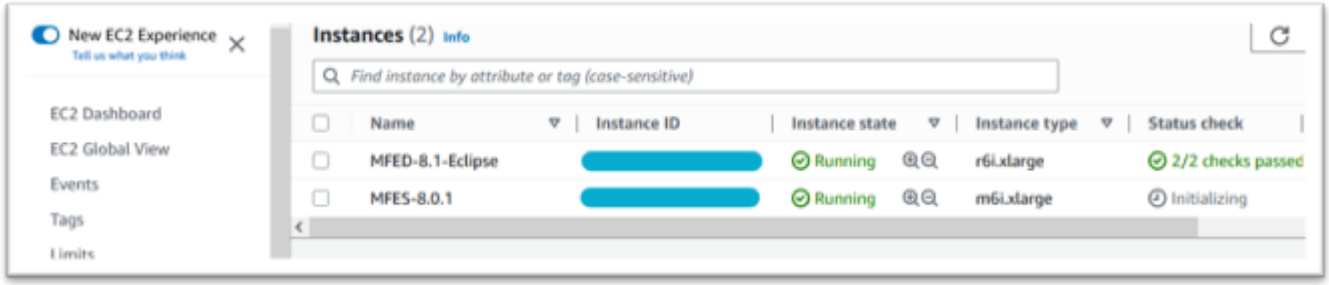
    如果发生这种情况，请选择编辑实例配置并更改实例类型。



14. 显示“成功”消息后，选择连接到实例以获取连接详细信息。



15. 或者，在 AWS Management Console中导航到 EC2。
16. 选择实例以查看新实例的状态。



无法访问 Internet 的子网或 VPC

如果子网或 VPC 没有出站 Internet 访问权限，请进行以下额外更改。

License Manager 需要访问以下 AWS 服务：

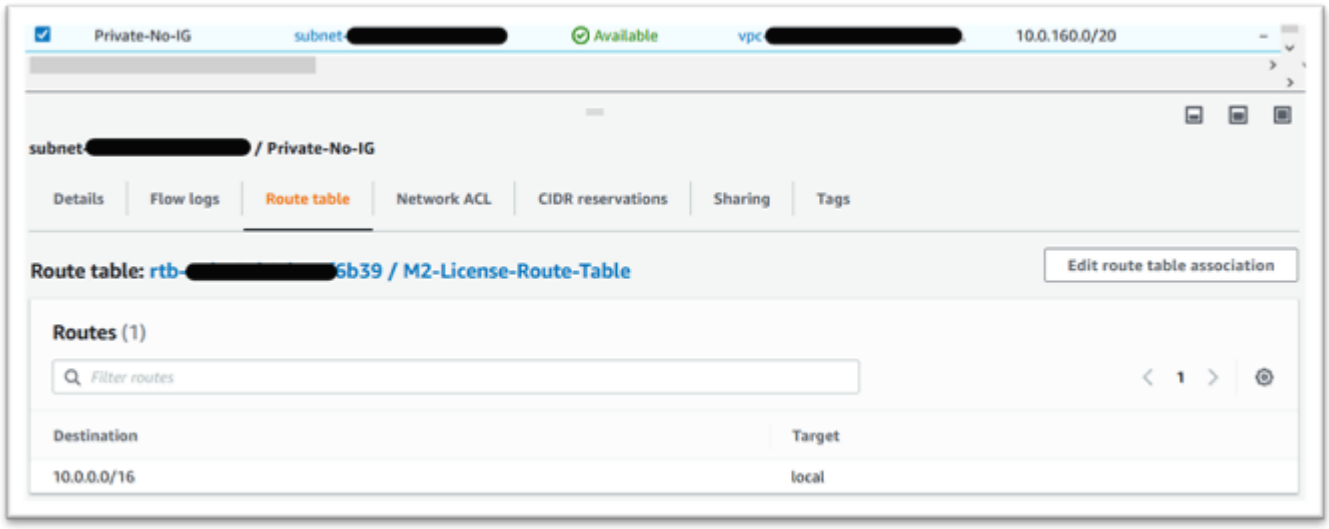
- com.amazonaws.*region*.s3
- com.amazonaws.*region*.ec2
- com.amazonaws.*region*.license-manager
- com.amazonaws.*region*.sts

之前的步骤定义了 com.amazonaws.*region*.s3 服务作为网关端点。此端点需要一个路由表条目，用于所有无法访问 Internet 的子网。

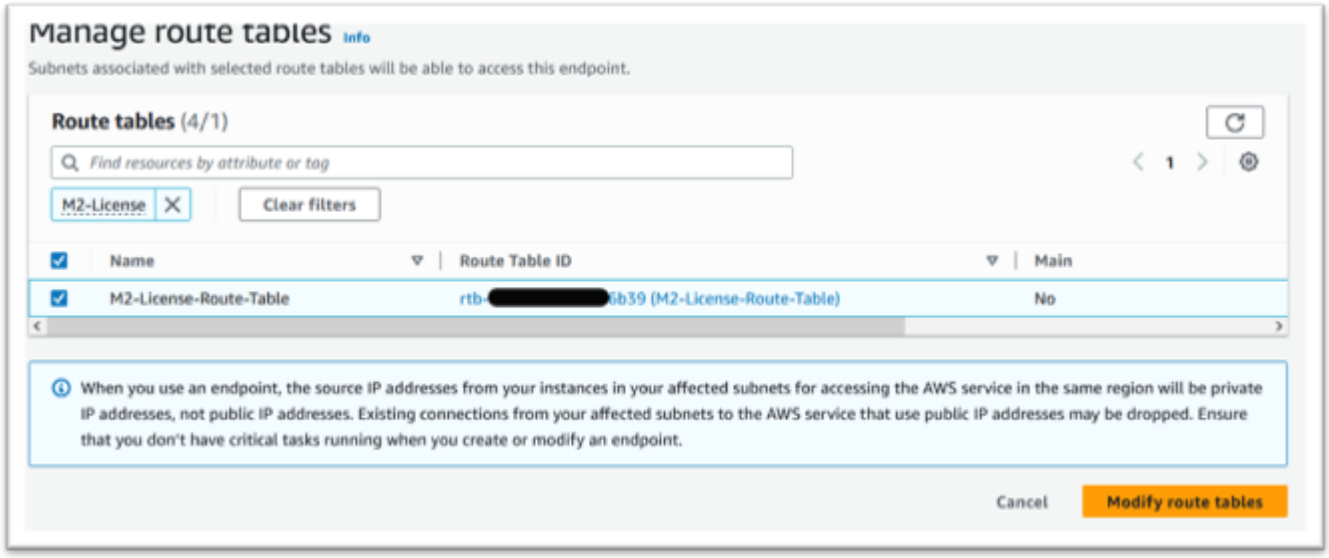
另外三项服务将被定义为接口端点。

为 Amazon S3 端点添加路由表条目

1. 导航到中的 VPC， AWS Management Console 然后选择子网。
2. 选择将在其中创建 Amazon EC2 实例的子网，然后选择“路由表”选项卡。
3. 记录路由表 ID 的几位尾部数字。例如，下图中的 6b39。



4. 在导航窗格中，选择端点。
5. 选择之前创建的端点，然后从端点的“路由表”选项卡或从“操作”下拉列表中选择管理路由表。
6. 选择标有之前记录数字的路由表，然后按“修改路由表”。

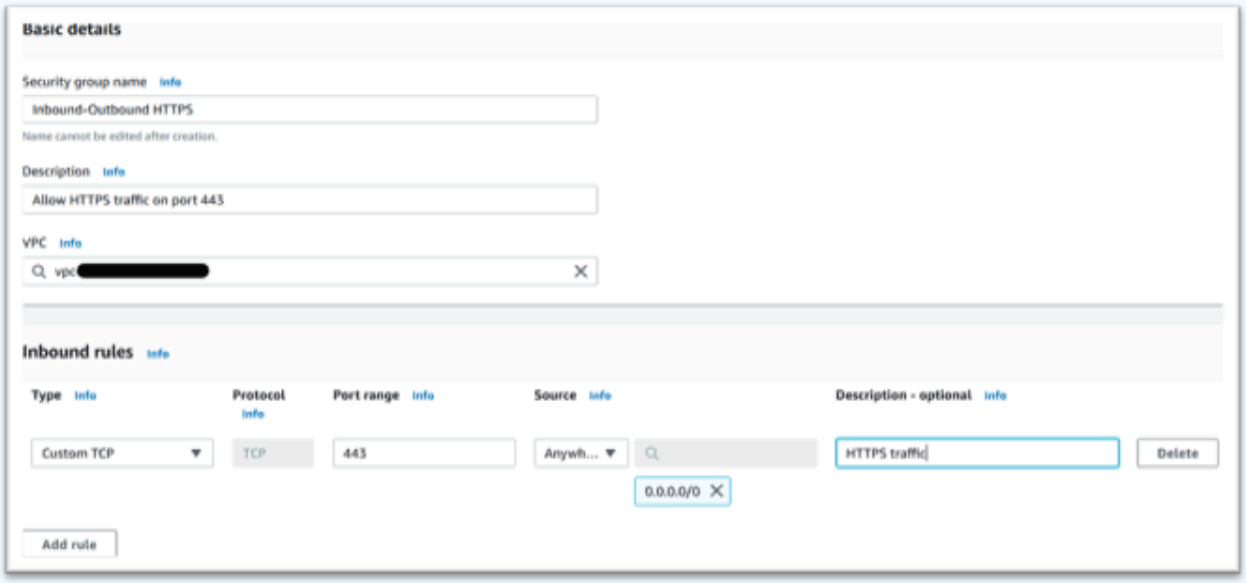


定义所需的安全组

Amazon EC2 和 Lic AWS STS ense Manager 服务通过端口 443 通过 HTTPS 进行通信。这种通信是双向的，需要入站和出站规则来允许实例与服务通信。

1. 在 AWS Management Console中，导航到 Amazon VPC。
2. 在导航栏中，找到安全组，然后选择创建安全组。
3. 输入安全组名称和描述，例如“Inbound-Outbound HTTPS”。
4. 在 VPC 选择区域中按“X“来移除默认 VPC，然后选择包含 S3 端点的 VPC。
5. 添加一条入站规则，在端口 443 上允许来自任何源的 TCP 流量。

> **Note**
>
> 通过限制“源”，可以进一步限制入站（和出站规则）。有关更多信息，请参阅 Amazon VPC 用户指南中的使用安全组控制 AWS 资源流量。

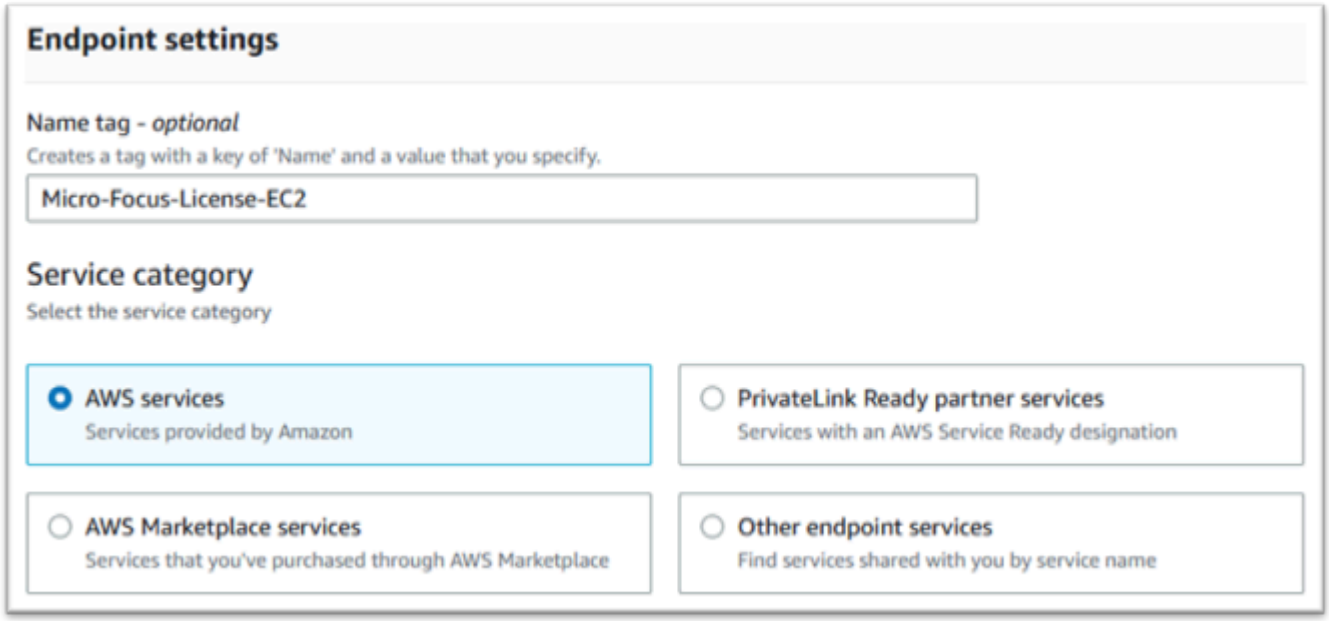


6. 按创建安全组。

创建服务端点

重复以下过程三次，即针对每个服务各执行一次。

1. 导航到中的 Amazon VPC， AWS Management Console 然后选择终端节点。
2. 按创建端点。
3. 输入名称，例如“Micro-Focus-License-EC2”、“Micro-Focus-License-STS”或“Micro-Focus-License-STS”。
4. 对于“服务类别”，选择 AWS 服务。



5. 在“服务”下搜索匹配的接口服务，即以下服务之一：

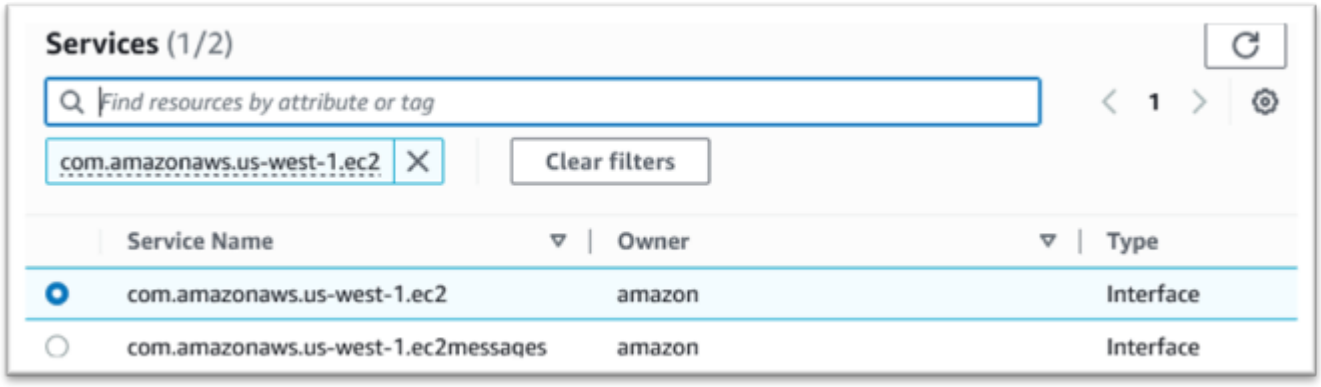
   - “com.amazonaws.*region*.ec2”
   - “com.amazonaws.*region*.sts”
   - “com.amazonaws.*region*.license-manager”

   例如：

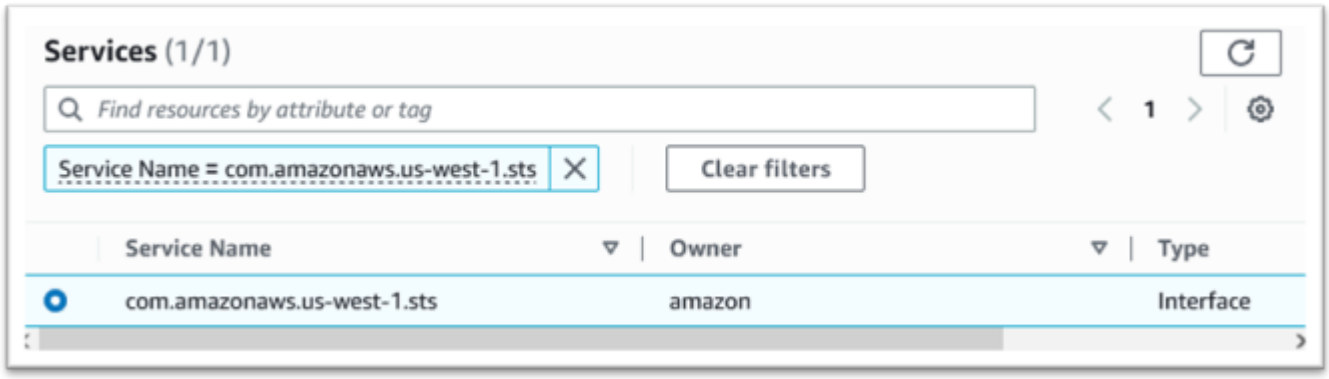
   - “com.amazonaws.us-west-1.ec2”
   - “com.amazonaws.us-west-1.sts”
   - “com.amazonaws.us-west-1.license-manager”

6. 选择匹配的接口服务。

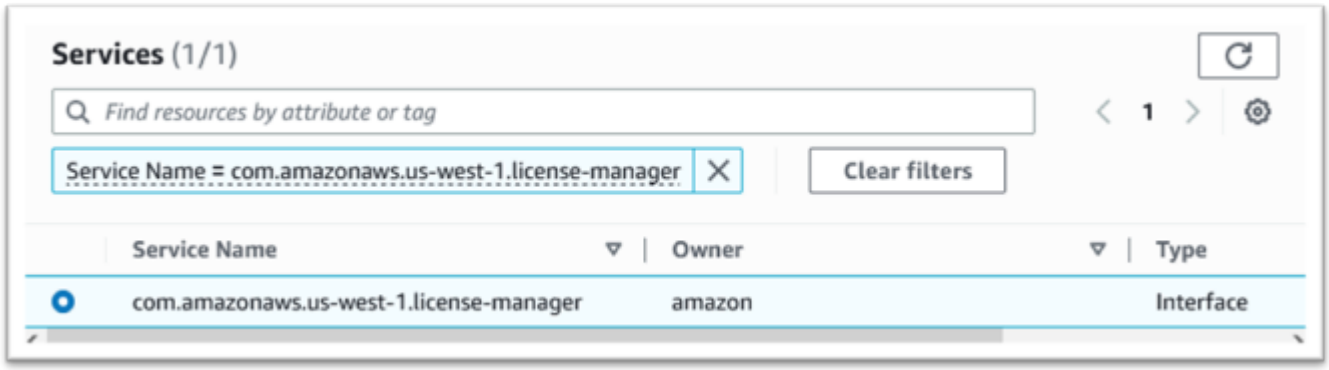
   com.amazonaws.***region***.ec2：



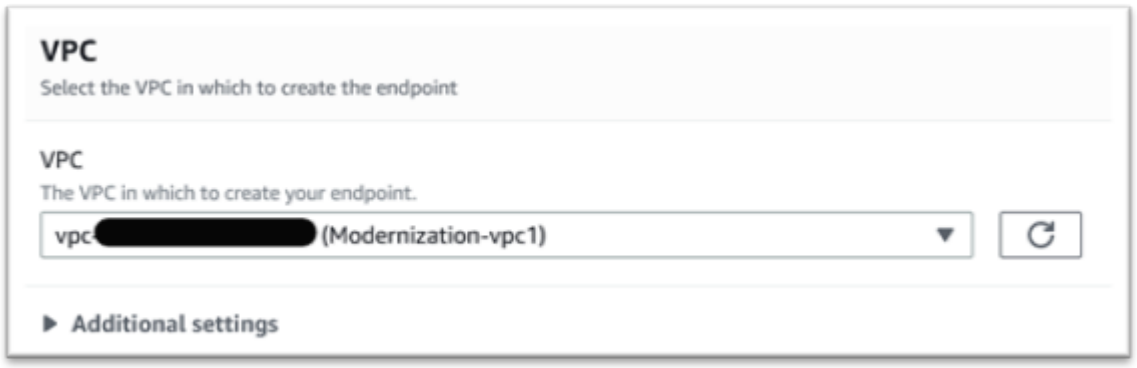
com.amazonaws.***region***.sts：



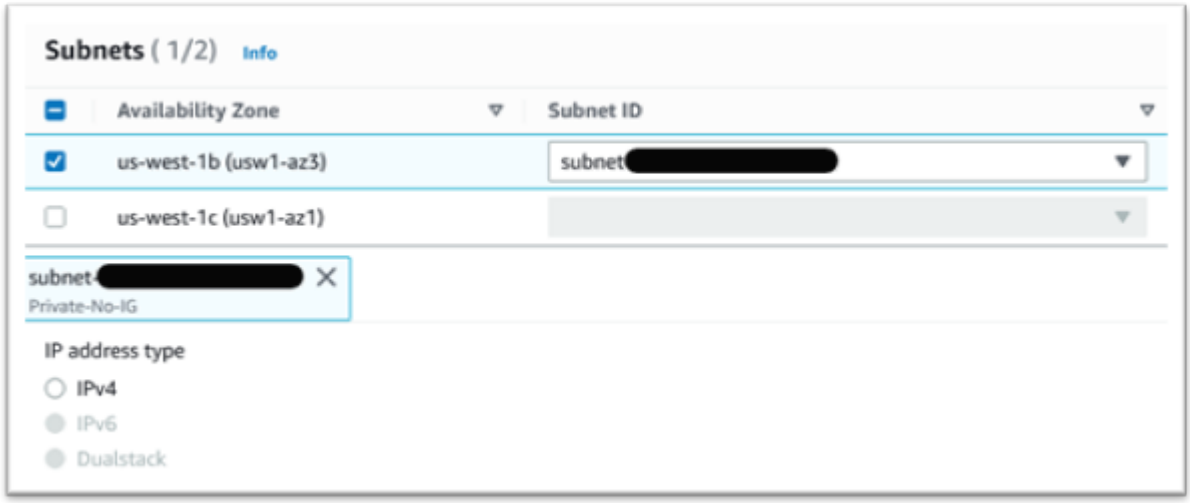
com.amazonaws.***region***.license-manager：



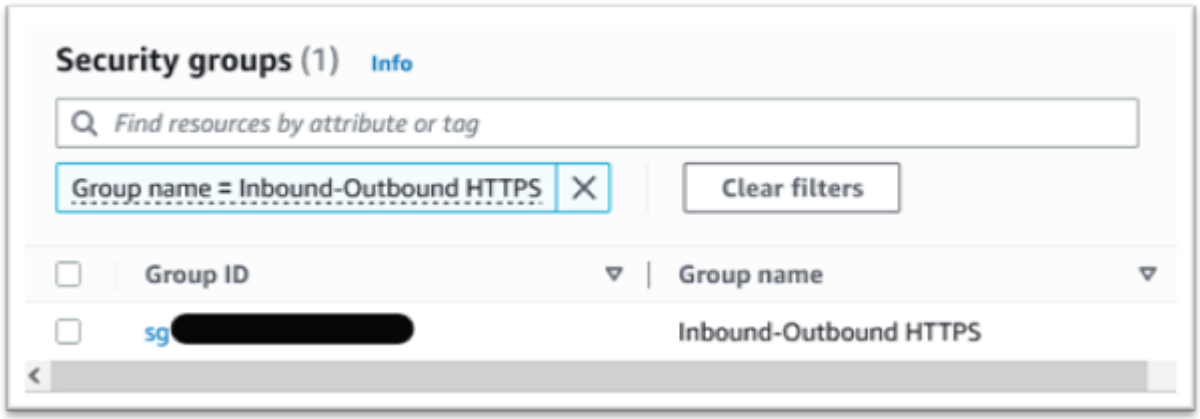
7. 对于 VPC，选择实例的 VPC。



8. 为 VPC 选择可用区和子网。



9. 选择之前创建的安全组。



10. 在“策略”下，选择完全访问权限。

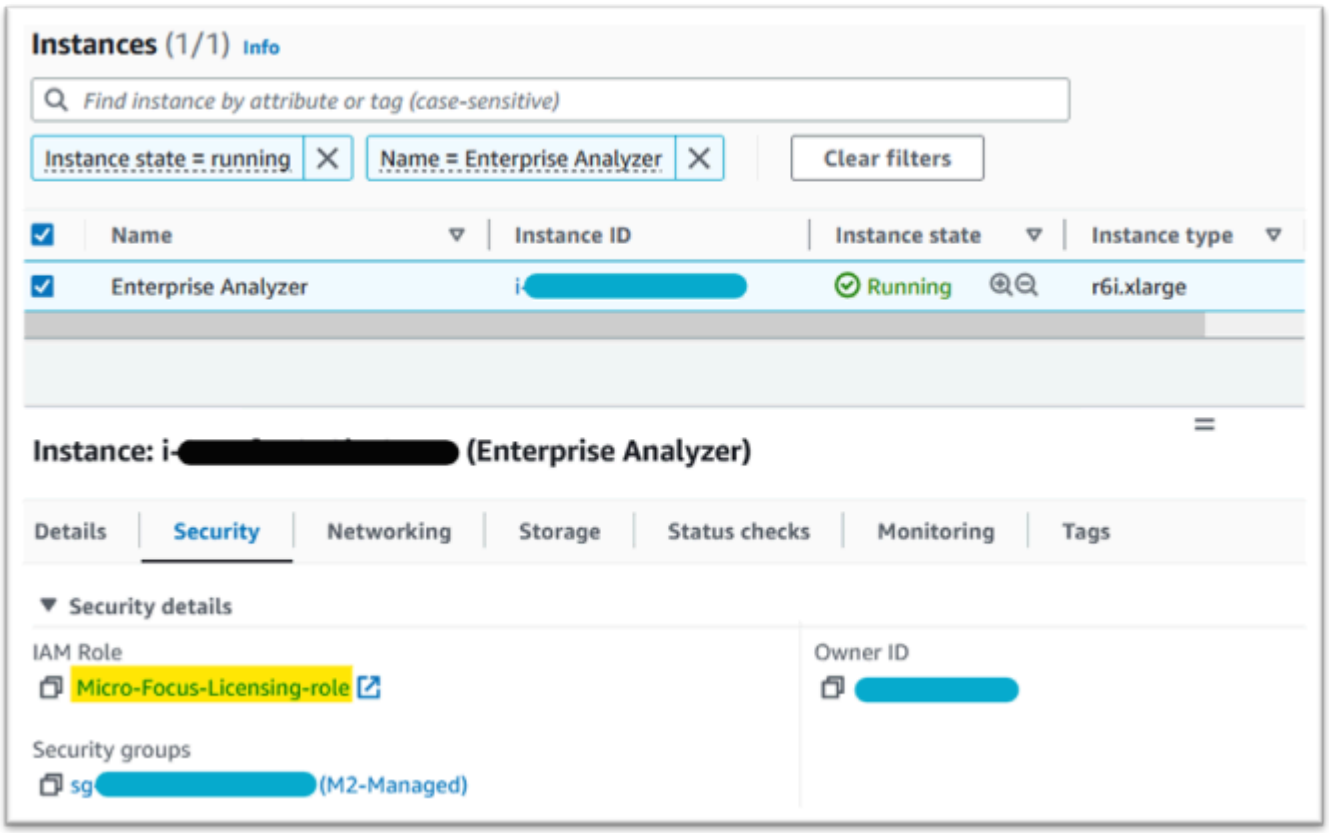
11. 选择创建端点。
12. 为其余接口重复此过程。

对许可证问题进行故障排除

如果您在访问或使用 AMI 时遇到问题，以下信息可能会对您有所帮助。

验证 Amazon EC2 实例是否具有 IAM 许可角色

可以在 Amazon EC2 实例详细信息的“安全”选项卡上进行检查。可以使用“操作”下拉菜单的“安全选项”进行更改。



使用 Reachability Analyzer

在控制台页面上找到 Reachability Analyzer。 AWS Network Manager

创建并分析从 AMI 创建的 Amazon EC2 实例和 Amazon S3 VPC 端点之间的路径。

如果 Amazon EC2 实例无法访问 Internet，请对所有 4 个端点重复路径分析。

有关 Reachability Analyzer 的更多信息，请参阅《Reachability Analyzer 指南》中的开始使用 Reachability Analyzer。

运行 license-daemon

在 Windows Enterprise Developer 中，从命令提示符下使用以下命令：

```
“C:\Program Files (x86)\Micro Focus\Enterprise Developer\AdoptOpenJDK\bin\java” -jar
 "C:\Program Files (x86)\Micro Focus\Licensing\aws-license-daemon.jar"
```

然后，查看输出。忽略 SLF4J 消息并查找第一个异常。

在 Enterprise Analyzer 中，从命令提示符下使用以下命令：

```
"C:\Program Files (x86)\Micro Focus\AdoptOpenJDK\bin\java" -jar "C:\Program Files
 (x86)\Micro Focus\Licensing\aws-license-daemon.jar"
```

然后，查看输出。忽略 SLF4J 消息并查找第一个异常。

在 Linux 上，运行：

```
java -jar /var/microfocuslicensing/bin/aws-license-daemon.jar
```

忽略 SLF4J 消息并查找第一个异常。

例如，如果 Amazon S3 资源不可用，则异常如下所示：

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Exception in thread "main" software.amazon.awssdk.services.s3.model.S3Exception: Access
 Denied (Service: S3, Status Code: 403, Request ID: P6
```

异常消息会指示哪个资源不可用。将配置值与本主题中显示的值进行比较。

教程：设置 AppStream 2.0 以与 Micro Focus Enterprise Analyzer 和 Micro Focus Enterprise Developer 搭配使用

AWS Mainframe Modernization 通过 Amazon AppStream 2.0 提供多种工具。AppStream 2.0 是一项完全托管的安全应用程序流式传输服务，可将桌面应用程序流式传输给用户，而无需重写这些应用程序。AppStream 2.0 使用户能够即时访问所需的应用程序，并在选定设备上获得流畅的响应式用户体验。使用 AppStream 2.0 托管特定于运行时引擎的工具，使客户应用程序团队能够直接通过 Web 浏览器使用这些工具，与存储在 Amazon S3 存储桶或 CodeCommit 存储库中的应用程序文件进行交互。

有关 AppStream 2.0 中浏览器支持的信息，请参阅《Amazon AppStream 2.0 管理指南》中的系统要求和特征支持（Web 浏览器）。如果您在使用 AppStream 2.0 时遇到问题，请参阅《Amazon AppStream 2.0 管理指南》中的 AppStream 2.0 用户问题疑难解答。

本文档适用于客户运营团队的成员。它描述了如何设置 Amazon AppStream 2.0 实例集和堆栈来托管与 AWS Mainframe Modernization 搭配使用的 Micro Focus Enterprise Analyzer 和 Micro Focus

Enterprise Developer 工具。在 AWS Mainframe Modernization 方法中，Micro Focus Enterprise Analyzer 通常用于评测阶段，Micro Focus Enterprise Developer 通常用于迁移与现代化阶段。如果您计划同时使用 Enterprise Analyzer 和 Enterprise Developer，则必须为每个工具创建单独的实例集和堆栈。每个工具都需要各自的实例集和堆栈，因为它们的许可条款不同。

> **Important**
>
> 本教程中的步骤基于可下载的 AWS CloudFormation 模板 cfn-m2-appstream-fleet-ea-ed.yml。

主题

先决条件

- 下载模板：cfn-m2-appstream-fleet-ea-ed.yml。
- 获取您的默认 VPC 的 ID 和安全组。有关默认 VPC 的更多信息，请参阅《Amazon VPC 用户指南》中的默认 VPC。有关默认安全组的更多信息，请参阅《适用于 Linux 实例的 Amazon EC2 用户指南》中的默认和自定义安全组。
- 请确保您具有以下权限：
  - 在 AppStream 2.0 中创建堆栈、实例集、用户。
  - 在 AWS CloudFormation 中使用模板创建堆栈。
  - 在 Amazon S3 中创建存储桶和并将文件上传到存储桶
  - 从 IAM 下载凭证（`access_key_id` 和 `secret_access_key`）。

步骤 1：获取 AppStream 2.0 映像

在此步骤中，您使用您的 AWS 账户为 Enterprise Analyzer 和 Enterprise Developer 共享 AppStream 2.0 映像。

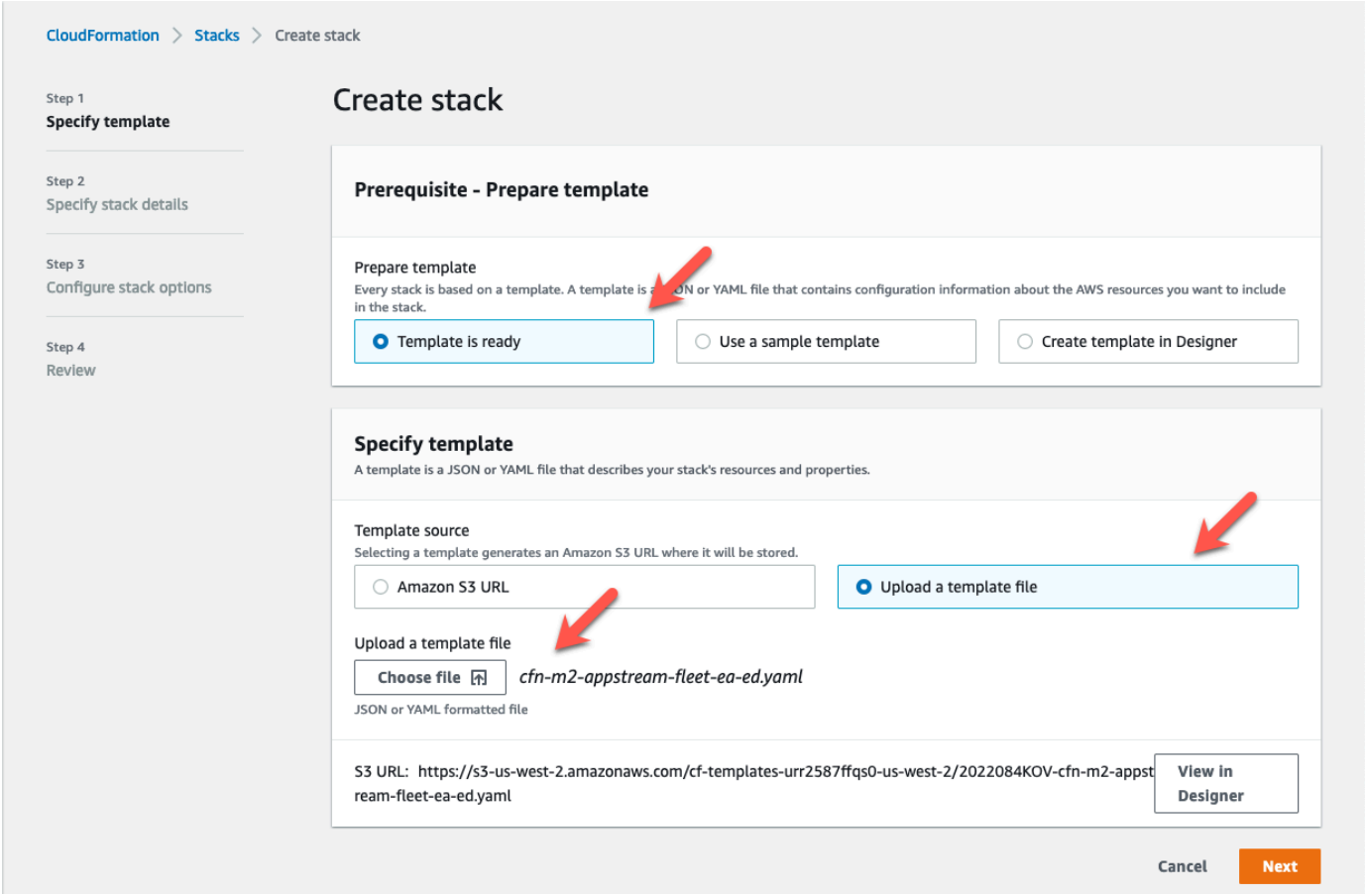
1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在左侧导航中，选择工具。
3. 在分析、开发和构建资产中，选择使用我的 AWS 账户共享资产。

步骤 2：使用 AWS CloudFormation 模板创建堆栈

在此步骤中，您将使用下载的 AWS CloudFormation 模板创建用于运行 Micro Focus Enterprise Analyzer 的 AppStream 2.0 堆栈和实例集。您可以稍后重复此步骤来创建另一个 AppStream 2.0 堆栈和实例集来运行 Micro Focus Enterprise Developer，因为 AppStream 2.0 中的每个工具都需要自己的实例集和堆栈。有关 AWS CloudFormation 堆栈的更多信息，请参阅《AWS CloudFormation 用户指南》中的使用堆栈。

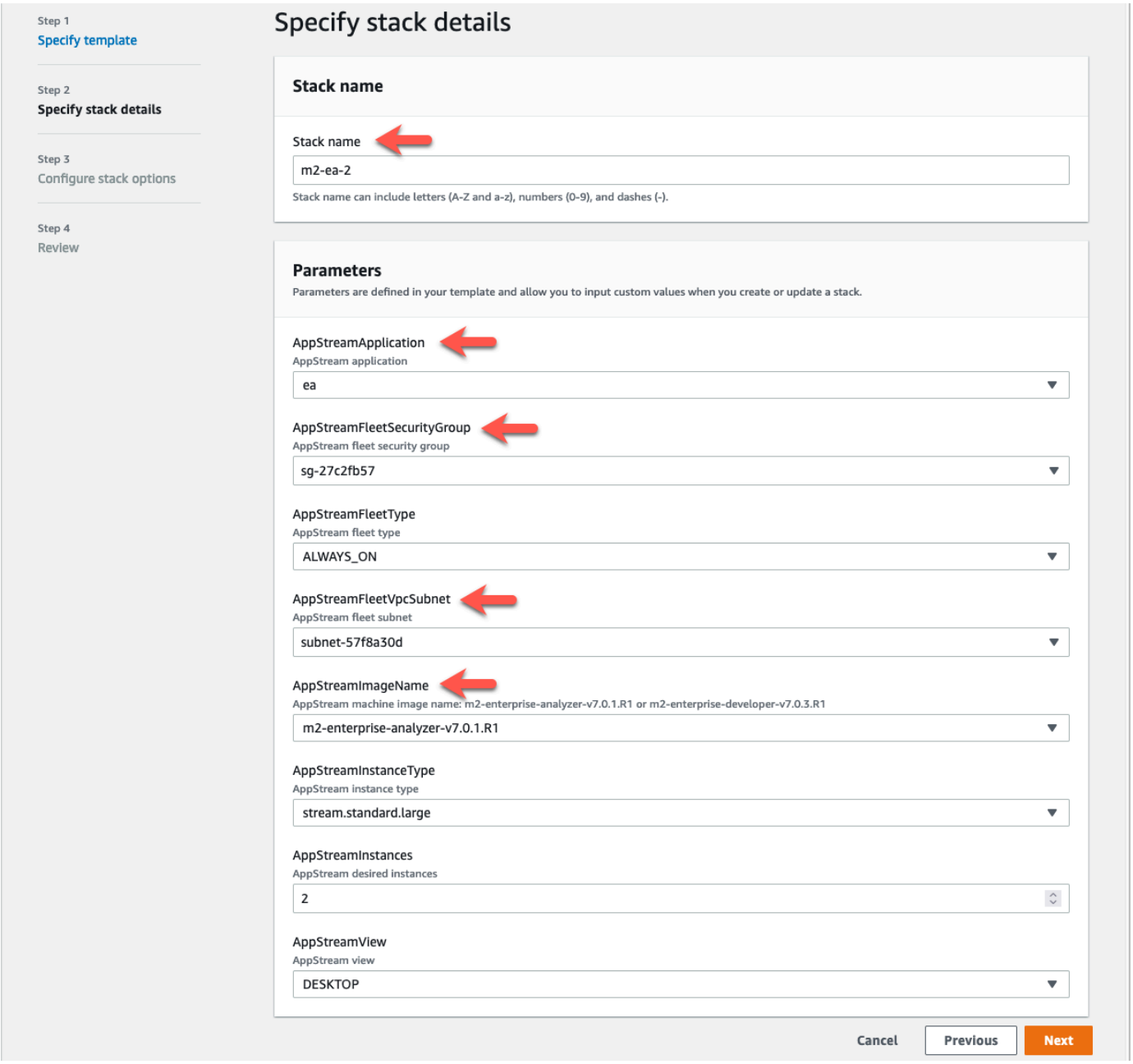
> **Note**
>
> AWS Mainframe Modernization 针对 Enterprise Analyzer 和 Enterprise Developer 在标准的 AppStream 2.0 定价基础上增加了额外的费用。有关更多信息，请参阅 AWS Mainframe Modernization 定价。

1. 下载 cfn-m2-appstream-fleet-ea-ed.yml 模板（如果需要）。
2. 打开 AWS CloudFormation 控制台，选择创建堆栈并使用新资源（标准）。
3. 在先决条件 – 准备模板中，选择模板准备就绪。
4. 在指定模板中，选择上传模板文件。
5. 在上传模板文件中，选择选择文件并上传 cfn-m2-appstream-fleet-ea-ed.yml 模板。
6. 选择下一步。



7. 在指定堆栈集详细信息页面上，输入以下信息：

   - 在堆栈名称中，输入您所选的名称。例如，**`m2-ea`**。
   - 在 AppStreamApplication中，选择 ea。
   - 在 AppStreamFleetSecurityGroup 中，选择默认 VPC 的默认安全组。
   - 在 AppStreamFleetVpcSubnet 中，选择默认 VPC 内的子网。
   - 在 AppStreamImageName 中，选择以 `m2-enterprise-analyzer` 开头的映像。此映像包含当前支持的 Micro Focus Enterprise Analyzer 工具版本。
   - 接受其他字段的默认值，然后选择下一步。



8. 接受所有默认值，然后选择下一步。
9. 在审核页面上，确保所有参数都符合您的预期。
10. 滚动到页面底部，选中我确认 AWS CloudFormation 可能会使用自定义名称创建 IAM 资源，然后选择创建堆栈。

创建堆栈和实例集需要 20 到 30 分钟。您可以选择刷新来查看发生的 AWS CloudFormation 事件。

步骤 3：在 AppStream 2.0 中创建用户

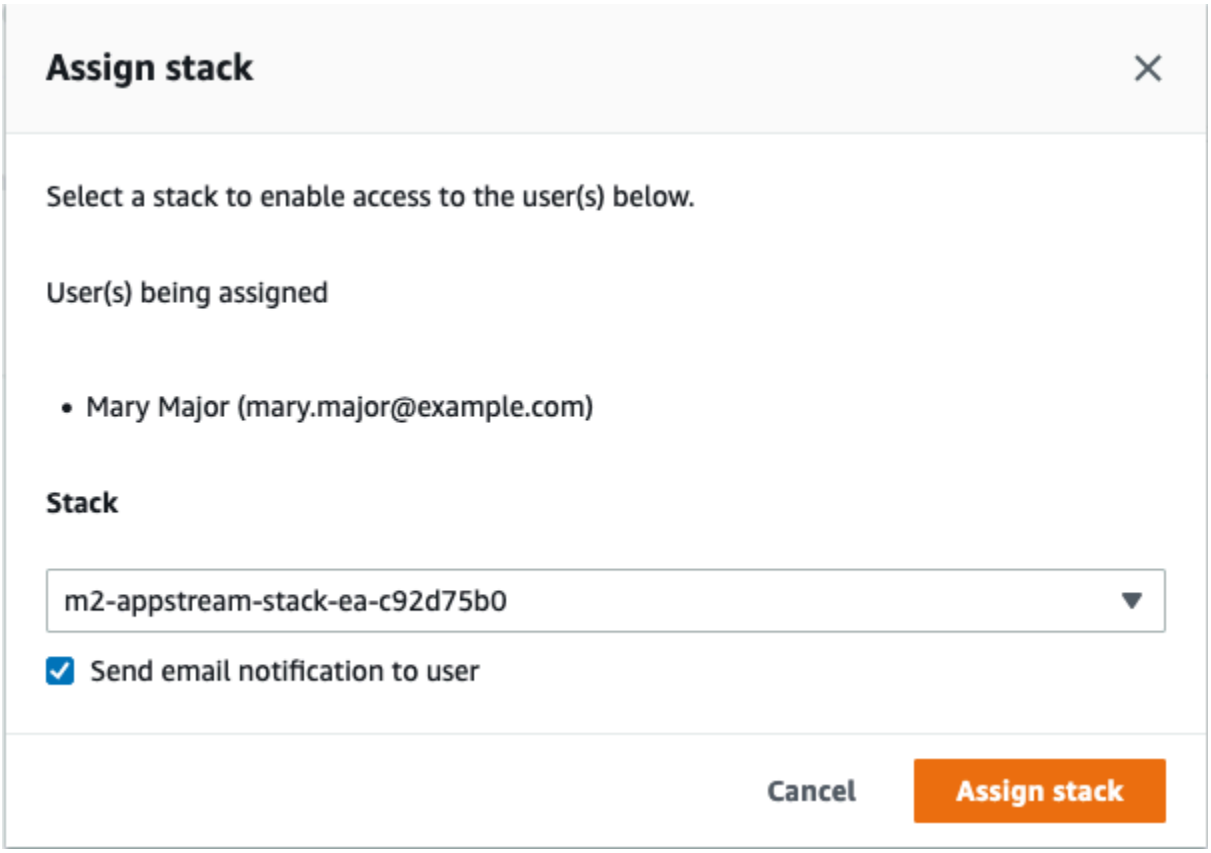
在等待 AWS CloudFormation 完成堆栈创建的过程中，可以在 AppStream 2.0 中创建一个或多个用户。这些用户即要在 AppStream 2.0 中使用 Enterprise Analyzer 的用户。您需要为每个用户指定一个电子邮件地址，并确保每个用户都有足够的权限在 Amazon S3 中创建存储桶、将文件上传到存储桶以及链接到存储桶以映射其内容。

1. 打开 AppStream 2.0 控制台。
2. 在左侧导航窗格中，选择 用户池。
3. 选择创建用户。
4. 提供用户用于接收使用 AppStream 2.0 的电子邮件邀请的电子邮件地址、名字和姓氏，然后选择创建用户。
5. 如有必要，请重复此步骤以创建更多用户。每个用户的电子邮件地址必须是唯一的。

有关创建 AppStream 2.0 的更多信息，请参阅《Amazon AppStream 2.0 管理指南》中的AppStream 2.0 用户池。

AWS CloudFormation 创建完堆栈后，您可以将创建的用户分配给堆栈，如下所示：

1. 打开 AppStream 2.0 控制台。
2. 选择用户名。
3. 依次选择操作和分配堆栈。
4. 在分配堆栈中，选择以 `m2-appstream-stack-ea` 开头的堆栈。
5. 选择分配堆栈。



将用户分配到堆栈会触发 AppStream 2.0 通过您提供的地址向该用户发送电子邮件。此电子邮件包含指向 AppStream 2.0 登录页面的链接。

步骤 4：登录 AppStream 2.0

在此步骤中，您将使用 AppStream 2.0 发送给您在步骤 3：在 AppStream 2.0 中创建用户中创建的用户的电子邮件中的链接登录到 AppStream 2.0。

1. 使用 AppStream 2.0 发送的电子邮件中提供的链接登录 AppStream 2.0。
2. 如果出现提示，请更改密码。您看到的 AppStream 2.0 屏幕与以下屏幕类似：

3. 选择桌面。
4. 在任务栏上，选择搜索，然后输入 **`D:`** 以导航到主文件夹。

> **Note**
>
> 如果跳过此步骤，则在尝试访问主文件夹时可能会出现`Device not ready`错误。

如果您在任何时候登录 AppStream 2.0 时遇到问题，都可以按照以下步骤重启您的 AppStream 2.0 实例集并尝试重新登录。

1. 打开 AppStream 2.0 控制台。
2. 在左侧导航窗格中，选择实例集。
3. 选择您要使用的实例集。
4. 选择操作，然后选择停止。
5. 等待实例集停止。
6. 选择操作，然后选择启动。

此过程大约需要 10 分钟。

步骤 5：验证 Amazon S3 中的存储桶（可选）

您用于创建堆栈的 AWS CloudFormation 模板完成的任务之一是在 Amazon S3 中创建两个存储桶，这些存储桶是跨工作会话保存和恢复用户数据和应用程序设置所必需的。这些存储桶如下所示：

- 名称以 `appstream2-` 开头。此存储桶将数据映射到 AppStream 2.0 中的主文件夹 (`D:\PhotonUser\My Files\Home Folder`)。

> **Note**
>
> 对于给定的电子邮件地址，主文件夹是唯一的，并且在给定 AWS 帐户中的所有实例集和堆栈之间共享。主文件夹的名称是用户电子邮件地址的 SHA256 哈希值，存储在基于该哈希值的路径上。

- 名称以 `appstream-app-settings-` 开头。此存储桶包含 AppStream 2.0 的用户会话信息，还包含浏览器收藏夹、IDE 和应用程序连接配置文件以及 UI 自定义等设置。有关更多信息，请参阅《Amazon AppStream 2.0 管理指南》中的应用程序设置持久性如何工作。

要验证存储桶是否已创建，请执行以下步骤：

1. 打开 Amazon S3 控制台。
2. 在左侧导航中，选择存储桶。
3. 在按名称查找存储桶中，输入 **`appstream`** 以筛选列表。

如果您看到存储桶，则无需采取进一步的操作。请注意存储桶存在即可。如果您没有看到存储桶，则可能是 AWS CloudFormation 模板未完成运行，或者发生了错误。转到 AWS CloudFormation 控制台并查看堆栈创建消息。

后续步骤

设置完 AppStream 2.0 基础设施后，即可设置并开始使用 Enterprise Analyzer。有关更多信息，请参阅教程：在 AppStream 2.0 上设置 Enterprise Analyzer。您还可以设置 Enterprise Developer。有关更多信息，请参阅教程：在 AppStream 2.0 上设置 Micro Focus Enterprise Developer。

清理资源

有关清理已创建堆栈和实例集的过程，请参阅创建 AppStream 2.0 实例集和堆栈。

删除 AppStream 2.0 对象后，账户管理员还可以在适合的情况下清理应用程序设置和主文件夹的 S3 存储桶。

> **Note**
>
> 给定用户的主文件夹在所有实例集中都是唯一的，因此，如果同一个账户中的其他 AppStream 2.0 堆栈处于活动状态，则可能需要保留该文件夹。

最后，AppStream 2.0 目前不允许您使用控制台删除用户。而是必须使用服务 API 和 CLI 进行删除。有关更多信息，请参阅 《Amazon AppStream 2.0 管理指南》中的用户池管理。

教程：在 AppStream 2.0 上设置 Enterprise Analyzer

本教程介绍如何设置 Micro Focus Enterprise Analyzer 来分析一个或多个大型机应用程序。Enterprise Analyzer 工具可根据其对应用程序源代码和系统定义的分析提供多个报告。

此设置旨在促进团队协作。安装使用 Amazon S3 存储桶与虚拟磁盘共享源代码。这样做会在 Windows 计算机上使用 Rclone。借助运行 PostgreSQL 的常用 Amazon RDS 实例，团队中的任何成员都可以访问所有请求的报告。

团队成员还可以将 Amazon S3 支持的虚拟磁盘挂载到个人机器上，并从其工作站更新源存储桶。如果他们连接到其他本地内部系统，还可以在其机器上使用脚本或任何其他形式的自动化。

该设置基于 AWS Mainframe Modernization 与客户共享的 AppStream 2.0 Windows 映像，以及创建的 AppStream 2.0 实例集和堆栈（如教程：设置 AppStream 2.0 以与 Micro Focus Enterprise Analyzer 和 Micro Focus Enterprise Developer 搭配使用中所述）。

> **Important**
>
> 本教程中的步骤假设您使用可下载的 AWS CloudFormation 模板 cfn-m2-appstream-fleet-ea-ed.yml 来设置 AppStream 2.0。有关更多信息，请参阅教程：设置 AppStream 2.0 以与 Micro Focus Enterprise Analyzer 和 Micro Focus Enterprise Developer 搭配使用。
> 要执行本教程中的步骤，您必须设置 Enterprise Analyzer 实例集和堆栈，并且它们必须正在运行。

有关 Enterprise Analyzer 功能和交付项的完整描述，请参阅 Micro Focus 网站上的 Enterprise Analyzer 文档。

映像内容

除了 Enterprise Analyzer 应用程序本身之外，该映像还包含以下工具和库。

第三方工具

- Python
- Rclone
- pgAdmin
- git-scm
- PostgreSQL ODBC 驱动程序

`C:\Users\Public` 中的库

- Enterprise Developer 的 BankDemo 源代码和项目定义：`m2-bankdemo-template.zip`。
- 大型机的 MFA 安装包：`mfa.zip`。有关更多信息，请参阅《Micro Focus Enterprise Developer》文档中的大型机访问权限概述。
- Rclone 的命令和配置文件（教程中的使用说明）：`m2-rclone.cmd` 和 `m2-rclone.conf`。

主题

先决条件

- 将您要分析的客户应用程序的源代码和系统定义上传到 S3 存储桶。系统定义包括 CICS CSD、DB2 对象定义等。您可以在存储桶中创建一个文件夹结构，该结构表示您要如何组织应用程序构件。例如，当您解压缩 BankDemo 示例时，其结构如下：

```
demo
     |--> jcl
     |--> RDEF
     |--> transaction
     |--> xa
```

- 创建并启动运行 PostgreSQL 的 Amazon RDS 实例 此实例将存储 Enterprise Analyzer 生成的数据和结果。您可以与应用程序团队的所有成员共享此实例。此外，在数据库中创建一个名为 m2_ea（或任何其他合适的名称）的空架构。为授权用户定义凭证，允许其创建、插入、更新和删除此架构中的项目。您可以从 Amazon RDS 控制台或从账户管理员处获取数据库名称、服务器端点 URL 和 TCP 端口。
- 确保您已设置了对您的 AWS 账户的编程访问。有关更多信息，请参阅《Amazon Web Services 一般参考》中的编程访问。

步骤 1：设置

1. 使用您收到的 AppStream 2.0 发送的欢迎电子邮件中的 URL，开始与 AppStream 2.0 的会话。
2. 使用您的电子邮件作为用户 ID，并定义您的永久密码。
3. 选择 Enterprise Analyzer 堆栈。
4. 在 AppStream 2.0 菜单页面上，选择桌面 以访问实例集正在进行流式传输的 Windows 桌面。

步骤 2：在 Windows 上创建基于 Amazon S3 的虚拟文件夹

> **Note**
>
> 如果您在 AWS Mainframe Modernization 预览版中已经使用过 Rclone，则必须将 m2-rclone.cmd 更新到位于 C:\Users\Public 中的较新版本。

1. 使用文件资源管理器将 C:\Users\Public 中提供的 m2-rclone.conf 和 m2-rclone.cmd 文件复制到您的主文件夹 C:\Users\PhotonUser\My Files\Home Folder。
2. 使用您的 AWS 访问密钥和相应的密钥以及 AWS 区域来更新 m2-rclone.conf 配置参数。

```
[m2-s3]
type = s3
```

```
provider = AWS
access_key_id = YOUR-ACCESS-KEY
secret_access_key = YOUR-SECRET-KEY
region = YOUR-REGION
acl = private
server_side_encryption = AES256
```

3. 在 m2-rclone.cmd 中，进行以下更改：

   - 将 your-s3-bucket 更改为您的 Amazon S3 存储桶名称。例如，m2-s3-mybucket。
   - 将 your-s3-folder-key 更改为您的 Amazon S3 存储桶密钥。例如，myProject。
   - 将 your-local-folder-path 更改为要从包含应用程序文件的 Amazon S3 存储桶同步应用程序文件的目录路径。例如，D:\PhotonUser\My Files\Home Folder\m2-new。此同步目录必须是主文件夹的子目录，AppStream 2.0 才能在会话开始和结束时对其进行正确备份和恢复。

```
:loop
timeout /T 10
"C:\Program Files\rclone\rclone.exe" sync m2-s3:your-s3-bucket/your-s3-folder-
key "D:\PhotonUser\My Files\Home Folder\your-local-folder-path" --config "D:
\PhotonUser\My Files\Home Folder\m2-rclone.conf"
goto :loop
```

4. 打开 Windows 命令提示符，使用 cd 命令切换到 C:\Users\PhotonUser\My Files\Home Folder（如果需要），然后运行 m2-rclone.cmd。此命令脚本连续循环运行，每 10 秒将您的 Amazon S3 存储桶和密钥同步到本地文件夹。您可以根据需要调整超时。您应该在 Windows 文件资源管理器的 Amazon S3 存储桶中看到该应用程序的源代码。

要将新文件添加到您正在处理的文件集中或更新现有文件，请将文件上传到 Amazon S3 存储桶，这些文件将在 m2-rclone.cmd 中定义的下一次迭代中同步到您的目录。同样，如果要删除某些文件，请将其从 Amazon S3 存储桶中删除。下次同步操作会将其从您的本地目录中删除。

步骤 3：为 Amazon RDS 实例创建 ODBC 源

1. 要启动 EA_Admin 工具，请导航到浏览器窗口左上角的应用程序选择器菜单，然后选择 MF EA_Admin。
2. 从管理菜单中，选择 ODBC 数据来源，然后从用户 DSN 选项卡中选择添加。
3. 在“创建新数据来源”对话框中，选择 PostgreSQL Unicode 驱动程序，然后选择完成。

4. 在 PostgreSQL Unicode ODBC 驱动程序（psqlODBC）设置对话框中，定义并记下所需的数据来源名称。使用您之前创建的 RDS 实例中的值填写以下参数：

   描述

   可选描述，可帮助您快速识别此数据库连接。

   数据库

   您之前创建的 Amazon RDS 数据库。

   服务器

   Amazon RDS 端点。

   端口

   Amazon RDS 端口。

   用户名称

   Amazon RDS 实例中所定义的用户名称。

   密码

   Amazon RDS 实例中所定义的密码。

5. 选择测试以验证与 Amazon RDS 的连接是否成功，然后选择保存以保存您的新用户 DSN。
6. 等待直到看到确认创建了正确工作区的消息，然后选择确定以完成 ODBC 数据来源操作并关闭 EA_Admin 工具。
7. 再次导航到应用程序选择器菜单，然后选择“Enterprise Analyzer”以启动该工具。选择新建。
8. 在工作区配置窗口中，输入您的工作区名称并定义其位置。如果您在此配置下工作，该工作区可以是基于 Amazon S3 的磁盘，也可以是您的主文件夹（如果需要）。
9. 选择选择其他数据库以连接到您的 Amazon RDS 实例。
10. 从选项中选择 Postgre 图标，然后选择确定。
11. 在选项 – 定义连接参数下的 Windows 设置中，输入您创建的数据来源的名称。此外，输入数据库名称、架构名称、用户名和密码。选择确定。
12. 等待 Enterprise Analyzer 创建用于存储结果所需的所有表、索引等。此过程可能需要几分钟时间。Enterprise Analyzer 会确认数据库和工作区何时准备就绪可供使用。
13. 再次导航到应用程序选择器菜单，然后选择“Enterprise Analyzer”以启动该工具。
14. Enterprise Analyzer 启动窗口出现在新的所选工作区位置中。选择确定。

15. 在左侧窗格中，导航到存储库，选择存储库名称，然后选择向工作区添加文件/文件夹。选择存储应用程序代码的文件夹，将其添加到工作区。如果需要，您可以使用之前的 BankDemo 示例代码。Enterprise Analyzer 提示您验证这些文件时，请选择验证以启动初始 Enterprise Analyzer 验证报告。该过程可能需要几分钟才能完成，具体取决于您的应用程序的大小。
16. 展开工作区，以便查看已添加到工作区的文件和文件夹。对象类型和圈复杂度报告也显示在图表查看器窗格的上象限中。

现在，您可以使用 Enterprise Analyzer 来完成所有需要的任务。

后续会话

1. 使用您收到的 AppStream 2.0 发送的欢迎电子邮件中的 URL，开始与 AppStream 2.0 的会话。
2. 使用您的电子邮件和永久密码登录。
3. 选择 Enterprise Analyzer 堆栈。
4. 启动 `Rclone` 以连接到 Amazon S3 支持的磁盘（如果您使用此选项共享工作区文件）。
5. 启动 Enterprise Analyzer 来完成任务。

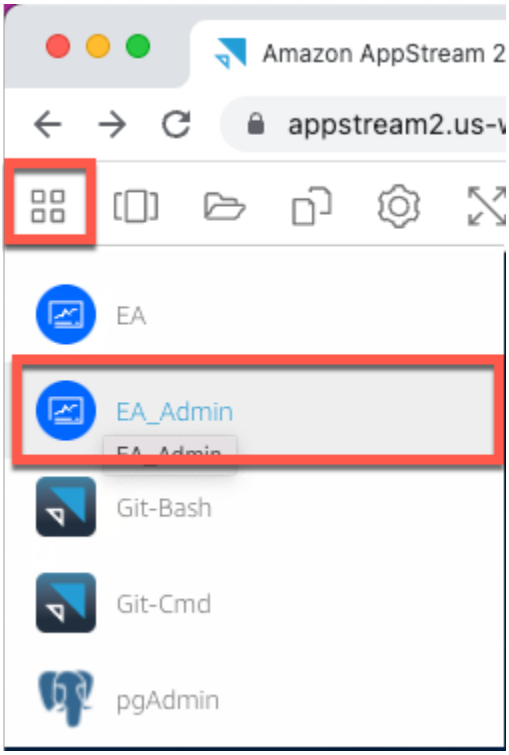
对工作区连接进行故障排除

当您尝试重新连接到 Enterprise Analyzer 工作区时，可能会看到类似如下错误：

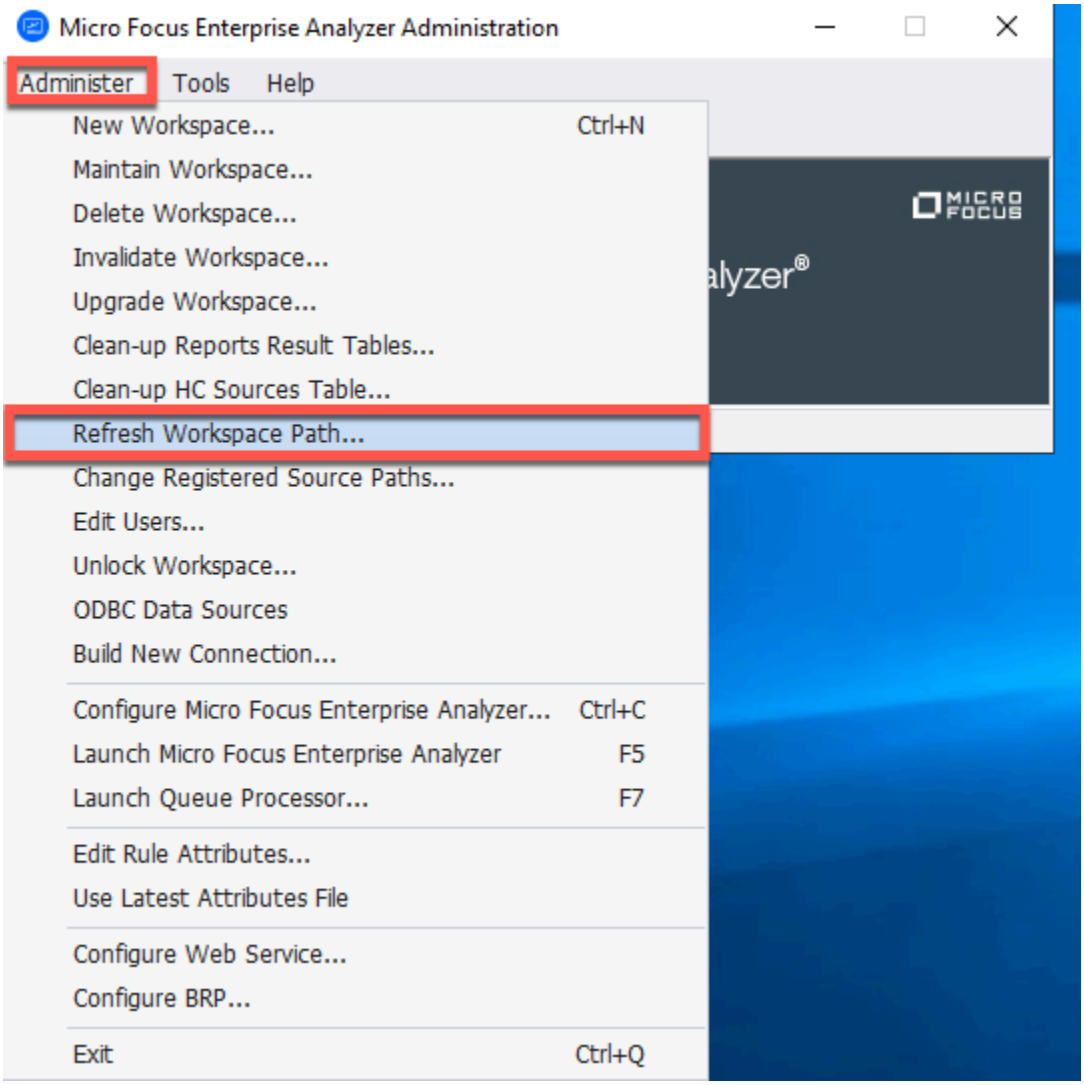
```
Cannot access the workspace directory D:\PhotonUser\My Files\Home Folder\EA_BankDemo.
 The workspace has been created on a non-shared disk of the EC2AMAZ-E6LC33H computer.
 Would you like to correct the workspace directory location?
```

要解决此问题，请选择确定来清除消息，然后完成以下步骤。

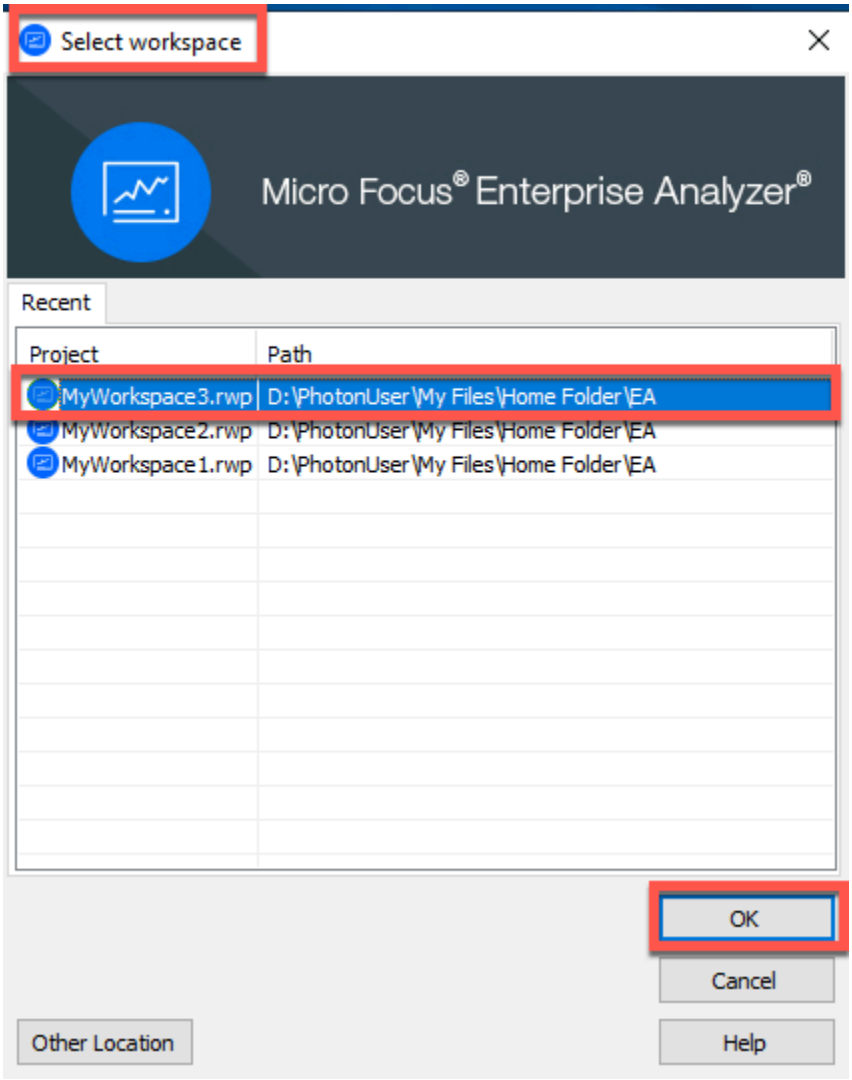
1. 在 AppStream 2.0 中，选择工具栏上的启动应用程序图标，然后选择 EA_Admin 来启动 Micro Focus Enterprise Analyzer Administration 工具。



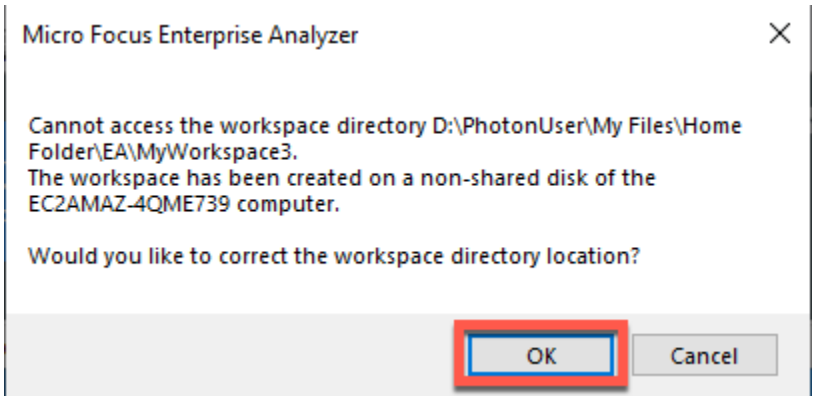
2. 从管理菜单中，选择刷新工作区路径...。



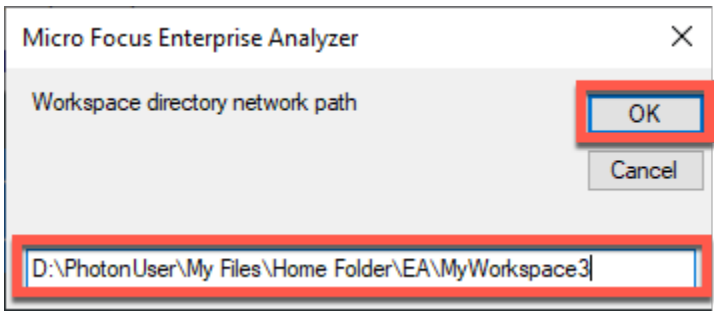
3. 在选择工作区下，选择所需的工作区，然后选择确定。



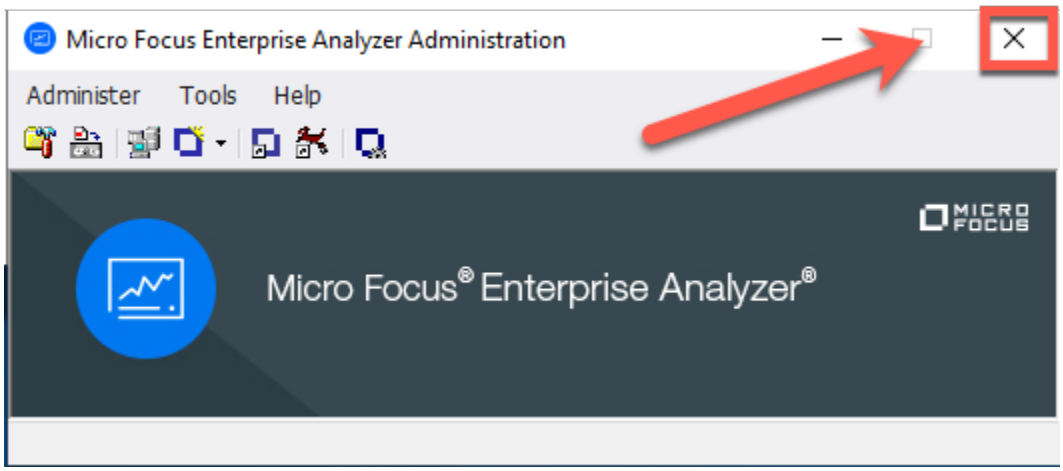
4. 选择确定，确认错误消息。



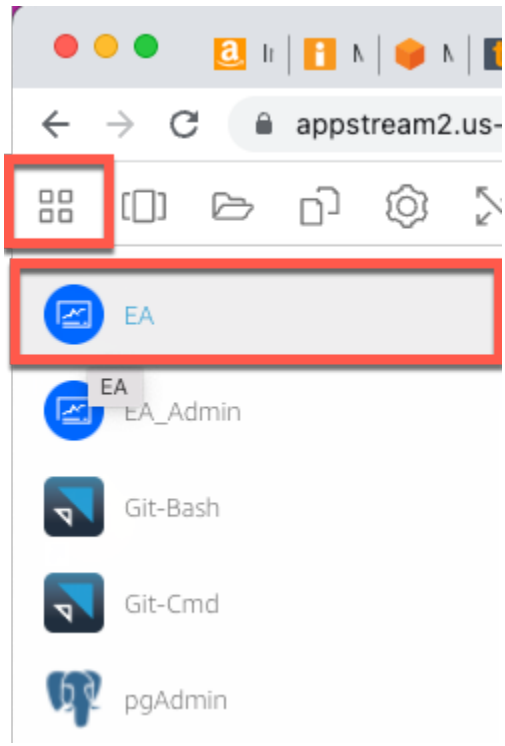
5. 在工作区目录网络路径下，输入工作区的正确路径，例如 `D:\PhotonUser\My Files\Home Folder\EA\MyWorkspace3`。



6. 关闭 Micro Focus Enterprise Analyzer Administration 工具。



7. 在 AppStream 2.0 中，选择工具栏上的启动应用程序图标，然后选择 EA 来启动 Micro Focus Enterprise Analyzer。



8. 重复步骤 3 – 5。

Micro Focus Enterprise Analyzer 此时应可使用现有工作区打开。

清理资源

如果您不再需要为本教程创建的资源，请将其删除，以免继续产生费用。完成以下步骤：

- 使用 EA_Admin 工具删除工作区。
- 删除您为本教程创建的 S3 存储桶。有关更多信息，请参阅《Amazon S3 用户指南》中的删除存储桶。
- 选择您为本教程创建的数据库。有关更多信息，请参阅删除数据库实例。

教程：在 AppStream 2.0 上设置 Micro Focus Enterprise Developer

本教程介绍如何为一个或多个大型机应用程序设置 Micro Focus Enterprise Developer，以便使用 Enterprise Developer 特征对应用程序进行维护、编译和测试。该设置基于 AWS Mainframe Modernization 与客户共享的 AppStream 2.0 Windows 映像，以及创建的 AppStream 2.0 实例集和堆栈（如教程：设置 AppStream 2.0 以与 Micro Focus Enterprise Analyzer 和 Micro Focus Enterprise Developer 搭配使用中所述）。

> **Important**
>
> 本教程中的步骤假设您使用可下载的 AWS CloudFormation 模板 cfn-m2-appstream-fleet-ea-ed.yaml 来设置 AppStream 2.0。有关更多信息，请参阅教程：设置 AppStream 2.0 以与 Micro Focus Enterprise Analyzer 和 Micro Focus Enterprise Developer 搭配使用。
> 如果 Enterprise Developer 实例集和堆栈已启动并正在运行时，则必须执行此设置的步骤。

有关 Enterprise Developer v7 特征和交付项的完整描述，请在 Micro Focus 网站上查看其最新在线文档 (v7.0)。

映像内容

除了 Enterprise Developer 本身，映像还包含包含 Rumba（TN3270 仿真器）的映像。此外，它还包含以下工具和库。

第三方工具

- Python
- Rclone
- pgAdmin
- git-scm
- PostgreSQL ODBC 驱动程序

`C:\Users\Public` 中的库

- Enterprise Developer 的 BankDemo 源代码和项目定义：`m2-bankdemo-template.zip`。
- 大型机的 MFA 安装包：`mfa.zip`。有关更多信息，请参阅《Micro Focus Enterprise Developer》文档中的大型机访问权限概述。
- Rclone 的命令和配置文件（教程中的使用说明）：`m2-rclone.cmd` 和 `m2-rclone.conf`。

如果您需要访问尚未加载到 CodeCommit 存储库但在 Amazon S3 存储桶中可用的源代码，例如要将源代码初始加载到 git 中，请按照教程：在 AppStream 2.0 上设置 Enterprise Analyzer 中所述的步骤创建虚拟 Windows 磁盘。

主题

- 先决条件
- 步骤 1：由个人 Enterprise Developer 用户设置
- 步骤 2：在 Windows 上创建基于 Amazon S3 的虚拟文件夹（可选）
- 步骤 3：克隆存储库
- 后续会话
- 清理资源

先决条件

- 一个或多个 CodeCommit 存储库，其中加载了待维护的应用程序的源代码。存储库设置应符合上述 CI/CD 管道的要求，以便通过结合使用这两种工具来实现协同效应。
- 每个用户都必须拥有账户管理员根据 AWS CodeCommit 的身份验证和访问控制中的信息定义的凭证，用于访问 CodeCommit 存储库。有关这些凭证的结构，请参阅 AWS CodeCommit 的身份验证和访问控制，有关 CodeCommit 的 IAM 授权的完整参考，请参阅 CodeCommit 权限参考：管理员可以为不同的角色定义不同的 IAM 策略，这些策略具有特定于每个存储库的角色的凭证，并将对用

户的授权限制为用户必须在给定存储库上完成的特定任务集。因此，账户管理员将为 CodeCommit 存储库的每位维护者生成一个主用户，并通过选择适用于 CodeCommit 访问的一个或多个 IAM 策略，授予该用户访问所需的一个或多个存储库的权限。

步骤 1：由个人 Enterprise Developer 用户设置

1. 获取您的 IAM 凭证：
    1. 连接到 AWS 控制台，网址为 https://console.aws.amazon.com/iam/。
    2. 按照《AWS CodeCommit 用户指南》中适用于使用 Git 凭证的 HTTPS 用户的设置的步骤 3 中描述的过程操作。
    3. 通过以下两种方式之一复制 IAM 为您生成的 CodeCommit 特定登录凭证：显示这些信息，然后将其复制并粘贴到本地计算机上安全的文件中；或选择下载凭证)，将这些信息下载为 .CSV 文件。您需要这些信息才能连接到 CodeCommit。
2. 根据收到的欢迎电子邮件中的 URL 开始与 AppStream 2.0 的会话。使用您的电子邮件作为用户名并创建密码。
3. 选择您的 Enterprise Developer 堆栈。
4. 在菜单页面上，选择桌面以访问实例集正在进行流式传输的 Windows 桌面。

步骤 2：在 Windows 上创建基于 Amazon S3 的虚拟文件夹（可选）

如果需要 Rclone（见上文），请在 Windows 上创建基于 Amazon S3 的虚拟文件夹：（如果所有应用程序构件都完全来自 CodeCommit 访问，则为可选）。

> **Note**
>
> 如果您在 AWS Mainframe Modernization 预览版中已经使用过 Rclone，则必须将 `m2-rclone.cmd` 更新到位于 `C:\Users\Public` 中的较新版本。

1. 使用文件资源管理器将 `C:\Users\Public` 中提供的 `m2-rclone.conf` 和 `m2-rclone.cmd` 文件复制到您的主文件夹 `C:\Users\PhotonUser\My Files\Home Folder`。
2. 使用您的 AWS 访问密钥和相应的密钥以及 AWS 区域来更新 `m2-rclone.conf` 配置参数。

```
[m2-s3]
type = s3
```

```
provider = AWS
access_key_id = YOUR-ACCESS-KEY
secret_access_key = YOUR-SECRET-KEY
region = YOUR-REGION
acl = private
server_side_encryption = AES256
```

3. 在 m2-rclone.cmd 中，进行以下更改：

   - 将 your-s3-bucket 更改为您的 Amazon S3 存储桶名称。例如，m2-s3-mybucket。
   - 将 your-s3-folder-key 更改为您的 Amazon S3 存储桶密钥。例如，myProject。
   - 将 your-local-folder-path 更改为要从包含应用程序文件的 Amazon S3 存储桶同步应用程序文件的目录路径。例如，D:\PhotonUser\My Files\Home Folder\m2-new。此同步目录必须是主文件夹的子目录，AppStream 2.0 才能在会话开始和结束时对其进行正确备份和恢复。

```
:loop
timeout /T 10
"C:\Program Files\rclone\rclone.exe" sync m2-s3:your-s3-bucket/your-s3-folder-
key "D:\PhotonUser\My Files\Home Folder\your-local-folder-path" --config "D:
\PhotonUser\My Files\Home Folder\m2-rclone.conf"
goto :loop
```

4. 打开 Windows 命令提示符，使用 cd 命令切换到 C:\Users\PhotonUser\My Files\Home Folder（如果需要），然后运行 m2-rclone.cmd。此命令脚本连续循环运行，每 10 秒将您的 Amazon S3 存储桶和密钥同步到本地文件夹。您可以根据需要调整超时。您应该在 Windows 文件资源管理器的 Amazon S3 存储桶中看到该应用程序的源代码。

要将新文件添加到您正在处理的文件集中或更新现有文件，请将文件上传到 Amazon S3 存储桶，这些文件将在 m2-rclone.cmd 中定义的下一次迭代中同步到您的目录。同样，如果要删除某些文件，请将其从 Amazon S3 存储桶中删除。下次同步操作会将其从您的本地目录中删除。

步骤 3：克隆存储库

1. 导航到浏览器窗口左上角的应用程序选择器菜单，然后选择“Enterprise Developer”。
2. 在您的主文件夹中完成 Enterprise Developer 所需的工作区创建，选择C:\Users\PhotonUser\My Files\Home Folder（又或 D: \PhotonUser\My Files\Home Folder）作为工作区的位置。

3. 在 Enterprise Developer 中，进入项目资源管理器，右键单击并选择导入、导入...、Git、Git 克隆 URI 中的项目，克隆您的 CodeCommit 存储库。然后，输入您的 CodeCommit 特定登录凭证并完成 Eclipse 对话框以导入代码。

CodeCommit git 存储库现已克隆到您的本地工作区中。

您的 Enterprise Developer 工作区现已准备就绪，可以开始对您的应用程序进行维护工作。具体而言，您可以使用与 Enterprise Developer 集成的 Microfocus Enterprise Server (ES) 本地实例以交互方式调试和运行应用程序，以便在本地验证您的更改。

> **Note**
>
> 本地 Enterprise Developer 环境（包括本地 Enterprise Server 实例）在 Windows 下运行，而 AWS Mainframe Modernization 则在 Linux 下运行。我们建议您在将新应用程序提交到 CodeCommit 并针对此目标进行重建之后，以及将新应用程序部署到生产环境之前，在 AWS Mainframe Modernization 提供的 Linux 环境中运行补充测试。

后续会话

当您选择一个受 AppStream 2.0 管理的文件夹（例如用于克隆 CodeCommit 存储库的主文件夹）时，该文件夹将在各个会话之间透明地保存和恢复。您可以在下次需要使用该应用程序时完成以下步骤：

1. 根据收到的欢迎电子邮件中的 URL 开始与 AppStream 2.0 的会话。
2. 使用您的电子邮件和永久密码登录。
3. 选择 Enterprise Developer 堆栈。
4. 启动 `Rclone` 以连接（见上文）到 Amazon S3 支持的磁盘（如果您使用此选项共享工作区文件）。
5. 启动 Enterprise Developer 来完成工作。

清理资源

如果您不再需要为本教程创建的资源，请将其删除，以免继续产生费用。完成以下步骤：

- 删除您为本教程创建的 CodeCommit 存储库。有关更多信息，请参阅《AWS CodeCommit 用户指南》中的删除 CodeCommit 存储库。

- 删除您为本教程创建的数据库。有关更多信息，请参阅删除数据库实例。

为 Micro Focus Enterprise Analyzer 和 Micro Focus Enterprise Developer 流式传输会话设置自动化

您可以在会话开始和结束时自动运行脚本，从而实现特定于您的客户环境的自动化。有关此 AppStream 2.0 特征的更多信息，请参阅《Amazon AppStream 2.0 管理指南》中的使用会话脚本来管理您的 AppStream 2.0 用户的流式传输体验。

此特征要求您至少具有以下版本的 Enterprise Analyzer 和 Enterprise Developer 映像：

- `m2-enterprise-analyzer-v8.0.4.R1`
- `m2-enterprise-developer-v8.0.4.R1`

主题

设置会话开始时的自动化

如果要在用户连接到 AppStream 2.0 时运行自动化脚本，请创建脚本并将其命名为 `m2-user-setup.cmd`。将脚本存储在用户的 AppStream 2.0 主文件夹中。AWS Mainframe Modernization 提供的 AppStream 2.0 映像会在此位置查找具有该名称的脚本，如果脚本存在，则运行该脚本。

> **Note**
>
> 脚本持续时间不能超过 AppStream 2.0 设定的限制，目前为 60 秒。有关更多信息，请参阅《Amazon AppStream 2.0 管理指南》中的在流式传输会话开始前运行脚本。

设置会话结束时的自动化

如果要在用户从 AppStream 2.0 断开连接时运行自动化脚本，请创建脚本并将其命名为 `m2-user-teardown.cmd`。将脚本存储在用户的 AppStream 2.0 主文件夹中。AWS Mainframe Modernization 提供的 AppStream 2.0 映像会在此位置查找具有该名称的脚本，如果脚本存在，则运行该脚本。

> **Note**
>
> 脚本持续时间不能超过 AppStream 2.0 设定的限制，目前为 60 秒。有关更多信息，请参阅《Amazon AppStream 2.0 管理指南》中的在流式传输会话结束后运行脚本。

在 Enterprise Developer 中以表和列的形式查看数据集

您可以访问在使用 Micro Focus 运行时的 AWS Mainframe Modernization 中部署的大型机数据集。您可以从 Micro Focus Enterprise Developer 实例中以表和列的形式查看迁移的数据集。通过这种方式查看数据集，您可以：

- 对迁移的数据文件执行 SQL SELECT 操作。
- 在迁移的大型机应用程序外部披露数据，无需更改应用程序。
- 轻松筛选数据并保存为 CSV 或其他文件格式。

> **Note**
>
> 步骤 1 和 2 是一次性活动。针对每个数据集重复步骤 3 和 4，以便创建数据库视图。

主题

先决条件

- 您必须能够通过 AppStream 2.0 访问 Micro Focus Enterprise Developer 桌面。
- 您必须拥有在使用 Micro Focus 运行时引擎的 AWS Mainframe Modernization 下部署和运行的应用程序。

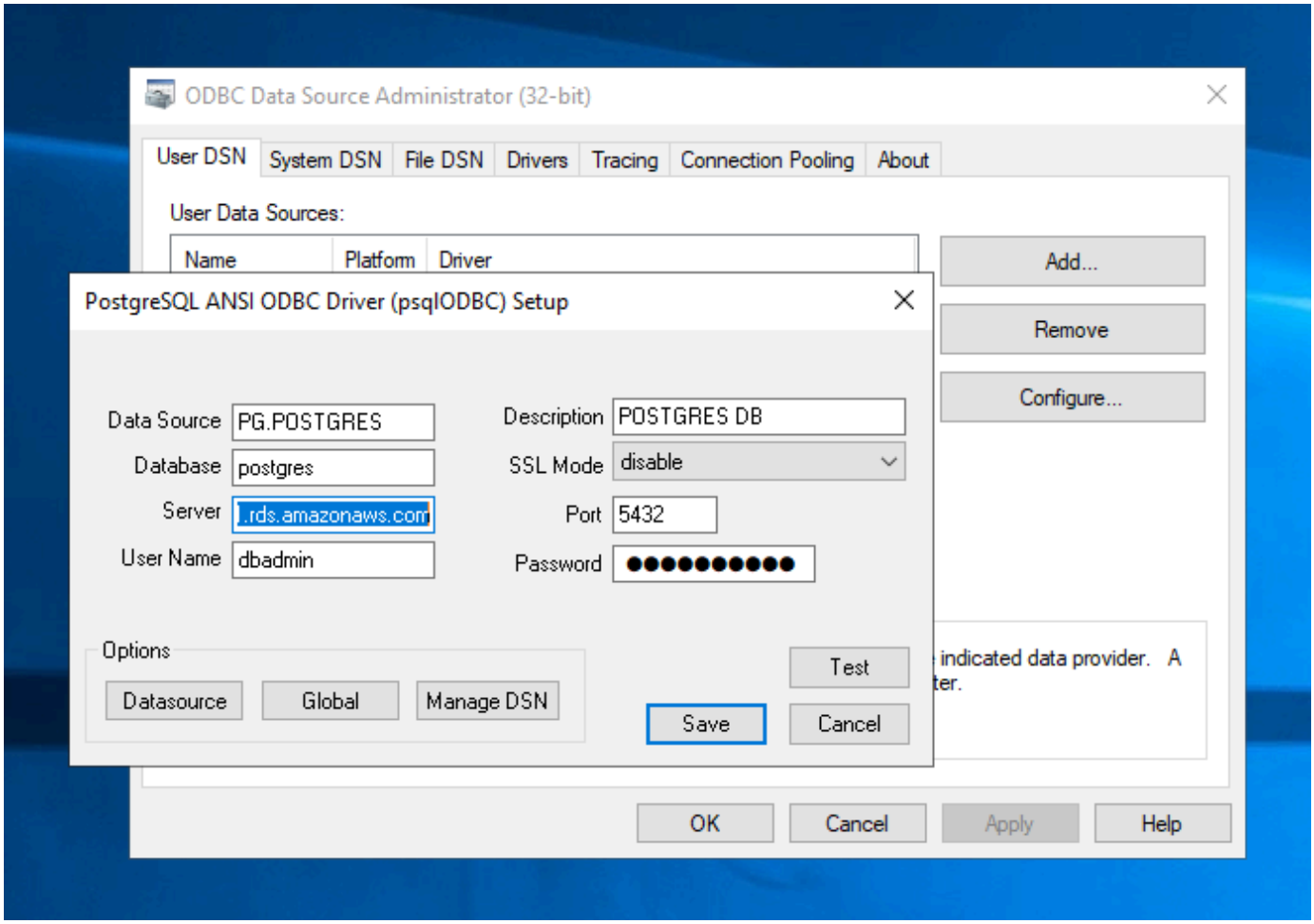
- 您将应用程序数据存储在 Aurora PostgreSQL 兼容版中。

步骤 1：设置与 Micro Focus 数据存储（Amazon RDS 数据库）的 ODBC 连接

在此步骤中，设置与数据库的 ODBC 连接，该数据库包含要以表和列形式查看的数据。您只需执行此步骤一次。

1. 使用 AppStream 2.0 流式处理 URL 登录 Micro Focus Enterprise Developer 桌面。
2. 打开 ODBC 数据来源管理器，选择用户 DSN，然后选择添加。
3. 在创建新数据来源中，选择 PostgreSQL ANSI，然后选择完成。
4. 通过提供必要的数据库信息为 PG.POSTGRES 创建数据来源，如下所示：

```
Data Source : PG.POSTGRES
Database    : postgres
Server      : rds_endpoint.rds.amazonaws.com
Port        : 5432
User Name   : user_name
Password    : user_password
```



5. 选择测试，以便确保连接正常。如果测试成功，您应该会看到消息 `Connection successful`。

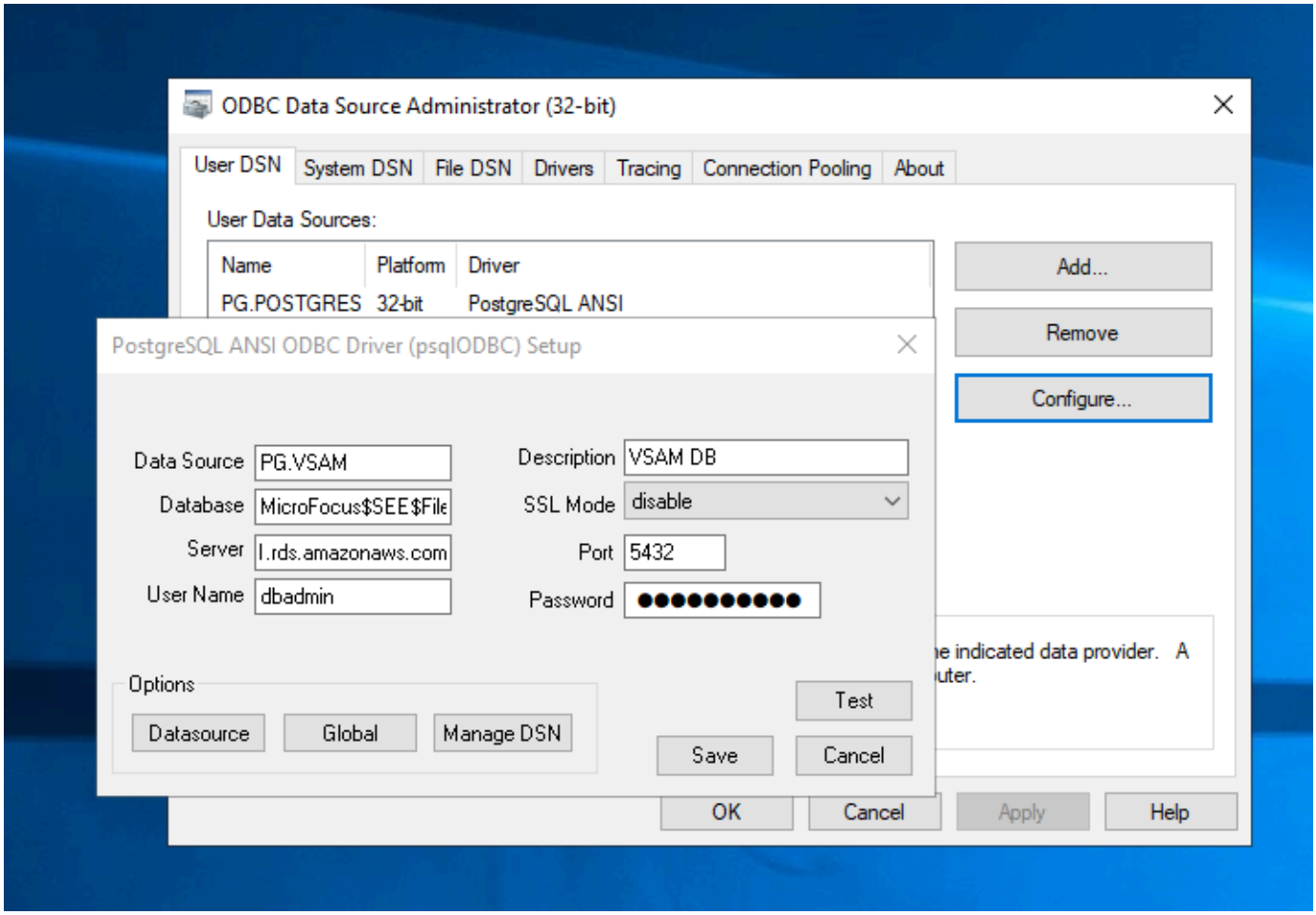
   如果测试不成功，请查看以下信息。

   - Amazon RDS 故障排除
   - 如何解决在连接我的 Amazon RDS 数据库实例时遇到的问题？

6. 保存数据来源。

7. 为 `PG.VSAM` 创建数据来源，测试连接，然后保存数据来源。提供以下数据库信息：

```
Data Source : PG.VSAM
Database    : MicroFocus$SEE$Files$VSAM
Server      : rds_endpoint.rds.amazonaws.com
Port        : 5432
User Name   : user_name
Password    : user_password
```



步骤 2：创建 MFDBFH.cfg 文件

在此步骤中，创建描述 Micro Focus 数据存储的配置文件。这是一个一次性配置步骤。

1. 在您的主文件夹（例如 `D:\PhotonUser\My Files\Home Folder\MFED\cfg\MFDBFH.cfg`）中，创建包含以下内容的 MFDBFH.cfg 文件。

```
<datastores>
       <server name="ESPACDatabase" type="postgresql" access="odbc">
        <dsn name="PG.POSTGRES" type="database" dbname="postgres"/>
        <dsn name="PG.VSAM" type="datastore" dsname="VSAM"/>
       </server>
      </datastores>
```

2. 通过运行以下命令查询 Micro Focus 数据存储来验证 MFDBFH 配置：

```
*###*
*## Test the connection by running the following commands*
*###*

set MFDBFH_CONFIG="D:\PhotonUser\My Files\Home Folder\MFED\cfg\MFDBFH.cfg"

dbfhdeploy list sql://ESPACDatabase/VSAM?folder=/DATA
```

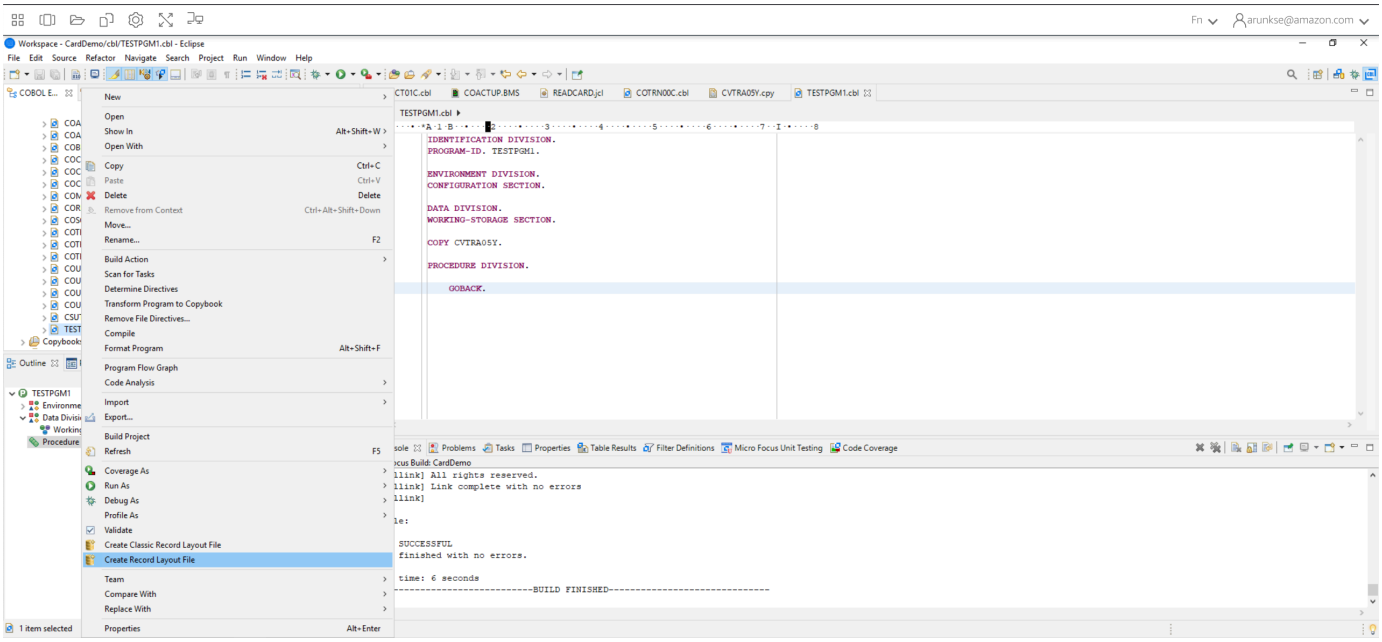
步骤 3：为 copybook 布局创建结构 (STR) 文件

在此步骤中，为 copybook 布局创建结构文件，以便以后使用该文件从数据集创建数据库视图。

1. 编译与您的 copybook 相关的程序。如果没有程序在使用 copybook，请创建并编译一个如下所示的简单程序，其中包含用于 copybook 的 COPY 语句。

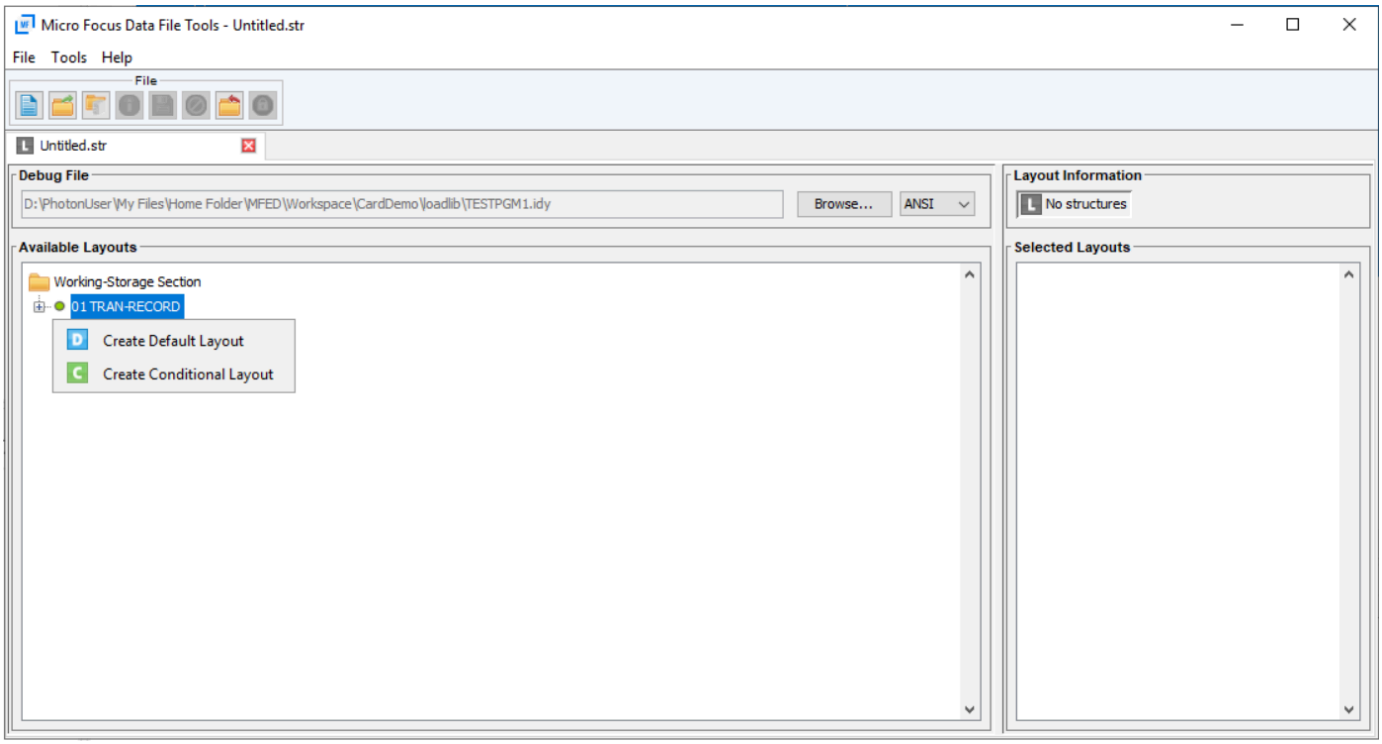
```
IDENTIFICATION DIVISION.
      PROGRAM-ID. TESTPGM1.

      ENVIRONMENT DIVISION.
      CONFIGURATION SECTION.

      DATA DIVISION.
      WORKING-STORAGE SECTION.

      COPY CVTRA05Y.

      PROCEDURE DIVISION.

      GOBACK.
```

2. 成功编译后，右键单击该程序并选择创建记录布局文件。这将使用编译期间生成的 .idy 文件打开 Micro Focus Data File Tools。

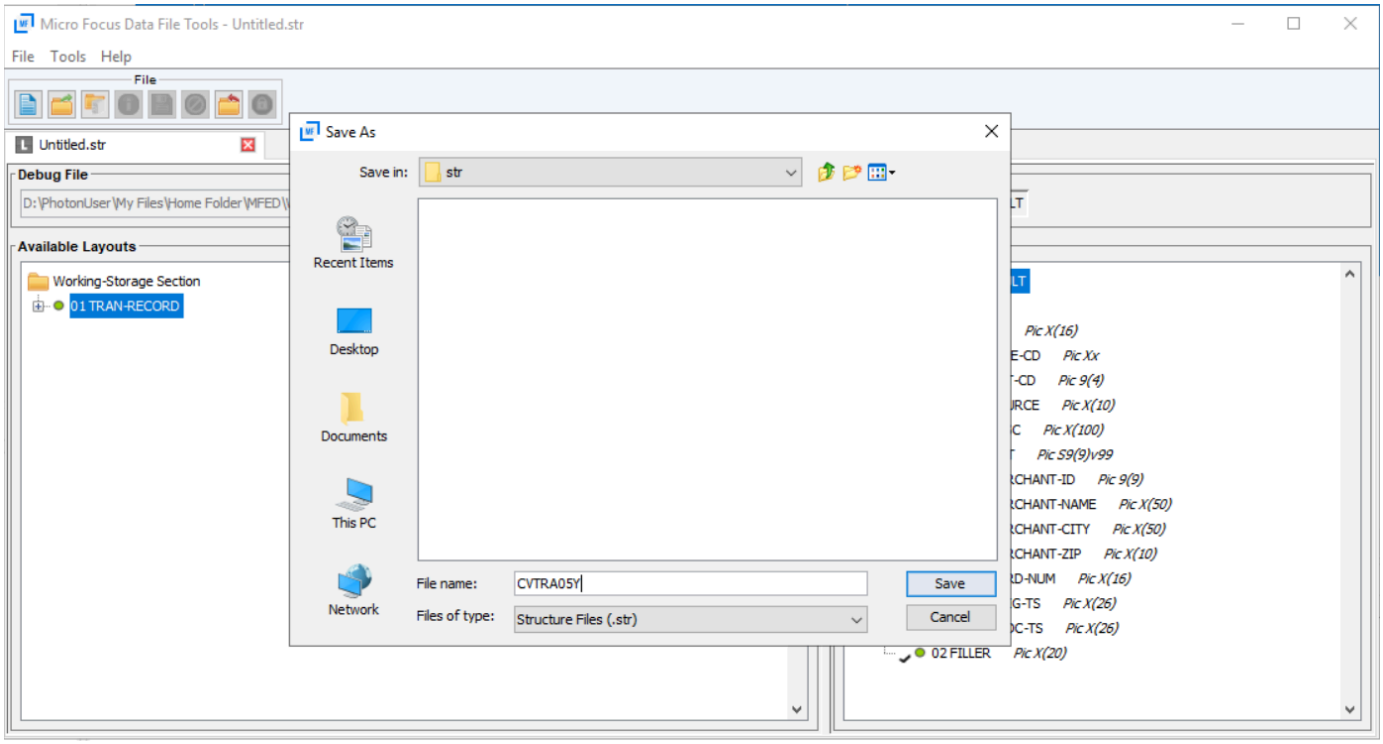


3. 右键单击记录结构，然后根据布局选择创建默认布局（单结构）或创建条件布局（多结构）。

   有关更多信息，请参阅 Micro Focus 文档中的创建结构文件和布局。



4. 创建布局后，从菜单中选择文件，然后选择另存为。浏览并将文件保存在主文件夹下，文件名与您的 copybook 相同。您可以选择创建名为 `str` 的文件夹，并将所有结构文件保存在此文件夹中。



步骤 4：使用结构 (STR) 文件创建数据库视图

在此步骤中，使用先前创建的结构文件为数据集创建数据库视图。

- 使用 dbfhview 命令为 Micro Focus 数据存储中已有的数据集创建数据库视图，如以下示例所示。

```
##
      ## The below command creates database view for VSAM file
 AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
      ## using the STR file CVTRA05Y.str
      ##

      dbfhview -create -struct:"D:\PhotonUser\My Files\Home Folder\MFED\str
\CVTRA05Y.str" -name:V_AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT -file:sql://
ESPACDatabase/VSAM/AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT?folder=/DATA

      ##
      ## Output:
      ##

      Micro Focus Database File Handler - View Generation Tool Version 8.0.00
```

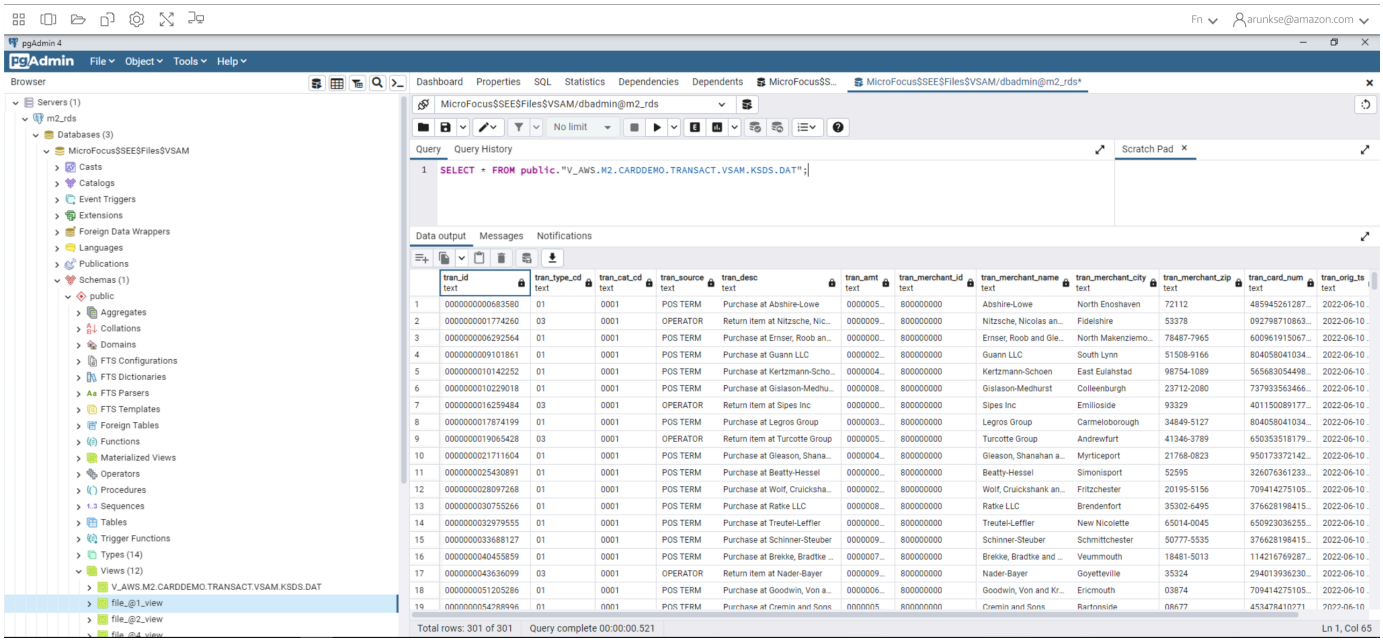
```
      Copyright (C) 1984-2022 Micro Focus. All rights reserved.

      VGN0017I Using structure definition 'TRAN-RECORD-DEFAULT'
      VGN0022I View 'V_AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT' installed in
datastore 'sql://espacdatabase/VSAM'
      VGN0002I The operation completed successfully
```

步骤 5：以表和列的形式查看 Micro Focus 数据集

在此步骤中，使用 pgAdmin 连接到数据库，以便您可以运行查询来以表和列的形式查看数据集。

- 使用 pgAdmin 连接到数据库 `MicroFocus$SEE$Files$VSAM` 并查询您在步骤 4 中创建的数据库视图。

```
SELECT * FROM public."V_AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT";
```



教程：在 Micro Focus Enterprise Developer 中使用模板

本教程介绍如何在 Micro Focus Enterprise Developer 中使用模板和预定义项目。它提供了三个用例。所有用例均使用 BankDemo 示例中提供的示例代码。要下载该示例，请选择 bankdemo.zip。

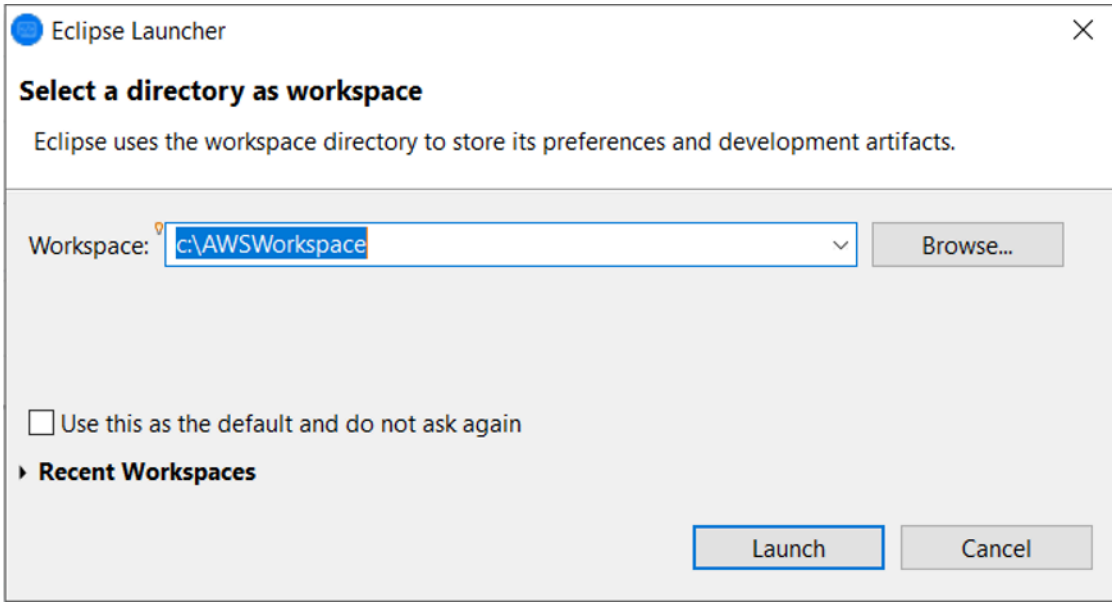
> **Important**
>
> 如果您使用适用于 Windows 的 Enterprise Developer 版本，则编译器生成的二进制文件只能在与 Enterprise Developer 一并提供的 Enterprise Server 上运行。您无法在基于 Linux 的 AWS Mainframe Modernization 运行时下运行这些文件。

主题

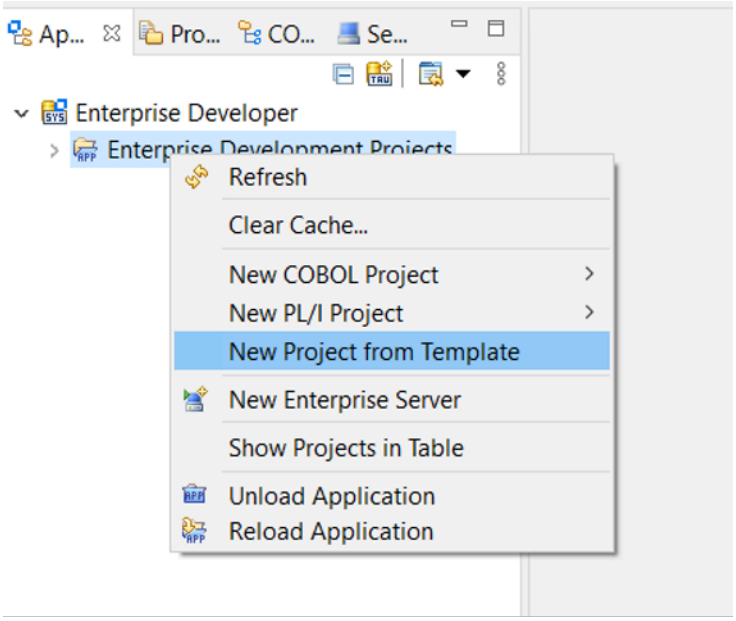
用例 1 – 使用包含源组件的 COBOL 项目模板

此用例要求您在演示预设置步骤中，将源组件复制到模板目录结构中。为了避免源有两个副本，在 `bankdemo.zip` 中，这点已从提供的原始 `AWSTemplates.zip` 中进行了更改。

1. 启动 Enterprise Developer 并指定所选工作区。

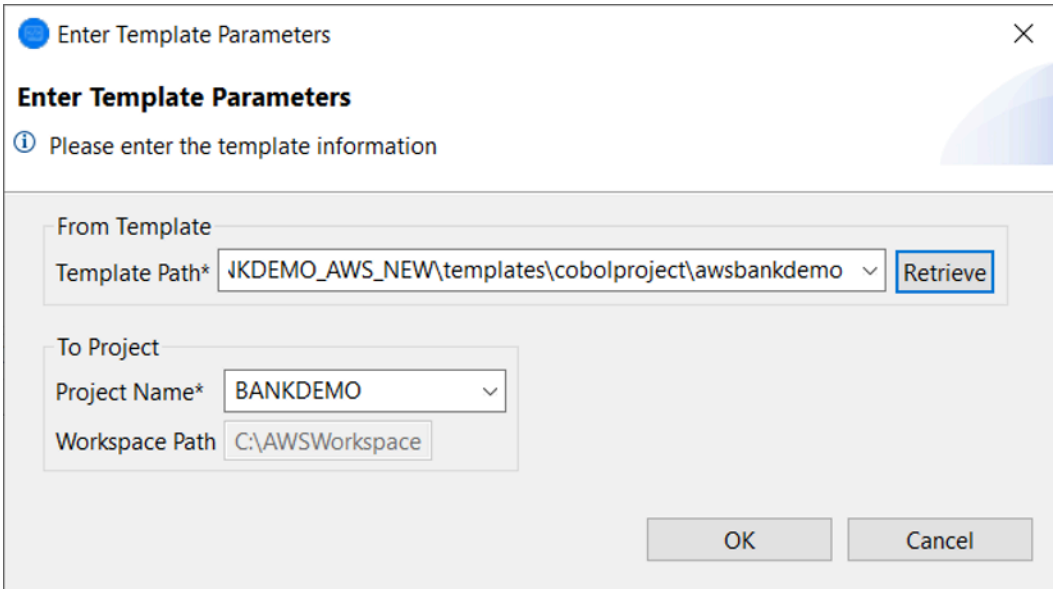


2. 在应用程序资源管理器视图中，从企业开发项目树视图项的上下文菜单中选择从模板新建项目。

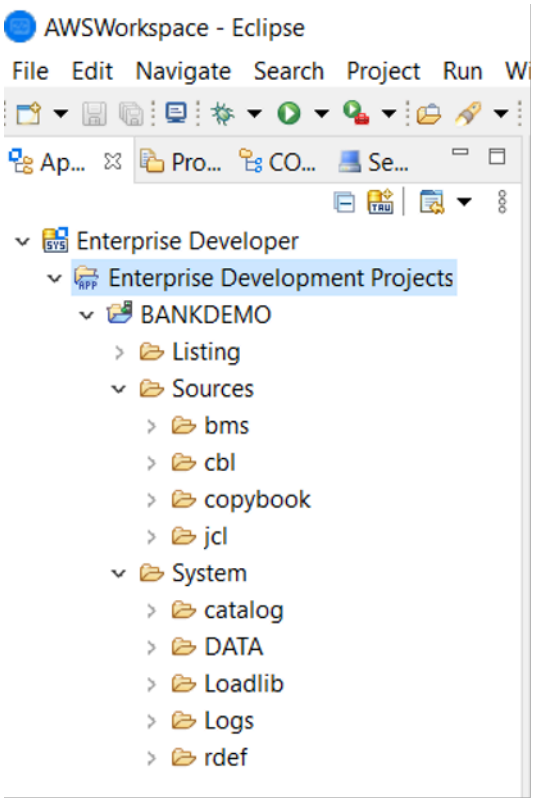


3. 输入模板参数，如图所示。

> **Note**
>
> “模板路径”是指 ZIP 的解压位置。



4. 选择“确定”将基于提供的模板创建一个本地开发 Eclipse 项目，该项目具有完整的源和执行环境结构。



System 结构包含一个完整的资源定义文件，其中包含 BANKDEMO 所需的条目、添加了条目的必需目录以及相应的 ASCII 数据文件。

由于源模板结构包含所有源项目，这些文件将复制到本地项目，因此会在 Enterprise Developer 中自动构建。

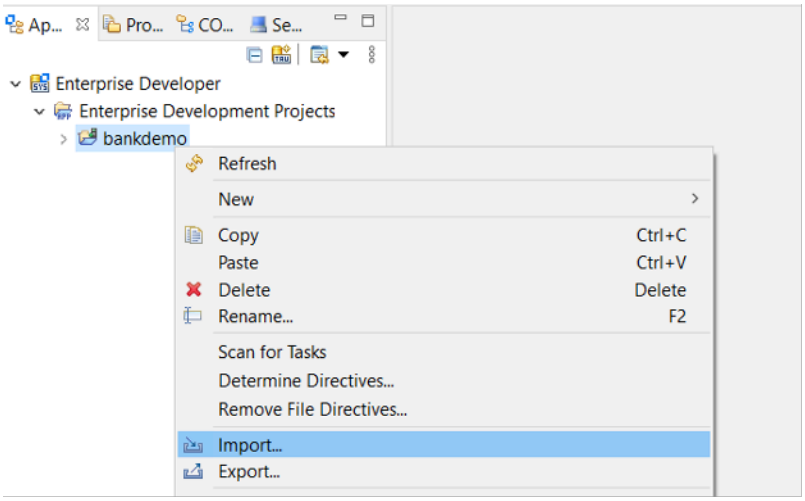
用例 2 – 使用不包含源组件的 COBOL 项目模板

步骤 1 到 3 与用例 1 – 使用包含源组件的 COBOL 项目模板相同。

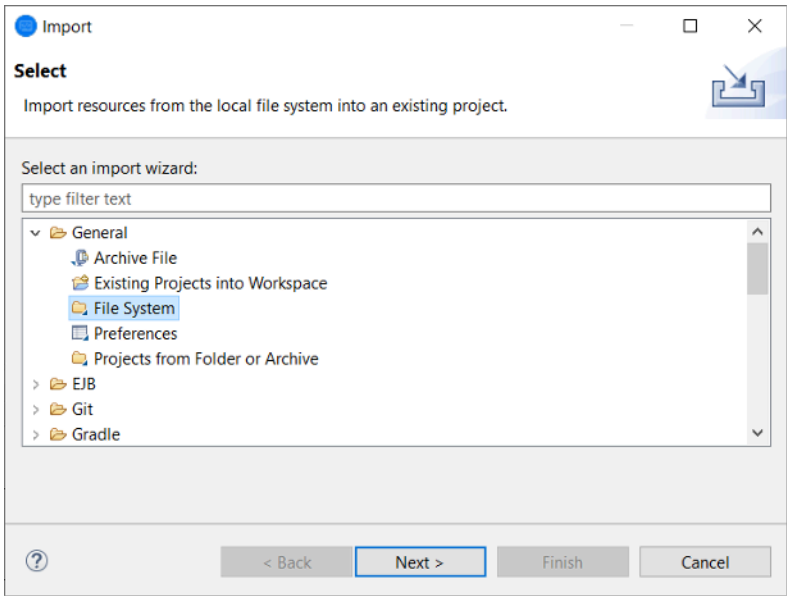
本用例中的 System 结构也包含一个完整的资源定义文件，其中包含 BankDemo 所需的条目、添加了条目的必需目录以及相应的 ASCII 数据文件。

不过，模板源结构不包含任何组件。您必须将组件从您正在使用的任何源存储库导入到项目中。

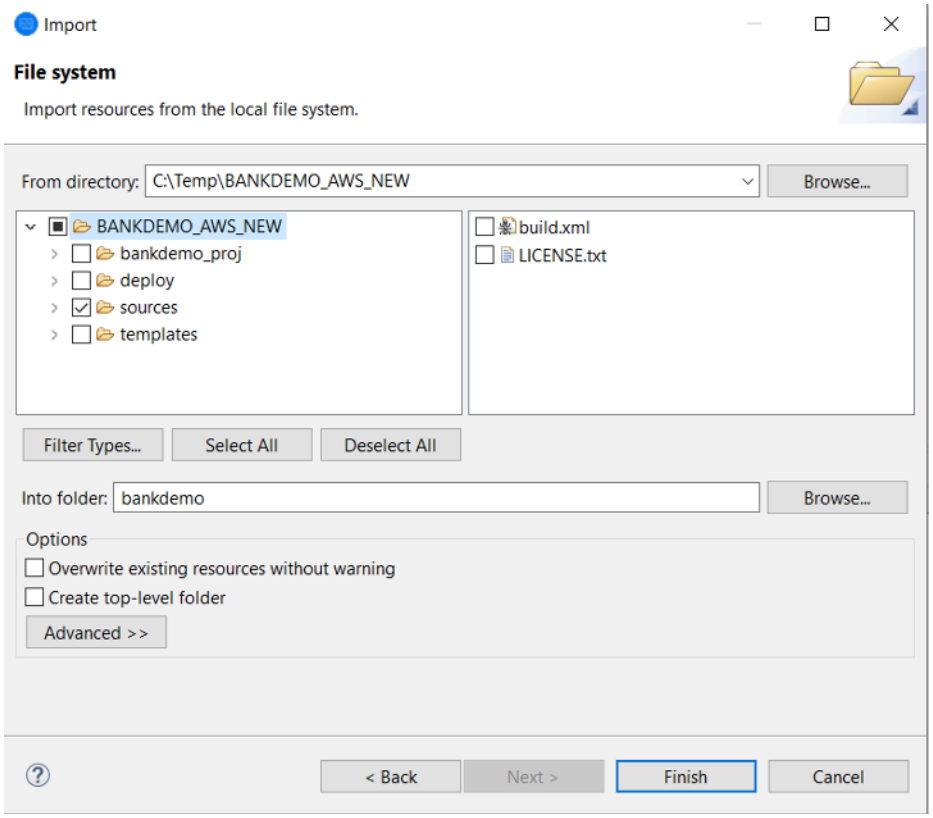
1. 选择项目名称。从相关的上下文菜单中，选择导入。



2. 在显示的对话框中，在 General 部分下，选择 File System，然后选择“下一步”。



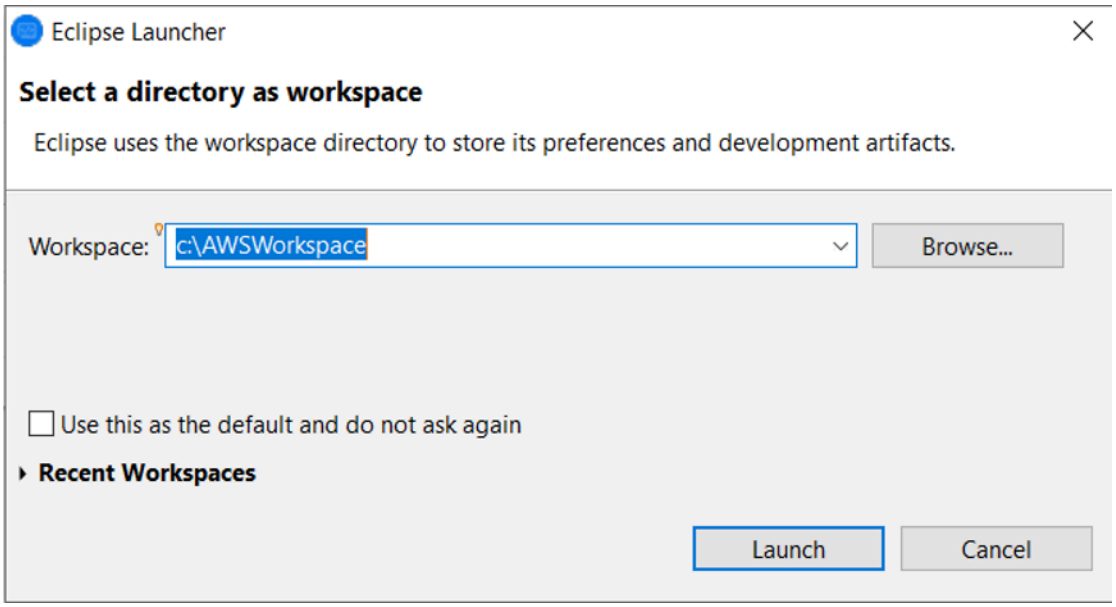
3. 通过浏览文件系统以指向存储库文件夹来填充从目录字段。选择要导入的所有文件夹，例如 `sources`。`Into folder` 字段将预先填充。选择完成。



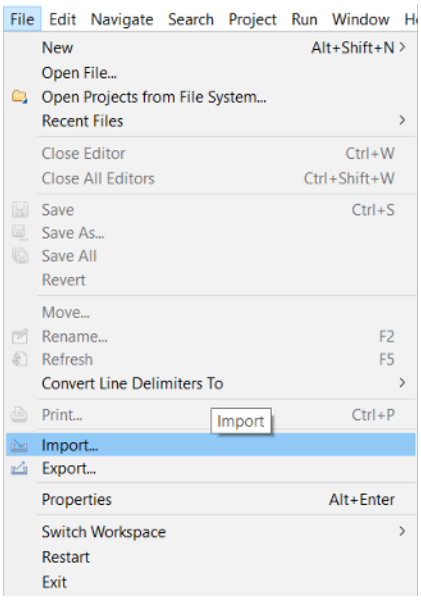
源模板结构包含所有源项目之后，即可在 Enterprise Developer 中自动构建。

用例 3 – 使用链接到源文件夹的预定义 COBOL 项目

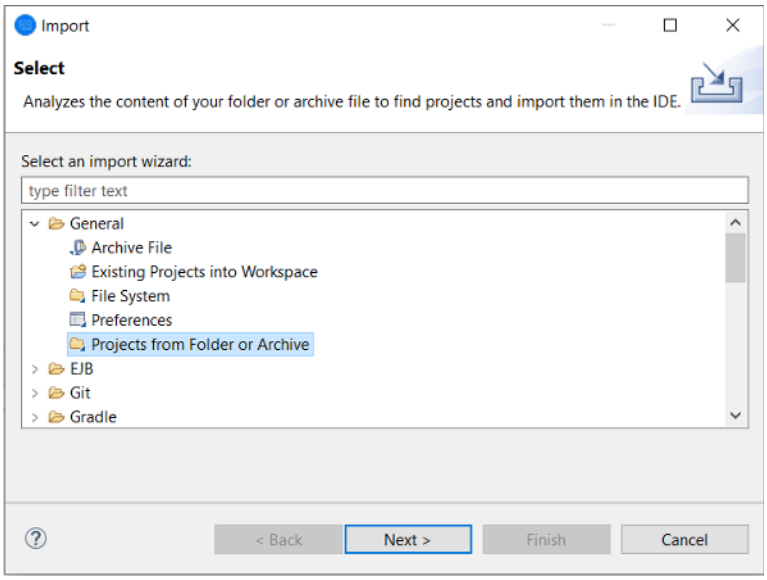
1. 启动 Enterprise Developer 并指定所选工作区。



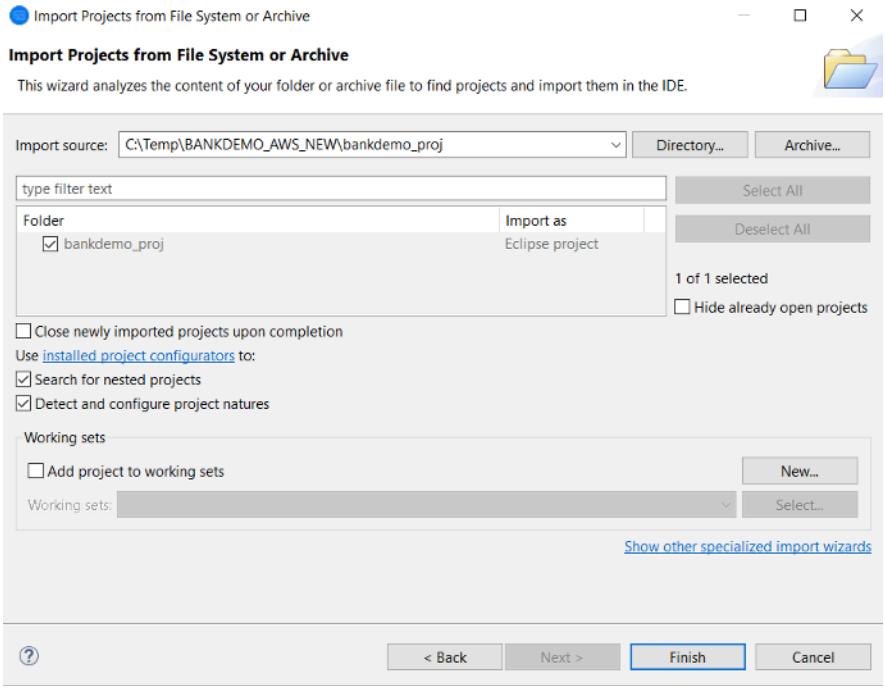
2. 从文件菜单中，选择导入。



3. 在显示的对话框中，在 General下，选择 Projects from Folder or Archive，然后选择下一步。



4. 填充导入源，选择目录，然后浏览文件系统以选择预定义的项目文件夹。其中包含的项目具有指向同一存储库中源文件夹的链接。

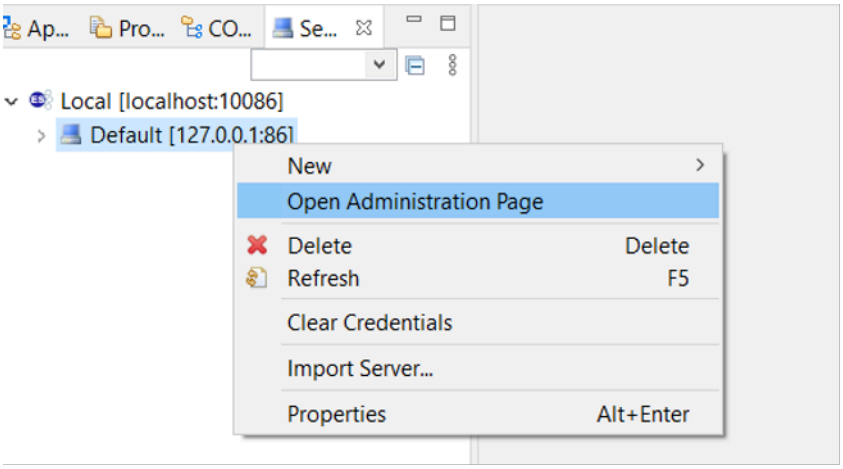


选择完成。

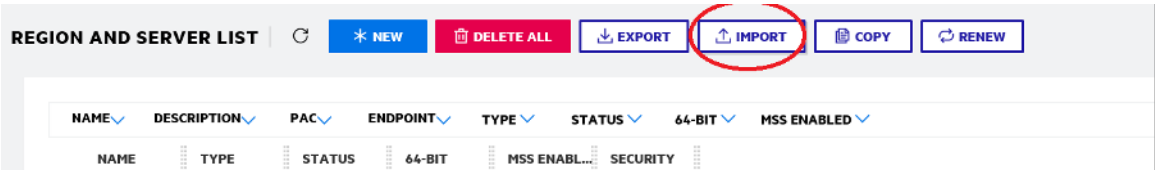
由于项目由源文件夹的链接填充，因此代码是自动生成的。

使用区域定义 JSON 模板

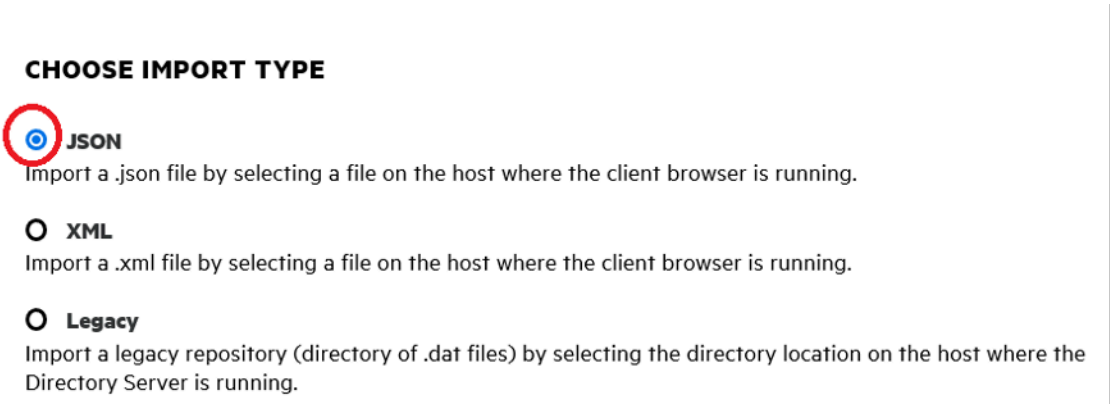
1. 切换到服务器资源管理器视图。从相关的上下文菜单中选择打开管理页面，这将启动默认浏览器。



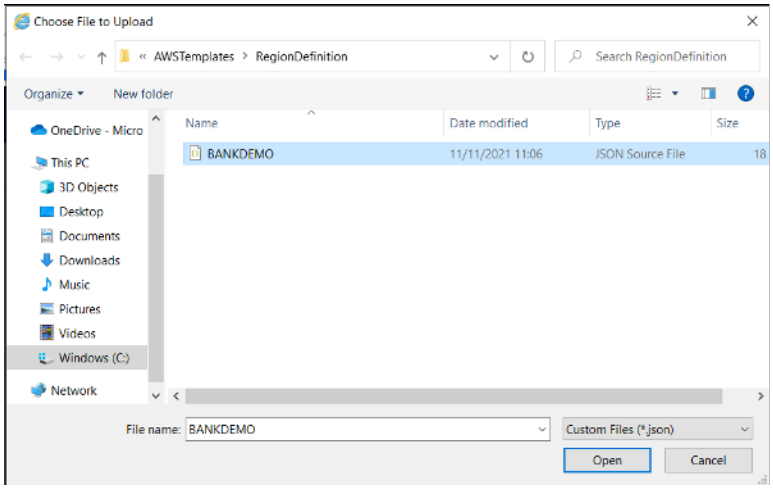
2. 从显示的 Enterprise Server 常用 Web 管理 (ESCWA) 屏幕中，选择导入。



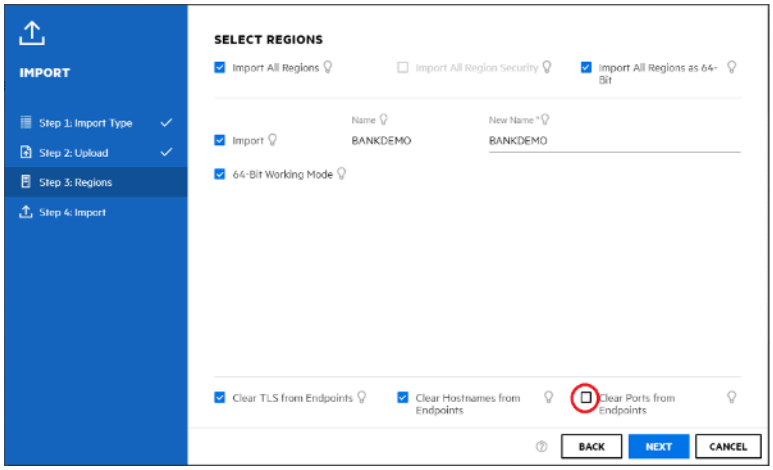
3. 选择 JSON 导入类型，然后选择下一步。



4. 上传提供的 BANKDEMO.JSON 文件。

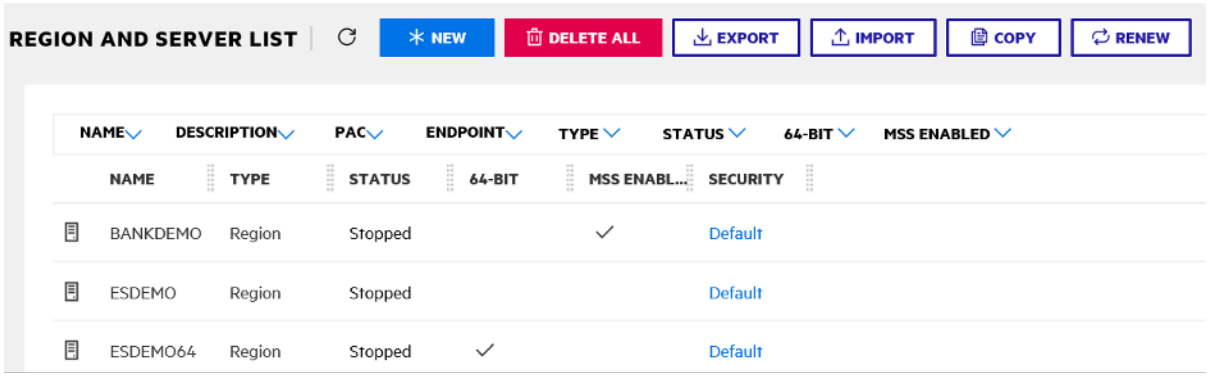


选择后，选择下一步。

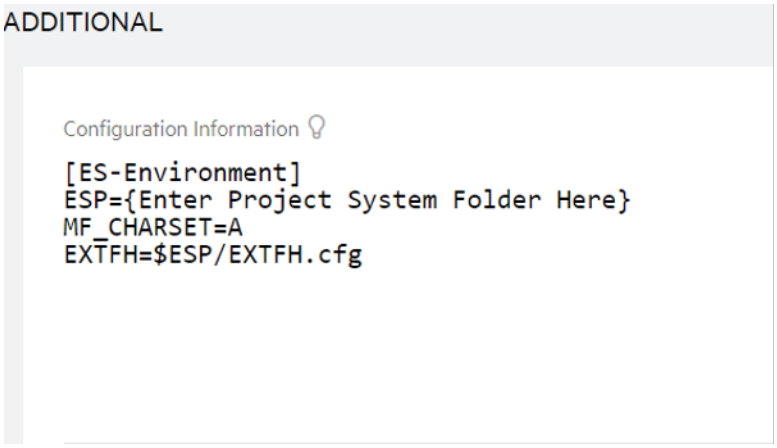


在选择区域面板上，确保未选中从端点清除端口选项，然后继续在面板中选择下一步，直到显示执行导入面板。然后选择导入。

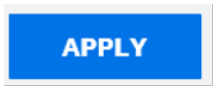
最后，单击完成。BANKDEMO 区域随后将被添加到服务器列表中。



5. 导航至 BANKDEMO 区域的常规属性。
6. 滚动到配置部分。
7. 需将 ESP 环境变量设置为与前面步骤中创建的 Eclipse 项目相关的 `System` 文件夹。应为 `workspacefolder/projectname/System`。

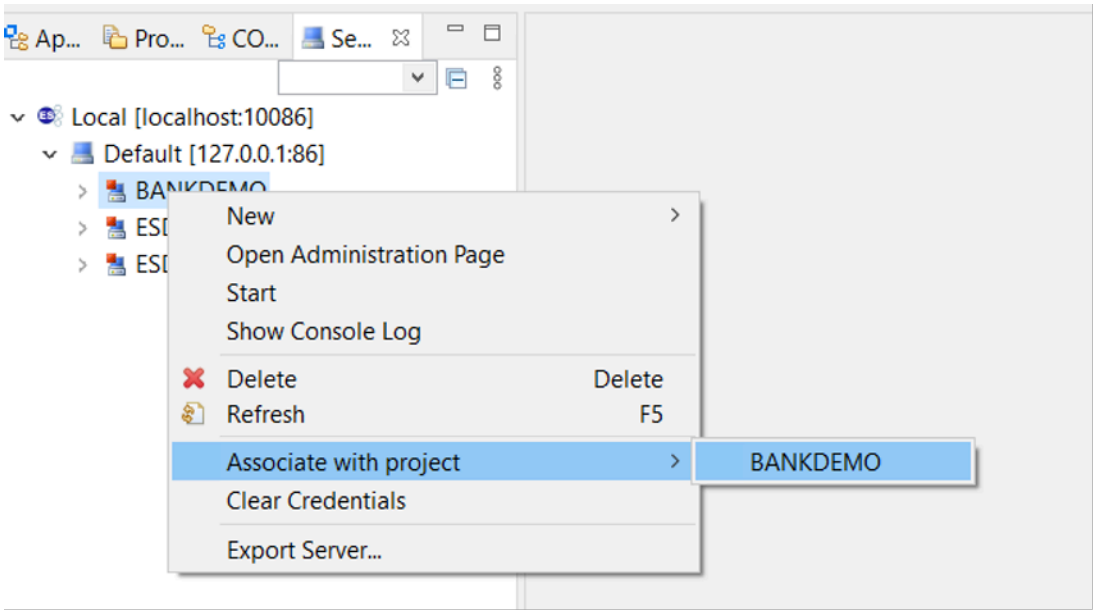


8. 单击应用。



该区域现已完全配置为与 Eclipse COBOL 项目配合运行。

9. 最后，回到 Enterprise Developer 中，将导入的区域与项目关联。



Enterprise Developer 环境现已准备就绪，具有完整的 BankDemo 工作版本。您可以针对该区域编辑、编译和调试代码。

> **Important**
>
> 如果您使用适用于 Windows 的 Enterprise Developer 版本，则编译器生成的二进制文件只能在与 Enterprise Developer 一并提供的 Enterprise Server 上运行。您无法在基于 Linux 的 AWS Mainframe Modernization 运行时下运行这些文件。

教程：为 BankDemo 示例应用程序设置 Micro Focus 构建

AWS Mainframe Modernization 使您能够为迁移的应用程序设置构建和持续集成/持续交付 (CI/CD) 管道。这些构建和管道使用 AWS CodeBuild、AWS CodeCommit 和 AWS CodePipeline 来提供这些功能。CodeBuild 是一项完全托管的构建服务，可编译源代码、运行单元测试以及生成可供部署的构件。CodeCommit 是一项版本控制服务，可用于在 AWS 云中私密地存储和管理 Git 存储库。CodePipeline 是一项持续交付服务，可用于建模、可视化和自动执行发布软件所需的步骤。

本教程演示如何使用 AWS CodeBuild 编译 Amazon S3 中的 BankDemo 示例应用程序源代码，然后将编译后的代码导回 Amazon S3。

AWS CodeBuild 是一项完全托管式连续集成服务，可编译源代码、运行测试以及生成可供部署的软件包。借助 CodeBuild，您可以使用预先打包的构建环境，也可以创建使用您自己的构建工具的自定义构建环境。此演示场景使用后者，包括一个使用预打包的 Docker 映像的 CodeBuild 构建环境。

> **Important**
>
> 在启动大型机现代化项目之前，我们建议您了解适用于大型机的 AWS 迁移加速计划 (MAP)，或者联系 AWS大型机专家，了解实现大型机应用程序现代化所需的步骤。

主题

先决条件

开始本教程之前，请满足以下先决条件：

- 下载 BankDemo 示例应用程序并将其解压缩到一个文件夹。源文件夹包含 COBOL 程序和 Copybook 以及 CICS BMS 定义。此外，它还包含一个供参考的 JCL 文件夹，尽管您不需要构建 JCL。该文件夹还包含构建所需的元文件。
- 在 AWS Mainframe Modernization 控制台中，选择工具。在分析、开发和构建资产中，选择使用我的 AWS 账户共享资产。

步骤 1：创建 Amazon S3 存储桶

在此步骤中，您将创建两个 Amazon S3 存储桶。一个是用于存放源代码的输入存储桶，另一个是用于存放构建输出的输出存储桶。有关更多信息，请参阅《Amazon S3 用户指南》中的创建、配置和使用 Amazon S3 存储桶。

1. 要创建输入存储桶，登录 Amazon S3 控制台并选择创建存储桶。
2. 在常规配置中，提供存储桶的名称，并指定要在其中创建存储桶的 AWS 区域。示例名称为 `codebuild-regionId-accountId-input-bucket`，其中 `regionId` 是存储桶的 AWS 区域，`accountId` 是您的 AWS 账户 ID。

   > **Note**
   >
   > 如果您要在非美国东部（弗吉尼亚州北部）的 AWS 区域创建存储桶，请指定 `LocationConstraint` 参数。有关更多信息，请参阅《Amazon Simple Storage Service API 参考》中的创建存储桶。

3. 保留所有其他设置并选择创建存储桶。
4. 重复步骤 1 – 3 来创建输出存储桶。示例名称为 `codebuild-regionId-accountId-output-bucket`，其中 `regionId` 是存储桶的 AWS 区域，`accountId` 是您的 AWS 账户 ID。

无论您为这些存储桶选择的名称如何，请务必在本教程中保持统一。

步骤 2：创建构建规范文件

在此步骤中，创建一个构建规范文件。此文件提供编译命令和相关设置并采用 YAML 格式，以便 CodeBuild 运行构建。有关更多信息，请参阅《AWS CodeBuild 用户指南》中的 CodeBuild 的构建规范参考。

1. 在您在先决条件中解压缩的目录中创建一个名为 `buildspec.yml` 的文件。
2. 将以下内容添加到该文件并保存。无需对该文件进行任何更改。

```
version: 0.2
env:
  exported-variables:
    - CODEBUILD_BUILD_ID
    - CODEBUILD_BUILD_ARN
phases:
  install:
    runtime-versions:
      python: 3.7
  pre_build:
    commands:
      - echo Installing source dependencies...
      - ls -lR $CODEBUILD_SRC_DIR/source
  build:
    commands:
      - echo Build started on `date`
      - /start-build.sh -Dbasedir=$CODEBUILD_SRC_DIR/source -Dloaddir=
$CODEBUILD_SRC_DIR/target
  post_build:
    commands:
      - ls -lR $CODEBUILD_SRC_DIR/target
      - echo Build completed on `date`
artifacts:
  files:
    - $CODEBUILD_SRC_DIR/target/**
```

此处的 `CODEBUILD_BUILD_ID`、`CODEBUILD_BUILD_ARN`、`$CODEBUILD_SRC_DIR/source` 和 `$CODEBUILD_SRC_DIR/target` 是 CodeBuild 中可用的环境变量。有关更多信息，请参阅构建环境中的环境变量。

此时，您的目录结构应如下所示。

```
(root directory name)
    |-- build.xml
    |-- buildspec.yml
    |-- LICENSE.txt
    |-- source
         |... etc.
```

3. 将文件夹的内容压缩到名为 BankDemo.zip 的文件中。在本教程中，您无法压缩文件夹。因此，将文件夹的内容压缩到文件 BankDemo.zip 中。

步骤 3：上传源文件

在此步骤中，将 BankDemo 示例应用程序的源代码上传到您的 Amazon S3 输入存储桶。

1. 登录到 Amazon S3 控制台，并在左侧导航窗格中，选择存储桶。然后选择您之前创建的输入存储桶。
2. 在对象下，选择上传。
3. 在文件和文件夹部分，选择添加文件。
4. 导航到并选择 BankDemo.zip 文件。
5. 选择上传。

步骤 4：创建 IAM 策略

在此步骤中，创建两个 IAM 策略。一个策略授予 AWS Mainframe Modernization 访问和使用 Docker 映像（包含 Micro Focus 构建工具）的权限。此策略不是为客户自定义的。另一个策略授予 AWS Mainframe Modernization 与输入和输出存储桶以及与 CodeBuild 生成的 Amazon CloudWatch 日志进行交互的权限。

要了解有关创建 IAM 策略的信息，请参阅《IAM 用户指南》中的编辑 IAM 策略。

创建允许访问 Docker 映像的策略

1. 在 IAM 控制台中，将以下策略文档复制并粘贴到策略编辑器中。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:*:673918848628:repository/m2-enterprise-build-
tools"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::aws-m2-repo-*/*"
        }
    ]
}
```

2. 提供策略名称，例如 `m2CodeBuildPolicy`。

创建允许 AWS Mainframe Modernization 与存储桶和日志交互的策略

1. 在 IAM 控制台中，将以下策略文档复制并粘贴到策略编辑器中。请务必将 `regionId` 更新为 AWS 区域，并将 `accountId` 更新为您的 AWS 账户。

```
{
      "Version": "2012-10-17",
      "Statement": [
          {
              "Action": [
                  "logs:CreateLogGroup",
                  "logs:CreateLogStream",
```

```
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:regionId:accountId:log-group:/aws/codebuild/
codebuild-bankdemo-project",
                "arn:aws:logs:regionId:accountId:log-group:/aws/codebuild/
codebuild-bankdemo-project:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::codebuild-regionId-accountId-input-bucket",
                "arn:aws:s3:::codebuild-regionId-accountId-input-bucket/*",
                "arn:aws:s3:::codebuild-regionId-accountId-output-bucket",
                "arn:aws:s3:::codebuild-regionId-accountId-output-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

2. 提供策略名称，例如 BankdemoCodeBuildRolePolicy。

步骤 5：创建 IAM 角色

在此步骤中，创建一个新的 IAM 角色。在您将之前创建的 IAM 策略与该新 IAM 角色关联后，该角色允许 CodeBuild 与您的 AWS 资源交互。

有关创建服务角色的信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

1. 登录到 IAM 控制台，然后在左侧导航窗格中，选择角色。
2. 选择创建角色。

3. 在可信实体类型下，选择 AWS 服务。
4. 在其他 AWS 服务的用例下，选择 CodeBuild，然后再次选择 CodeBuild。
5. 选择下一步。
6. 在添加权限页面上，选择下一步。您稍后给该角色分配策略。
7. 在角色详细信息下，提供角色的名称，例如 BankdemoCodeBuildServiceRole。
8. 在选择可信实体下，验证策略文档是否如下所示：

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

9. 选择 Create role（创建角色）。

步骤 6：将 IAM 策略附加到 IAM 角色

在此步骤中，将您之前创建的两个 IAM 策略附加到 BankdemoCodeBuildServiceRole IAM 角色。

1. 登录到 IAM 控制台，然后在左侧导航窗格中，选择角色。
2. 在角色中，选择您之前创建的角色，例如 BankdemoCodeBuildServiceRole。
3. 在权限策略中，依次选择添加权限和附加策略。
4. 在其他权限策略中，选择您之前创建的策略，例如 m2CodeBuildPolicy 和 BankdemoCodeBuildRolePolicy。
5. 选择附加策略。

步骤 7：创建 CodeBuild 项目

在此步骤中，创建 CodeBuild 项目。

1. 登录到 CodeBuild 控制台并选择创建构建项目。
2. 在项目配置部分中，提供项目的名称，例如 codebuild-bankdemo-project。
3. 在源部分中，对于源提供商，选择 Amazon S3，然后选择您之前创建的输入存储桶，例如codebuild-regionId-accountId-input-bucket。
4. 在 S3 对象密钥或 S3 文件夹字段中，输入您上传到 S3 存储桶的 zip 文件的名称。在本例中，此文件名为 bankdemo.zip。
5. 在环境部分中，选择自定义映像。
6. 在环境类型字段中，选择 Linux。
7. 在映像注册表中，选择其他注册表。
8. 在外部注册表 URL 字段中，输入 673918848628.dkr.ecr.us-west-2.amazonaws.com/m2-enterprise-build-tools:latest
9. 在服务角色下，选择现有服务角色，然后在角色 ARN 字段中，选择您之前创建的服务角色，例如 BankdemoCodeBuildServiceRole。
10. 在构建规范部分中，选择使用构建规范文件。
11. 在构件部分的类型下，选择 Amazon S3，然后选择您的输出存储桶，例如 codebuild-regionId-accountId-output-bucket。
12. 在名称字段中，输入存储桶中要包含构建输出构件的文件夹的名称，例如 bankdemo-output.zip。
13. 在构件打包下，选择 Zip。
14. 选择创建构建项目。

步骤 8：启动构建

在此步骤中，启动构建。

1. 登录到 CodeBuild 控制台。
2. 在左侧导航窗格中，选择构建项目。
3. 选择您之前创建的构建项目，例如 codebuild-bankdemo-project。
4. 选择启动构建。

此命令开始构建。构建以异步方式运行。此命令的输出是一个包含属性 ID 的 JSON。此属性 ID 是对刚刚启动的构建的 CodeBuild 构建 ID 的引用。您可以在 CodeBuild 控制台中查看构建的状态。您还可

以在控制台中查看有关构建执行的详细日志。有关更多信息，请参阅《AWS CodeBuild 用户指南》中的查看详细构建信息。

当前阶段完成后，即表示您的构建已成功完成，并且编译的构件已在 Amazon S3 上准备就绪。

步骤 9：下载输出构件

在此步骤中，从 Amazon S3 下载输出构件。Micro Focus 构建工具可以创建几种不同的可执行文件类型。在本教程中，它将生成共享对象。

1. 登录 Amazon S3 控制台。
2. 在存储桶部分中，选择您的输出桶的名称，例如 `codebuild-regionId-accountId-output-bucket`。
3. 选择下载。
4. 解压下载的文件。导航到目标文件夹以查看构建构件。其中包括 `.so` Linux 共享对象。

清理资源

如果您不再需要为本教程创建的资源，请将其删除，以免产生额外费用。为此，请完成以下步骤：

- 删除您为本教程创建的 S3 存储桶。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的删除存储桶。
- 删除您为本教程创建的 IAM 策略。有关更多信息，请参阅《IAM 用户指南》 中的删除 IAM 策略。
- 删除您为本教程创建的 IAM 角色。有关更多信息，请参阅《IAM 用户指南》中的删除角色或实例配置文件。
- 删除您为本教程创建的 CodeBuild 项目。有关详细信息，请参阅《AWS CodeBuild 用户指南》中的在 CodeBuild 中删除构建项目。

教程：设置 CI/CD 管道以与 Micro Focus Enterprise Developer 搭配使用

本教程展示如何在 Micro Focus Enterprise Developer 中导入、编辑、编译和运行 BankDemo 示例应用程序，然后提交更改以触发 CI/CD 管道。

目录

先决条件

下载以下文件。

- `basic-infra.yaml`
  - 从欧洲地区（法兰克福）区域下载。
  - 从美国东部（弗吉尼亚州北部）区域下载。
- `pipeline.yaml`

  - 从欧洲地区（法兰克福）区域下载。
  - 从美国东部（弗吉尼亚州北部）区域下载。
- m2-code-sync-function.zip
  - 从欧洲地区（法兰克福）区域下载。
  - 从美国东部（弗吉尼亚州北部）区域下载。
- config_git.yml
  - 从欧洲地区（法兰克福）区域下载。
  - 从美国东部（弗吉尼亚州北部）区域下载。
- BANKDEMO-source.zip
  - 从欧洲地区（法兰克福）区域下载。
  - 从美国东部（弗吉尼亚州北部）区域下载。
- BANKDEMO-exercise.zip
  - 从欧洲地区（法兰克福）区域下载。
  - 从美国东部（弗吉尼亚州北部）区域下载。

每个文件的用途如下：

basic-infra.yaml

该 AWS CloudFormation 模板创建 CI/CD 管道所需的基本基础设施：VPC、Amazon S3 存储桶等。

pipeline.yaml

Lambda 函数使用此 AWS CloudFormation 模板来启动管道堆栈。请确保此模板位于可公开访问的 Amazon S3 存储桶中。将指向此存储桶的链接添加为 basic-infra.yaml 模板中 PipelineTemplateURL 参数的默认值。

m2-code-sync-function.zip

此 Lambda 函数创建 CodeCommit 存储库和基于 config_git.yaml 的目录结构，并使用 pipeline.yaml 启动管道堆栈。确保此 zip 文件位于所有支持 AWS Mainframe Modernization 的 AWS 区域中可公开访问的 Amazon S3 存储桶中。我们建议您将文件存储在一个 AWS 区域的存储桶中，然后将其复制到所有 AWS 区域的存储桶中。使用以下存储桶命名约定：带有标识特定 AWS 区域的后缀（例如 m2-cicd-deployment-source-eu-west-1），添加前缀 m2-cicd-deployment-source 作为参数 DeploymentSourceBucket 的默认值，并使用 AWS

CloudFormation 替换函数 `!Sub {DeploymentSourceBucket}-${AWS::Region}` 形成完成的存储桶名称，同时针对资源 `SourceSyncLambdaFunction` 引用 `basic-infra.yaml` 模板中的存储桶。

`config_git.yml`

CodeCommit 目录结构定义。有关更多信息，请参阅 示例 YAML 触发器文件 config_git.yml。

`BANKDEMO-source.zip`.

从 CodeCommit 存储库中创建的 BankDemo 源代码和配置文件。

`BANKDEMO-exercise.zip`.

从 CodeCommit 存储库中创建的用于教程练习的 BankDemo 源。

创建 CI/CD 管道基本基础设施

通过 AWS CloudFormation 控制台，使用 AWS CloudFormation 模板 `basic-infra.yaml` 创建 CI/CD 管道基本基础设施堆栈。此堆栈创建可用于上传应用程序代码和数据的 Amazon S3 存储桶，以及一个用于创建其他必要资源（例如 AWS CodeCommit 存储库和 AWS CodePipeline 管道）的支持 AWS Lambda 函数。

> **Note**
>
> 要启动此堆栈，您需要管理 IAM、Amazon S3、Lambda 和 AWS CloudFormation 的权限以及使用 AWS KMS 的权限。

1. 登录到 AWS Management Console 并打开 AWS CloudFormation 控制台 https://console.aws.amazon.com/cloudformation。
2. 使用以下其中一个选项创建堆栈：
   - 选择Create Stack（创建堆栈）。如果您当前有正在运行的堆栈，这将是唯一选项。
   - 在堆栈页面上，选择 创建堆栈。仅当您没有运行堆栈时，才可以看到此选项。
3. 在指定模板页面上：
   - 在准备模板中，选择模板准备就绪。
   - 在指定模板中，选择 Amazon S3 URL 作为模板来源，并根据您的 AWS 区域，输入以下 URL 之一。

  - https://m2-us-east-1.s3.us-east-1.amazonaws.com/cicd/mf/basic-infra.yaml
  - https://m2-eu-central-1.s3.eu-central-1.amazonaws.com/cicd/mf/basic-infra.yaml
- 接受您的设置，选择下一步。

随后会打开创建堆栈页面。

Specify stack details

Stack name

Stack name

m2-cicd-pipeline-foundation

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

Networking Configuration

Do you want to use an existing VPC in your account?

If you select 'Yes', then you must provide the VPC ID and the Subnet IDs.

No (Create one)

Which VPC ID should be used?

If you selected 'Yes' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

Which private subnet ID should be used?

If you selected 'Yes' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

Which private subnet ID in a different AZ should be used for HA?

If you selected 'Yes' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

Enter the CIDR block that should be used for the new VPC

If you selected 'No (Create one)' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

11.11.0.0/16

CIDR bits for creating subnets. Choose 5 for /27, 6 for /26, 7 for /25, 8 for /24 range

If you selected 'No (Create one)' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

5

Deployment Configuration

Name of the S3 bucket which contains the source files for this stack deployment

Don't change unless you know what you are doing.

m2

Name of the source package file for the infrastructure Lambda function

Don't change unless you know what you are doing.

cicd/mf/m2-code-sync-function.zip

Full URL of the pipeline CloudFormation template file

Don't change unless you know what you are doing.

https://m2-us-east-1.s3.us-east-1.amazonaws.com/cicd/mf/pipeline.yaml

What name prefix to use for the new S3 buckets?

A name prefix for the S3 buckets that will be created by this stack.

mf

Cancel Previous Next

进行以下更改：

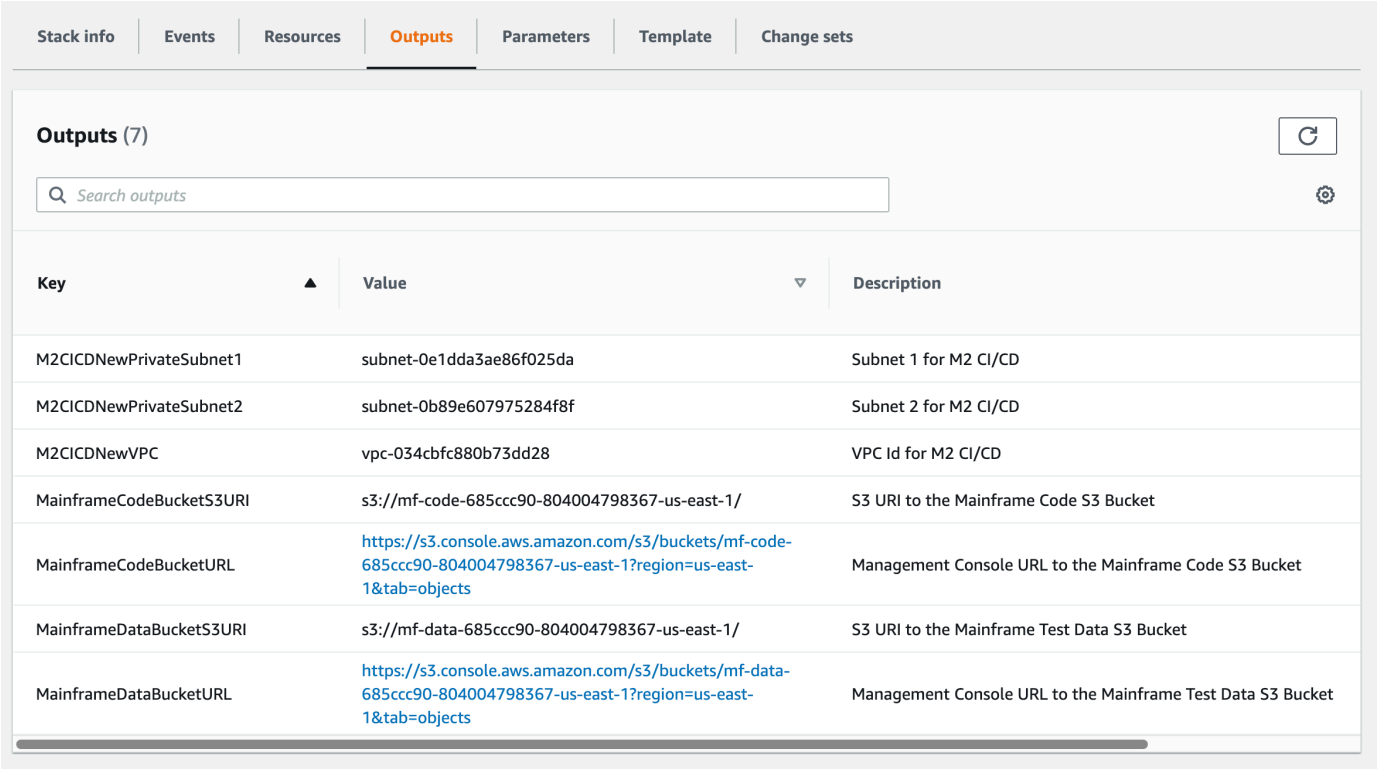
- 为堆栈名称和网络配置参数提供适当的值。
- 部署配置中的大多数参数都已适当预先填充，因此您无需对其进行修改。根据您的 AWS 区域，将管道 AWS CloudFormation 模板更改为以下 Amazon S3 URL 之一。
  - `https://m2-us-east-1.s3.amazonaws.com/cicd/mf/pipeline.yaml`
  - `https://m2-eu-central-1.s3.eu-central-1.amazonaws.com/cicd/mf/pipeline.yaml`
- 选择下一步。

> **Note**
>
> 除非您自己修改了 AWS CloudFormation 模板，否则，请勿更改默认参数值。

4. 在配置堆栈选项中，选择下一步。
5. 在功能中，选择我确认 AWS CloudFormation 可能会创建 IAM 资源，以便允许 AWS CloudFormation 代表您创建 IAM 角色。选择创建堆栈。

> **Note**
>
> 该堆栈配置需要花费 3 到 5 分钟。

6. 成功创建堆栈后，导航至新配置堆栈的输出部分。在此，您可以找到需要上传大型机代码和相关文件的 Amazon S3 存储桶。



创建 AWS CodeCommit 存储库和 CI/CD 管道

在此步骤中，创建一个 CodeCommit 存储库，并通过调用 Lambda 函数（调用 AWS CloudFormation 来创建管道堆栈）来配置 CI/CD 管道堆栈。

1. 将 BankDemo 示例应用程序下载到本地机器中。
2. 将 `bankdemo.zip` 从本地机器上传到创建 CI/CD 管道基本基础设施中创建的 Amazon S3 桶。
3. 下载 `config_git.yml`。
4. 按如下方式修改 `config_git.yml`（如有必要）：

   - 添加您自己的目标存储库名称、目标分支和提交消息。

   ```
   repository-config:
     target-repository: bankdemo-repo
     target-branch: main
     commit-message: Initial commit for bankdemo-repo main branch
   ```

   - 输入您用于接收通知的电子邮件地址。

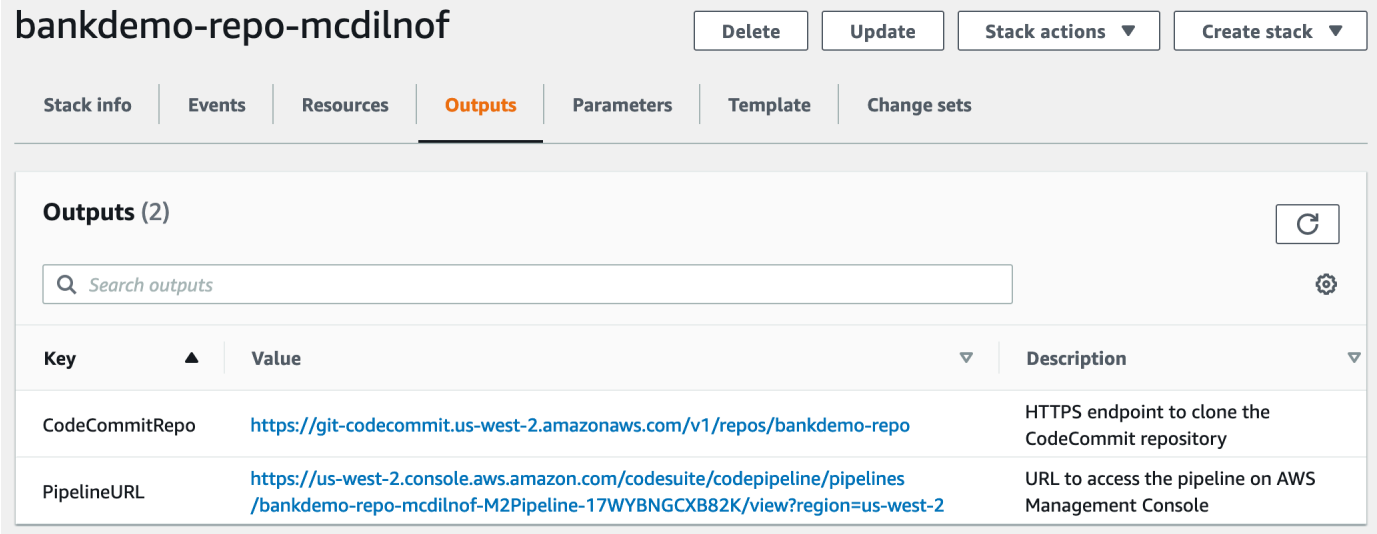
   ```
   pipeline-config:
   ```

```
  # Send pipeline failure notifications to these email addresses
  alert-notifications:
    - myname@mycompany.com
  # Send notifications for manual approval before production deployment to these
email addresses
  approval-notifications:
    - myname@mycompany.com
```

5. 将包含 CodeCommit 存储库文件夹结构定义的 `config_git.yml` 文件上传到创建 CI/CD 管道基本基础设施中创建的 Amazon S3 存储桶。这将调用 Lambda 函数，该函数将自动配置存储库和管道。

   这将创建一个 CodeCommit 存储库，该存储库使用 `config_git.yml` 文件中定义的 `target-repository` 中提供的名称，例如 `bankdemo-repo`。

   Lambda 函数还将通过 AWS CloudFormation 创建 CI/CD 管道堆栈。AWS CloudFormation 堆栈使用提供的 `target-repository` 名称作为前缀，后跟一个随机字符串（例如 `bankdemo-repo-`*`01234567`*）。您可以在 AWS Management Console 中找到 CodeCommit 存储库 URL 和用于访问已创建管道的 URL。



6. CodeCommit 存储库创建完成时，将立即触发 CI/CD 管道来执行完整的 CI/CD。
7. 推送文件后，将自动触发管道，该管道将构建、分阶段部署、运行一些测试并等待手动批准，然后再将其部署到生产环境中。

示例 YAML 触发器文件 config_git.yml

```
repository-config:
```

```
target-repository: bankdemo-repo
target-branch: main
commit-message: Initial commit for bankdemo-repo main branch
directory-structure:
  - '/':
      files:
      - build.xml
      - '*.yaml'
      - '*.yml'
      - '*.xml'
      - 'LICENSE.txt'
      readme: |
        # Root Folder
        - 'build.xml' : Build configuration for the application
  - tests:
      files:
      - '*.py'
      readme: |
        # Test Folder
        - '*.py' : Test scripts
  - config:
      files:
      - 'BANKDEMO.csd'
      - 'BANKDEMO.json'
      - 'BANKDEMO_ED.json'
      - 'dfhdrdat'
      - 'ESPGSQLXA.dll'
      - 'ESPGSQLXA64.so'
      - 'ESPGSQLXA64_S.so'
      - 'EXTFH.cfg'
      - 'm2-2021-04-28.normal.json'
      - 'MFDBFH.cfg'
      - 'application-definition-template-config.json'
      readme: |
        # Config Folder
        This folder contains the application configuration files.
        - 'BANKDEMO.csd'     : CICS Resource definitions export file
        - 'BANKDEMO.json'    : Enterprise Server configuration
        - 'BANKDEMO_ED.json' : Enterprise Server configuration for ED
        - 'dfhdrdat'         : CICS resource definition file
        - 'ESPGSQLXA.dll'    : XA switch module Windows
        - 'ESPGSQLXA64.so'   : XA switch module Linux
        - 'ESPGSQLXA64_S.so' : XA switch module Linux
        - 'EXTFH.cfg'        : Micro Focus File Handler configuration
```

```
          - 'm2-2021-04-28.normal.json' : M2 request document
          - 'MFDBFH.cfg'       : Micro Focus Database File Handler
          - 'application-definition-template-config.json' : Application definition for
 M2
    - source:
        subdirs:
        - .settings:
            files:
              - '.bms.mfdirset'
              - '.cbl.mfdirset'
        - copybook:
            files:
              - '*.cpy'
              - '*.inc'
            readme: |
              # Copy folder
              This folder contains the source for COBOL copy books, PLI includes, ...
              - .cpy COBOL copybooks
              - .inc PLI includes
#        - ctlcards:
#            files:
#              - '*.ctl'
#              - 'KBNKSRT1.txt'
#            readme: |
#              # Control Card folder
#              This folder contains the source for Batch Control Cards
#              - .ctl Control Cards
        - ims:
            files:
              - '*.dbd'
              - '*.psb'
            readme: |
              # ims folder
              This folder contains the IMS DB source files with the extensions
              - .dbd for IMS DBD source
              - .psb for IMS PSB source
        - jcl:
            files:
              - '*.jcl'
              - '*.ctl'
              - 'KBNKSRT1.txt'
              - '*.prc'
            readme: |
              # jcl folder
```

```
              This folder contains the JCL source files with the extensions
              - .jcl
#        - proclib:
#            files:
#              - '*.prc'
#            readme: |
#              # proclib folder
#              This folder contains the JCL procedures referenced via PROCLIB
 statements in the JCL with extensions
#              - .prc
        - rdbms:
            files:
              - '*.sql'
            readme: |
              # rdbms folder
              This folder contains any DB2 related source files with extensions
              - .sql for any kind of SQL source
        - screens:
            files:
            - '*.bms'
            - '*.mfs'
            readme: |
              # screens folder
              This folder contains the screens source files with the extensions
              - .bms for CICS BMS screens
              - .mfs for IMS MFS screens
            subdirs:
            - .settings:
                files:
                    - '*.bms.mfdirset'
        - cobol:
            files:
              - '*.cbl'
              - '*.pli'
            readme: |
              # source folder
              This folder contains the program source files with the extensions
              - .cbl for COBOL source
              - .pli for PLI source
            subdirs:
            - .settings:
                files:
                   - '*.cbl.mfdirset'
    - tests:
```

```
        files:
        - 'test_script.py'
        readme: |
          # tests Folder
          This folder contains the application test scripts
pipeline-config:
  alert-notifications:
    - myname@mycompany.com
  approval-notifications:
    - myname@mycompany.com
```

创建 Enterprise Developer AppStream 2.0

要在 AppStream 2.0 上设置 Micro Focus Enterprise Developer，请参阅教程：在 AppStream 2.0 上设置 Micro Focus Enterprise Developer。

要将 CodeCommit 存储库连接到 Enterprise Developer，请使用示例 YAML 触发器文件 config_git.yml中 `target-repository` 指定的名称。

Enterprise Developer 设置和测试

主题

- 在 Enterprise Developer 中克隆 BankDemo CodeCommit 存储库
- 创建 BankDemo 大型机 COBOL 项目并构建应用程序
- 创建用于测试的本地 BankDemo CICS 和批处理环境
- 从 Enterprise Developer 启动 BANKDEMO 服务器
- 启动 Rumba 3270 终端
- 运行 BankDemo 事务
- 从 Enterprise Developer 停止 BANKDEMO 服务器

连接到您在创建 Enterprise Developer AppStream 2.0 中创建的 Enterprise Developer AppStream 2.0 实例。

1. 从 Windows“开始”中，启动 Enterprise Developer。选择“Micro Focus Enterprise Developer”，然后选择“Enterprise Developer for Eclipse”。如果您是初次启动，则可能需要一些时间。
2. 在 Eclipse 启动器中的工作区中，输入 `C:\Users\<username>\workspace` 然后选择启动。

> **Note**
>
> 请确保在重新连接到 AppStream 2.0 实例后选择相同的位置。工作区选择不是永久性的。

3. 在欢迎中，选择打开 COBOL 透视图。这仅在新工作区首次使用时显示。

在 Enterprise Developer 中克隆 BankDemo CodeCommit 存储库

1. 选择窗口 / 透视图 / 打开透视图 / 其他... / Git。
2. 选择克隆 Git 存储库。
3. 在克隆 Git 存储库中，输入以下信息：
   - 在位置 URI 中，输入 CodeCommit 存储库的 HTTPS URL。

     > **Note**
     >
     > 在 AWS Management Console 中复制 CodeCommit 存储库的克隆 URL HTTPS 并将其粘贴到此处。URI 将分为主机和存储库路径。

   - 在身份验证用户和密码中，输入用户 CodeCommit 存储库凭证，然后在安全存储中选择存储。
4. 在分支选择中，选择主分支，然后选择下一步。
5. 在本地目标的目录中，输入 `C:\Users\<username>\workspace` 并选择完成。

   Git 存储库视图中显示 `BANKDEMO [main]` 时，即表示克隆过程完成。

创建 BankDemo 大型机 COBOL 项目并构建应用程序

1. 更改为 COBOL 透视图。
2. 在项目 中，禁用自动构建。
3. 在文件中，选择新建，然后选择大型机 COBOL 项目。
4. 在新建大型机 COBOL 项目中，输入以下信息：
   - 在项目名称中，输入 `BankDemo`。
   - 选择 Micro Focus 模板 [64 位]。
   - 选择完成。

5. 在 COBOL 资源管理器 中，展开新的 BankDemo 项目。

> **Note**
>
> 方括号中的 `[BANKDEMO main]` 表示该项目已连接到本地 BankDemo CodeCommit 存储库。

6. 如果树视图未显示 COBOL 程序、Copybook、BMS 源和 JCL 文件的条目，请从 BankDemo 项目上下文菜单中选择刷新。
7. 从 BankDemo 上下文菜单中选择属性 / Micro Focus / 项目设置 / COBOL：
   - 选择字符集 – ASCII。
   - 选择应用，然后选择确定。
8. 如果 BMS 和 COBOL 源的构建没有立即启动，在项目菜单中检查是否已启用自动构建选项。

   构建输出将显示在控制台视图中，应在几分钟后完成，并显示消息 `BUILD SUCCESSFUL` 和 `Build finished with no errors`。

   BankDemo 应用程序现在应该已经编译完毕，可以进行本地执行。

创建用于测试的本地 BankDemo CICS 和批处理环境

1. 在 COBOL 资源管理器中，展开 `BANKDEMO / config`。
2. 在编辑器中，打开 `BANKDEMO_ED.json`。
3. 找到字符串 `ED_Home=` 并更改路径以指向 Enterprise Developer 项目，如下所示：`D:\\<username>\\workspace\\BANKDEMO`。请注意在路径定义中使用双斜杠 (\\)。
4. 保存并关闭文件。
5. 选择服务器资源管理器。
6. 从默认上下文菜单中，选择打开管理页面。随后，Micro Focus Enterprise Server 管理页面在默认浏览器中打开。
7. （仅适用于 AppStream 2.0 会话）进行以下更改，以便可以保存本地 Enterprise Serve 区域以供本地测试使用：
   - 在目录服务器 / 默认中，选择属性 / 配置。
   - 将存储库位置替换为 `D:\<username>\My Files\Home Folder\MFDS`。

> **Note**
>
> 每次与 AppStream 2.0 实例建立新连接后，都必须完成步骤 5 – 8。

8. 在目录服务器 / 默认中，选择导入，然后完成以下步骤：

   - 在步骤 1：导入类型中，选择 JSON，然后选择下一步。
   - 在步骤 2：上传中，单击以在蓝色方块中上传文件。
   - 在选择要上传的文件中，输入：
     - 文件名：`D:\<username>\workspace\BANKDEMO\config\BANKDEMO_ED.json`。
     - 选择 打开。
   - 选择下一步。
   - 在步骤 3 中：区域中，取消选中从端点清除端口。
   - 选择下一步。
   - 在步骤 4：导入中，选择导入。
   - 选择完成。

   此时，列表中将显示一个新的服务器名称 BANKDEMO。

从 Enterprise Developer 启动 BANKDEMO 服务器

1. 选择Enterprise Developer。
2. 在服务器资源管理器中，选择默认，然后从上下文菜单中选择刷新。

   此时，服务器列表中也应该显示 BANKDEMO。
3. 选择 BANKDEMO。
4. 从上下文菜单中，选择与项目关联，然后选择 BANKDEMO。
5. 从上下文菜单中，选择启动。

   控制台视图中应显示服务器启动的日志。

   如果显示消息 `BANKDEMO CASSI5030I PLTPI Phase 2 List(PI) Processing Completed`，则服务器已准备好用于测试 CICS BANKDEMO 应用程序。

启动 Rumba 3270 终端

1. 从 Windows“开始”中，启动 Micro Focus Rumba+ Desktop / Rumba+ Desktop。
2. 在欢迎中，选择创建新会话 / 大型机显示。
3. 在大型机显示中，选择连接 / 配置。
4. 在会话配置中，选择连接 / TN3270。
5. 在主机名 / 地址中，选择插入并输入 IP 地址 `127.0.0.1`。
6. 在 Telnet 端口中，输入端口 `6000`。
7. 选择应用。
8. 选择连接。

   CICS 欢迎屏幕显示的屏幕带有第 1 行消息：`This is the Micro Focus MFE CICS region BANKDEMO`。
9. 按 Ctrl+Shift+Z 可清除屏幕。

运行 BankDemo 事务

1. 在空白屏幕中，输入 BANK。
2. 在屏幕 BANK10 中，在用户 ID...：输入字段中，输入 `guest` 并按 Enter。
3. 在屏幕 BANK20 中，在计算贷款成本之前的输入字段中，输入 `/`（正斜杠），然后按 Enter。
4. 在屏幕 BANK70 中：
   - 在希望借贷的金额...：中，输入 `10000`。
   - 在利率为...：中，输入 `5.0`。
   - 在借期（月）为...：中，输入 `10`。
   - 按 Enter。

   此时应显示以下结果：

   ```
   Resulting monthly payment.............:   $1023.06
   ```

   Enterprise Developer 中的 BANKDEMO 应用程序设置到此完成。

从 Enterprise Developer 停止 BANKDEMO 服务器

1. 在服务器资源管理器中，选择默认，然后从上下文菜单中选择刷新。
2. 选择 BANKDEMO。
3. 从上下文菜单中，选择停止。

   控制台视图中应显示服务器停止的日志。

   如果显示消息 `Server: BANKDEMO stopped successfully`，即表示服务器已成功关闭。

练习 1：在 BANKDEMO 应用程序中增强贷款计算

主题

在此场景中，逐步完成对代码进行示例更改、部署和测试的过程。

贷款部门希望在贷款计算屏幕 BANK70 上添加一个新字段来显示贷款总额。这需要更改 BMS 屏幕 MBANK70.CBL，添加一个新字段和相应的屏幕处理程序 SBANK70P.CBL 以及相关的copybook。此外，BBANK70P.CBL 中的贷款计算例程需要使用额外的公式进行扩展。

要完成本练习，请确保满足以下先决条件。

- 将 BANKDEMO-exercise.zip 下载到 `D:\PhotonUser\My Files\Home Folder`。
- 将 zip 文件解压缩到 `D:\PhotonUser\My Files\Home Folder\BANKDEMO-exercise`。
- 创建文件夹 `D:\PhotonUser\My Files\Home Folder\AnalysisRules`。
- 将规则文件 `Loan+Calculation+Update.General-1.xml` 从 `BANKDEMO-exercise` 文件夹复制到 `D:\PhotonUser\My Files\Home Folder\AnalysisRules`。

> **Note**
>
> 在本练习中，*.CBL 和 *.CPY 中的代码更改在 1 – 6 列用 EXER01 标记。

向 Enterprise Developer 代码分析添加贷款分析规则

可从 Enterprise Analyzer 导出 Micro Focus Enterprise Analyzer 中定义的分析规则，并将其导入到 Enterprise Developer 中，以便在 Enterprise Developer 项目中跨源运行相同的分析规则。

1. 打开 `Window/Preferences/Micro Focus/COBOL/Code Analysis/Rules`。
2. 选择编辑... 并输入文件夹名称 `D:\PhotonUser\My Files\Home Folder \AnalysisRules`（其中包含规则文件 `Loan+Calculation+Update.General-1.xml`）。
3. 选择完成。
4. 选择应用，然后选择关闭。
5. 从 BANKDEMO 项目上下文菜单中，选择代码分析。

   您会看到贷款计算更新条目。

步骤 1：执行贷款计算代码分析

借助新的分析规则，我们可以在表达式、语句和变量中识别出与搜索模式 *PAYMENT*、*LOAN* 和 *RATE* 相匹配的 COBOL 程序和代码行。这将有助于浏览代码并确定所需的代码更改。

1. 从 BANKDEMO 项目上下文菜单中，选择代码分析/贷款计算更新。

   该操作将运行搜索规则，并在名为代码分析的新选项卡中列出结果。当右下角的绿色进度条消失时，即表示分析运行完成。

   代码分析选项卡应显示 `BBANK20P.CBL`、`BBANK70P.CBL` 和 `SBANK70P.CBL` 的扩展列表，每个列表都列出了与搜索模式匹配的语句、表达式和变量。

   从 `BBANK20P.CBL` 的结果来看，只移动了与搜索模式匹配的文字。因此，可以忽略此程序。
2. 在选项卡菜单栏中，选择 - 图标，可全部折叠。
3. 双击，按任意顺序展开 `SBANK70P.CBL` 并选择任意行，查看如何打开源代码并突出显示源代码中选定的行。您还将看到，所有已识别的源行都已标记。

步骤 2：修改 CICS BMS 映射和 COBOL 程序并进行测试

首先，我们更改 BMS 映射 MBANK70.BMS、屏幕处理程序 SBANK70P.CBL 和 copybook CBANKDAT.CPY，以便显示新字段。为避免在本练习中进行不必要的编码，D:\PhotonUser\My Files\Home Folder\BANKDEMO-exercise\Exercise01 文件夹中提供了修改后的源模块。通常，开发人员会使用代码分析结果来浏览和修改源代码。如果您有时间并想手动进行更改，请使用 *MBANK70.BMS 和 SBANK70P.CBL 中的手动更改（可选）* 中提供的信息进行手动更改。

要进行快速更改，请复制以下文件：

1. ..\BANKDEMO-exercise\Exercis01\screens\MBANK70.BMS 到 D:\PhotonUser\workspace\bankdemo\source\screens。
2. .\BANKDEMO-exercise\Exercis01\cobol\SBANK70P.CBL 到 D:\PhotonUser\workspace\bankdemo\source\cobol。
3. ..\BANKDEMO-exercise\Exercis01\copybook\CBANKDAT.CPY 到 D:\PhotonUser\workspace\bankdemo\source\copybook。
4. 为确保受更改影响的所有程序都进行编译，请选择 项目/清理.../清理所有项目。

要手动更改 MBANK70.BMS 和 SBANK70P.CBL，请完成以下步骤：

- 要手动更改 BMS MBANK70.BMS 源，请在 PAYMENT 字段后面添加：
  - TXT09（具有与 TXT08 相同的属性，INITIAL 值为“贷款总额”）
  - TOTAL（与 PAYMENT 具有相同属性）

测试更改

要测试这些更改，请重复以下各节中的步骤：

1. 从 Enterprise Developer 启动 BANKDEMO 服务器
2. 启动 Rumba 3270 终端
3. 运行 BankDemo 事务

   此外，您此时还应该看到文本 Total Loan Amount.....:。
4. 从 Enterprise Developer 停止 BANKDEMO 服务器

步骤 3：在 COBOL 程序中添加总金额计算

在第二步中，我们更改 BBANK70P.CBL 并添加贷款总额的计算。D:\PhotonUser\My Files\Home Folder\BANKDEMO-exercise\Exercise01 文件夹中提供经过必要更改的可用源。如果您有时间并想手动进行更改，请使用 *BBANK70P.CBL 中的手动更改（可选）* 中提供的信息进行手动更改。

要进行快速更改，请复制以下文件：

- ..\BANKDEMO-exercise\Exercis01\source\cobol\BBANK70P.CBL 到 D:\PhotonUser\workspace\bankdemo\source\cobol。

要手动更改 BBANK70P.CBL，请完成以下步骤：

- 使用代码分析结果来确定所需的更改。

测试更改

要测试这些更改，请重复以下各节中的步骤：

1. 从 Enterprise Developer 启动 BANKDEMO 服务器
2. 启动 Rumba 3270 终端
3. 运行 BankDemo 事务

   此外，您此时还应该看到文本 Total Loan Amount.....: $10230.60。
4. 从 Enterprise Developer 停止 BANKDEMO 服务器

步骤 4：提交更改并运行 CI/CD 管道

将更改提交到中央 CodeCommit 存储库并触发 CI/CD 管道来构建、测试和部署更改。

1. 从 BANKDEMO 项目的上下文菜单中，选择团队/提交。
2. 在 Git 暂存选项卡中，输入以下提交消息：Added Total Amount Calculation。
3. 选择提交并推送...。
4. 打开 CodePipeline 控制台并查看管道执行状态。

> **Note**
>
> 如果您在使用 Enterprise Developer 或团队功能“提交”或“推送”时遇到任何问题，请使用 Git Bash 命令行界面。

练习 2：在 BANKDEMO 应用程序中提取贷款计算

主题

在本练习中，您将完成另一个示例变更请求。在此场景中，贷款部门希望将贷款计算例程作为独立的 Web 服务重复使用。该例程应保留在 COBOL 中，并且仍然可以从现有的 CICS COBOL 程序 `BBANK70P.CBL` 中调用。

步骤 1：将贷款计算例程重构为 COBOL 部分

首先，我们将贷款计算例程提取到 COBOL部分中。需要执行此步骤才能在下一步骤中将代码提取到独立的 COBOL 程序中。

1. 在 COBOL 编辑器中打开 `BBANK70P.CBL`。
2. 在编辑器中，从上下文菜单中选择代码分析/贷款计算更新。这只会扫描当前来源中是否存在分析规则中定义的模式。
3. 在代码分析选项卡的结果中，找到第一条算术语句 `DIVIDE WS-LOAN-INTEREST BY 12`。
4. 双击该语句可在编辑器中导航到源代码行。这是贷款计算例程的第一个语句。
5. 标记贷款计算例程的以下代码块，以便将提取到某个部分。

```
DIVIDE WS-LOAN-INTEREST BY 12
             GIVING WS-LOAN-INTEREST ROUNDED.
           COMPUTE WS-LOAN-MONTHLY-PAYMENT ROUNDED =
             ((WS-LOAN-INTEREST * ((1 + WS-LOAN-INTEREST)
                 ** WS-LOAN-TERM)) /
             (((1 + WS-LOAN-INTEREST) * WS-LOAN-TERM) - 1 ))
                 * WS-LOAN-PRINCIPAL.
```

```
EXER01     COMPUTE WS-LOAN-TOTAL-PAYMENT =
EXER01             (WS-LOAN-MONTHLY-PAYMENT * WS-LOAN-TERM).
```

6. 从编辑器的上下文菜单中，选择重构/提取到部分...。
7. 输入新部分名称：LOAN-CALCULATION。
8. 选择“确定”。

   标记的代码块现已提取到新的 LOAN-CALCULATION 部分，并且该代码块已替换为 PERFROM LOAN-CALCULATION 语句。

测试更改

要测试这些更改，请重复以下各节中所述的步骤：

1. 从 Enterprise Developer 启动 BANKDEMO 服务器
2. 启动 Rumba 3270 终端
3. 运行 BankDemo 事务

   此外，您此时还应该看到文本 Total Loan Amount.....: $10230.60。
4. 从 Enterprise Developer 停止 BANKDEMO 服务器

> **Note**
>
> 如果要避免上述步骤将代码块提取到某个部分，则可以将步骤 1 中的已修改源从 ..\BANKDEMO-exercise\Exercis02\Step1\cobol\BBANK70P.CBL 复制到 D:\PhotonUser\workspace\bankdemo\source\cobol。

步骤 2：将贷款计算例程提取到独立的 COBOL 程序中

在步骤 2 中，LOAN-CALCULATION 部分中的代码块将被提取到独立程序中，原始代码将替换为用于调用新子程序的代码。

1. 在编辑器中打开 BBANK70P.CBL，找到在步骤 1 中创建的新 PERFORM LOAN-CALCULATION 语句。
2. 将光标置于部分名称内。它将被标记为灰色。
3. 从上下文菜单中，选择重构->提取部分/段落到程序...。

4. 在提取部分/段落到程序中，输入新文件名：LOANCALC.CBL。
5. 选择确定。

   新的 LOANCALC.CBL 程序将在编辑器中打开。
6. 向下滚动并查看正在为调用接口提取和生成的代码。
7. 在编辑器中选择 BBANK70P.CBL，然后转至 LOAN-CALCULATION SECTION。查看正在生成的用于调用新子程序 LOANCALC.CBL 的代码。

> **Note**
>
> CALL 语句正在使用 DFHEIBLK 和 DFHCOMMAREA 调用带 CICS 控制块的 LOANCALC。由于我们希望将新的 LOANCALC.CBL 子程序作为非 CICS 程序调用，因此必须通过注释或删除从调用中移除 DFHEIBLK 和 DFHCOMMAREA。

测试更改

要测试这些更改，请重复以下各节中所述的步骤：

1. 从 Enterprise Developer 启动 BANKDEMO 服务器
2. 启动 Rumba 3270 终端
3. 运行 BankDemo 事务

   此外，您此时还应该看到文本 Total Loan Amount.....: $10230.60。
4. 从 Enterprise Developer 停止 BANKDEMO 服务器

> **Note**
>
> 如果要避免上述步骤将代码块提取到某个部分，则可以将步骤 1 中的已修改源从 ..\BANKDEMO-exercise\Exercis02\Step2\cobol\BBANK70P.CBL 和 LOANCALC.CBL 复制到 D:\PhotonUser\workspace\bankdemo\source\cobol。

步骤 3：提交更改并运行 CI/CD 管道

将更改提交到中央 CodeCommit 存储库并触发 CI/CD 管道来构建、测试和部署更改。

1. 从 BANKDEMO 项目的上下文菜单中，选择团队/提交。
2. 在 Git 暂存选项卡中
   - 在未暂存的阶段 LOANCALC.CBL 和 LOANCALC.CBL.mfdirset 中，添加内容。
   - 输入提交消息：`Added Total Amount Calculation`。
3. 选择提交并推送...。
4. 打开 CodePipeline 控制台并查看管道执行状态。

> **Note**
>
> 如果您在使用 Enterprise Developer 或团队功能“提交”或“推送”时遇到任何问题，请使用 Git Bash 命令行界面。

清理资源

如果您不再需要为本教程创建的资源，请将其删除，以免继续产生费用。完成以下步骤：

- 删除 CodePipeline 管道。有关更多信息，请参阅《AWS CodePipeline 用户指南》中的在 CodePipeline 中删除管道。
- 删除 CodeCommit 存储库。有关更多信息，请参阅《AWS CodeCommit 用户指南》中的删除 CodeCommit 存储库。
- 删除 S3 存储桶。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的删除存储桶。
- 删除 AWS CloudFormation 堆栈。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的在 AWS CloudFormation 控制台上删除堆栈。

AWS 大型机现代化中的 Batch 实用工具

大型机应用程序通常使用批处理实用程序来执行特定功能，例如对数据进行排序、使用 FTP 传输文件、将数据加载到 DB2 等数据库、从数据库中卸载数据等。

当您将应用程序迁移到 AWS 大型机现代化时，您需要功能等同的替代实用程序，这些实用程序可以执行与您在大型机上使用的相同任务。其中一些实用程序可能已经作为 AWS 大型机现代化运行时引擎的一部分提供，但我们提供了以下替代实用程序：

- M2SFTP – 使用 SFTP 协议启用安全的文件传输。

- M2WAIT – 等待指定时间后，再继续执行批处理作业中的下一步操作。
- TXT2PDF – 将文本文件转换为 PDF 格式。
- M2DFUTIL - 为数据集提供备份、恢复、删除和复制功能，类似于大型机 ADRDSSU 实用程序所提供的支持。
- M2RUNCMD - 允许您直接从 JCL 运行 Micro Focus 命令、脚本和系统调用。

我们根据客户反馈开发了这些批处理实用程序，旨在使其提供与大型机实用程序相同的功能。目标是尽可能顺利地从大型机过渡到 AWS 大型机现代化。

主题

二进制位置

这些实用程序预装在 Micro Focus Enterprise Developer (ED) 和 Micro Focus Enterprise Server (ES) 产品上。您可以在以下位置找到 ED 和 ES 所有变体的实用程序：

- Linux：`/opt/aws/m2/microfocus/utilities/64bit`
- Windows（32 位）：`C:\AWS\M2\MicroFocus\Utilities\32bit`
- Windows（64 位）：`C:\AWS\M2\MicroFocus\Utilities\64bit`

M2SFTP 批处理实用程序

M2SFTP 是一个 JCL 实用程序，旨在使用安全文件传输协议 (SFTP) 在系统之间执行安全的文件传输。该程序使用 Putty SFTP 客户端 `psftp` 来执行实际的文件传输。该程序的工作原理与大型机 FTP 实用程序类似，使用用户和密码进行身份验证。

> **Note**
>
> 不支持使用公钥进行身份验证。

要将您的大型机 FTP JCL 转换为使用 SFTP，请将 PGM=FTP 更改为 PGM=M2SFTP。

主题

支持的平台

您可以在以下任何平台上使用 M2SFTP：

- AWS 大型机现代化 Micro Focus 管理
- Micro Focus 运行时（在 Amazon EC2 上）
- Micro Focus Enterprise Developer (ED) 和 Micro Focus Enterprise Server (ES) 产品的所有变体。

安装依赖项

在 Windows 上安装 Putty SFTP 客户端

- 下载并安装 Putty SFTP 客户端。

在 Linux 上安装 Putty SFTP 客户端：

- 运行以下命令来安装 Putty SFTP 客户端：

```
sudo yum -y install putty
```

配置 M2SFTP 以实现 AWS 大型机现代化托管

如果您迁移的应用程序在 AWS 大型机现代化管理版上运行，则需要按如下方式配置 M2SFTP。

- 为 MFFTP 设置相应的 Micro Focus Enterprise Server 环境变量。下面是几个示例：
  - `MFFTP_TEMP_DIR`
  - `MFFTP_SENDEOL`
  - `MFFTP_TIME`
  - `MFFTP_ABEND`

  您可以根据需要设置任意数量的变量。您可以在 JCL 中使用 `ENVAR DD` 语句设置它们。有关这些变量的更多信息，请参阅 Micro Focus 文档中的 MFFTP 控制变量。

要测试配置，请参阅示例 JCL。

在 Amazon EC2（包括 AppStream 2.0）上为 AWS 大型机现代化运行时配置 M2SFTP

如果您迁移的应用程序在 Amazon EC2 AWS 的大型机现代化运行时上运行，请按如下方式配置 M2SFTP。

1. 更改 Micro Focus JES 程序路径以包含批处理实用程序的二进制位置。如果您需要指定多个路径，请在 Linux 上使用冒号 (`:`) 分隔路径，在 Windows 上使用分号 (`;`) 分隔路径。
   - Linux：`/opt/aws/m2/microfocus/utilities/64bit`
   - Windows（32 位）：`C:\AWS\M2\MicroFocus\Utilities\32bit`
   - Windows（64 位）：`C:\AWS\M2\MicroFocus\Utilities\64bit`
2. 为 MFFTP 设置相应的 Micro Focus Enterprise Server 环境变量。下面是几个示例：
   - `MFFTP_TEMP_DIR`
   - `MFFTP_SENDEOL`
   - `MFFTP_TIME`
   - MFFTP_ABEND

您可以根据需要设置任意数量的变量。您可以在 JCL 中使用 ENVAR DD 语句设置它们。有关这些变量的更多信息，请参阅 Micro Focus 文档中的 MFFTP 控制变量。

要测试配置，请参阅示例 JCL。

示例 JCL

要测试安装，您可以使用以下其中一个示例 JCL 文件。

M2SFTP1.jcl

此 JCL 展示了如何调用 M2SFTP，以便将文件发送到远程 SFTP 服务器。请注意 ENVVAR DD 语句中设置的环境变量。

```
//M2SFTP1 JOB 'M2SFTP1',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*-------------------------------------------------------------------**
//* Sample SFTP JCL step to send a file to SFTP server*
//*-------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=M2SFTP,
//            PARM='127.0.0.1 (EXIT=99 TIMEOUT 300'
//*
//SYSFTPD  DD  *
RECFM FB
LRECL 80
SBSENDEOL CRLF
MBSENDEOL CRLF
TRAILINGBLANKS FALSE
/*
//NETRC    DD  *
machine 127.0.0.1 login sftpuser password sftppass
/*
//SYSPRINT DD  SYSOUT=*
//OUTPUT   DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//INPUT    DD  *
type a
```

```
locsite notrailingblanks
cd files
put 'AWS.M2.TXT2PDF1.PDF' AWS.M2.TXT2PDF1.pdf
put 'AWS.M2.CARDDEMO.CARDDATA.PS' AWS.M2.CARDDEMO.CARDDATA.PS1.txt
quit
/*
//ENVVAR   DD *
MFFTP_VERBOSE_OUTPUT=ON
MFFTP_KEEP=N
/*
//*
//
```

M2SFTP2.jcl

此 JCL 展示了如何调用 M2SFTP，以便接收来自远程 SFTP 服务器的文件。请注意 ENVVAR DD 语句中设置的环境变量。

```
//M2SFTP2 JOB 'M2SFTP2',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*--------------------------------------------------------------------**
//* Sample SFTP JCL step to receive a file from SFTP server*
//*--------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=M2SFTP
//*
//SYSPRINT DD  SYSOUT=*
//OUTPUT   DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//INPUT    DD  *
open 127.0.0.1
sftpuser
sftppass
cd files
locsite recfm=fb lrecl=150
get AWS.M2.CARDDEMO.CARDDATA.PS.txt +
'AWS.M2.CARDDEMO.CARDDATA.PS2' (replace
quit
/*
//ENVVAR   DD *
```

```
MFFTP_VERBOSE_OUTPUT=ON
MFFTP_KEEP=N
/*
//*
//
```

> **Note**
>
> 我们强烈建议将 FTP 凭证存储在 NETRC 文件中，并限制只有经过授权的用户才能访问。

Putty SFTP (PSFTP) 客户端命令参考

PSFTP 客户端不支持所有 FTP 命令。以下列表显示了 PSFTP 支持的所有命令。

命令	描述
!	运行本地命令
bye	完成 SFTP 会话
cd	更改远程工作目录
chmod	更改文件权限和模式
close	完成 SFTP 会话但不退出 PSFTP
del	删除远程服务器上的文件
dir	列出远程文件
exit	完成 SFTP 会话
get	将文件从服务器下载到本地机器上
help	提供帮助
lcd	更改本地工作目录
lpwd	打印本地工作目录

命令	描述
ls	列出远程文件
mget	一次下载多个文件
mkdir	在远程服务器上创建目录
mput	一次上传多个文件
mv	移动或重命名远程服务器上的文件
open	连接到主机
put	将文件从本地机器上传到服务器
pwd	打印远程工作目录
quit	完成 SFTP 会话
reget	继续下载文件
ren	移动或重命名远程服务器上的文件
reput	继续上传文件
rm	删除远程服务器上的文件
rmdir	删除远程服务器上的目录

后续步骤

要使用 SFTP 将文件上传和下载到亚马逊简单存储服务，您可以将 M2SFTP 与结合使用 AWS Transfer Family，如以下博客文章中所述。

- 使用 AWS SFTP 逻辑目录构建简单的数据分发服务
- 启用密码身份验证以供 AWS Transfer for SFTP 使用 AWS Secrets Manager

M2WAIT Batch 实用程序

M2WAIT 是一个大型机实用程序，允许您在 JCL 脚本中引入等待期，即通过以秒、分钟或小时为单位指定持续时间。您可以直接从 JCL 调用 M2WAIT，方法是将要等待的时间作为输入参数传递。在内部，M2WAIT 程序调用 Micro Focus 提供的模块 C$SLEEP 来等待指定的时间。

> **Note**
>
> 您可以使用 Micro Focus 别名来替换 JCL 脚本中的名称。有关更多信息，请参阅 Micro Focus 文档中的 JES 别名。

主题

支持的平台

您可以在以下任何平台上使用 M2WAIT：

- AWS 大型机现代化 Micro Focus 管理
- Micro Focus 运行时（在 Amazon EC2 上）
- Micro Focus Enterprise Developer (ED) 和 Micro Focus Enterprise Server (ES) 产品的所有变体。

配置 M2WAIT 以实现 AWS 大型机现代化托管

如果您迁移的应用程序在 AWS 大型机现代化管理版上运行，则需要按如下方式配置 M2WAIT。

- 通过传递输入参数（如示例 JCL 中所示）在 JCL 中使用 M2WAIT 程序。

在 Amazon EC2（包括 AppStream 2.0）上为 AWS 大型机现代化运行时配置 M2WAIT

如果您迁移的应用程序在 Amazon EC2 AWS 的大型机现代化运行时上运行，请按如下方式配置 M2WAIT。

1. 更改 Micro Focus JES 程序路径以包含批处理实用程序的二进制位置。如果您需要指定多个路径，请在 Linux 上使用冒号 (:) 分隔路径，在 Windows 上使用分号 (;) 分隔路径。

   - Linux：`/opt/aws/m2/microfocus/utilities/64bit`
   - Windows（32 位）：`C:\AWS\M2\MicroFocus\Utilities\32bit`
   - Windows（64 位）：`C:\AWS\M2\MicroFocus\Utilities\64bit`

2. 通过传递输入参数（如示例 JCL 中所示）在 JCL 中使用 M2WAIT 程序。

示例 JCL

要测试安装，您可以使用 `M2WAIT1.jcl` 程序。

此示例 JCL 展示了如何调用 M2WAIT 并采用几个不同的持续时间将其进行传递。

```
//M2WAIT1 JOB 'M2WAIT',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*--------------------------------------------------------------------**
//* Wait for 12 Seconds*
//*--------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=M2WAIT,PARM='S012'
//SYSOUT DD SYSOUT=*
//*
//*--------------------------------------------------------------------**
//* Wait for 0 Seconds (defaulted to 10 Seconds)*
//*--------------------------------------------------------------------**
//*
//STEP02 EXEC PGM=M2WAIT,PARM='S000'
//SYSOUT DD SYSOUT=*
//*
//*--------------------------------------------------------------------**
//* Wait for 1 Minute*
//*--------------------------------------------------------------------**
//*
//STEP03 EXEC PGM=M2WAIT,PARM='M001'
//SYSOUT DD SYSOUT=*
//*
//
```

TXT2PDF 批处理实用程序

TXT2PDF 是一个大型机实用程序，通常用于将文本文件转换为 PDF 文件。此实用程序与 TXT2PDF（z/OS 免费软件）使用相同的源代码。我们对其进行了修改，使其在 AWS 大型机现代化 Micro Focus 运行时环境下运行。

主题

支持的平台

您可以在以下任何平台上使用 TXT2PDF：

- AWS 大型机现代化 Micro Focus 管理
- Micro Focus 运行时（在 Amazon EC2 上）
- Micro Focus Enterprise Developer (ED) 和 Micro Focus Enterprise Server (ES) 产品的所有变体。

配置 TXT2PDF 以实现 AWS 大型机现代化托管

如果您迁移的应用程序在 AWS 大型机现代化管理版上运行，请按以下方式配置 TXT2PDF。

- 创建一个名为 `AWS.M2.REXX.EXEC` 的 REXX EXEC 库。下载这些 REXX 模块并将其复制到库中。
  - `TXT2PDF.rex` – TXT2PDF z/OS 免费软件（已修改）
  - `TXT2PDFD.rex` – TXT2PDF z/OS 免费软件（未修改）
  - `TXT2PDFX.rex` – TXT2PDF z/OS 免费软件（已修改）
  - `M2GETOS.rex` – 检查操作系统类型（Windows 或 Linux）

要测试配置，请参阅示例 JCL。

在 Amazon EC2（包括 AppStream 2.0）上为 AWS 大型机现代化运行时配置 TXT2PDF

如果您迁移的应用程序在 Amazon EC2 AWS 的大型机现代化运行时上运行，请按如下方式配置 TXT2PDF。

1. 将 Micro Focus 环境变量 MFREXX_CHARSET 设置为适当的值，例如，为 ASCII 数据设置为“A”。

   > **Important**
   >
   > 输入错误的值可能会导致数据转换问题（从 EBCDIC 到 ASCII），使生成的 PDF 无法读取或无法操作。我们建议设置 MFREXX_CHARSET 来匹配 MF_CHARSET。

2. 更改 Micro Focus JES 程序路径以包含批处理实用程序的二进制位置。如果您需要指定多个路径，请在 Linux 上使用冒号 (:) 分隔路径，在 Windows 上使用分号 (;) 分隔路径。

   - Linux：`/opt/aws/m2/microfocus/utilities/64bit`
   - Windows（32 位）：`C:\AWS\M2\MicroFocus\Utilities\32bit`
   - Windows（64 位）：`C:\AWS\M2\MicroFocus\Utilities\64bit`

3. 创建一个名为 `AWS.M2.REXX.EXEC` 的 REXX EXEC 库。下载这些 REXX 模块并将其复制到库中。

   - `TXT2PDF.rex` – TXT2PDF z/OS 免费软件（已修改）
   - `TXT2PDFD.rex` – TXT2PDF z/OS 免费软件（未修改）
   - `TXT2PDFX.rex` – TXT2PDF z/OS 免费软件（已修改）
   - `M2GETOS.rex` – 检查操作系统类型（Windows 或 Linux）

要测试配置，请参阅示例 JCL。

示例 JCL

要测试安装，您可以使用以下其中一个示例 JCL 文件。

TXT2PDF1.jcl

此示例 JCL 文件使用 DD 名称进行 TXT2PDF 转换。

```
//TXT2PDF1 JOB 'TXT2PDF1',CLASS=A,MSGCLASS=X,TIME=1440
//*
```

```
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*-------------------------------------------------------------------**
//* PRE DELETE*
//*-------------------------------------------------------------------**
//*
//PREDEL  EXEC PGM=IEFBR14
//*
//DD01     DD DSN=AWS.M2.TXT2PDF1.PDF.VB,
//            DISP=(MOD,DELETE,DELETE)
//*
//DD02     DD DSN=AWS.M2.TXT2PDF1.PDF,
//            DISP=(MOD,DELETE,DELETE)
//*
//*-------------------------------------------------------------------**
//* CALL TXT2PDF TO CONVERT FROM TEXT TO PDF (VB)*
//*-------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=IKJEFT1B
//*
//SYSEXEC  DD DISP=SHR,DSN=AWS.M2.REXX.EXEC
//*
//INDD     DD *
1THIS IS THE FIRST LINE ON THE PAGE 1
0THIS IS THE THIRD LINE ON THE PAGE 1
-THIS IS THE   6TH LINE ON THE PAGE 1
THIS IS THE   7TH LINE ON THE PAGE 1
+_____________________________________ - OVERSTRIKE 7TH LINE
1THIS IS THE FIRST LINE ON THE PAGE 2
0THIS IS THE THIRD LINE ON THE PAGE 2
-THIS IS THE   6TH LINE ON THE PAGE 2
THIS IS THE   7TH LINE ON THE PAGE 2
+_____________________________________ - OVERSTRIKE 7TH LINE
/*
//*
//OUTDD    DD DSN=AWS.M2.TXT2PDF1.PDF.VB,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(LRECL=256,DSORG=PS,RECFM=VB,BLKSIZE=0)
//*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=SYSIN
//*
//SYSIN    DD *
```

```
%TXT2PDF BROWSE Y IN DD:INDD +
OUT DD:OUTDD +
CC YES
/*
//*
//*-------------------------------------------------------------------**
//* CONVERT PDF (VB) TO PDF (LSEQ - BYTE STREAM)*
//*-------------------------------------------------------------------**
//*
//STEP02 EXEC PGM=VB2LSEQ
//*
//INFILE   DD DSN=AWS.M2.TXT2PDF1.PDF.VB,DISP=SHR
//*
//OUTFILE  DD DSN=AWS.M2.TXT2PDF1.PDF,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(LRECL=256,DSORG=PS,RECFM=LSEQ,BLKSIZE=0)
//*
//SYSOUT   DD SYSOUT=*
//*
//
```

TXT2PDF2.jcl

此示例 JCL 文件使用 DSN 名称进行 TXT2PDF 转换。

```
//TXT2PDF2 JOB 'TXT2PDF2',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*-------------------------------------------------------------------**
//* PRE DELETE*
//*-------------------------------------------------------------------**
//*
//PREDEL  EXEC PGM=IEFBR14
//*
//DD01     DD DSN=AWS.M2.TXT2PDF2.PDF.VB,
//            DISP=(MOD,DELETE,DELETE)
//*
//DD02     DD DSN=AWS.M2.TXT2PDF2.PDF,
//            DISP=(MOD,DELETE,DELETE)
//*
//*-------------------------------------------------------------------**
//* CALL TXT2PDF TO CONVERT FROM TEXT TO PDF (VB)*
```

```
//*--------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=IKJEFT1B
//*
//SYSEXEC  DD DISP=SHR,DSN=AWS.M2.REXX.EXEC
//*
//INDD     DD *
1THIS IS THE FIRST LINE ON THE PAGE 1
0THIS IS THE THIRD LINE ON THE PAGE 1
-THIS IS THE   6TH LINE ON THE PAGE 1
THIS IS THE   7TH LINE ON THE PAGE 1
+_____________________________________ - OVERSTRIKE 7TH LINE
1THIS IS THE FIRST LINE ON THE PAGE 2
0THIS IS THE THIRD LINE ON THE PAGE 2
-THIS IS THE   6TH LINE ON THE PAGE 2
THIS IS THE   7TH LINE ON THE PAGE 2
+_____________________________________ - OVERSTRIKE 7TH LINE
/*
//*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=SYSIN
//*
//SYSIN    DD *
%TXT2PDF BROWSE Y IN DD:INDD +
OUT 'AWS.M2.TXT2PDF2.PDF.VB' +
CC YES
/*
//*
//*--------------------------------------------------------------------**
//* CONVERT PDF (VB) TO PDF (LSEQ - BYTE STREAM)*
//*--------------------------------------------------------------------**
//*
//STEP02 EXEC PGM=VB2LSEQ
//*
//INFILE   DD DSN=AWS.M2.TXT2PDF2.PDF.VB,DISP=SHR
//*
//OUTFILE  DD DSN=AWS.M2.TXT2PDF2.PDF,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(LRECL=256,DSORG=PS,RECFM=LSEQ,BLKSIZE=0)
//*
//SYSOUT   DD SYSOUT=*
//*
//
```

修改

为了让 TXT2PDF 程序在 AWS 大型机现代化 Micro Focus 运行时环境中运行，我们进行了以下更改：

- 为确保与 Micro Focus REXX 运行时兼容而对源代码进行了更改
- 为确保该程序可以同时在 Windows 和 Linux 操作系统上运行而进行了更改
- 为支持 EBCDIC 和 ASCII 运行时而进行了修改

参考信息

TXT2PDF 参考文献和源代码：

- 文本转 PDF 转换器
- z/OS 免费软件 TCP/IP 和邮件工具
- TXT2PDF 用户参考指南

M2DFUTIL 批处理实用程序

M2DFUTIL 是一个 JCL 实用程序，为数据集提供备份、恢复、删除和复制功能，类似于大型机 ADRDSSU 实用程序提供的支持。该程序保留了 ADRDSSU 中的许多 SYSIN 参数，从而简化了迁移到这个新实用程序的过程。

主题

- 支持的平台
- 平台要求
- 计划的未来支持
- 资产位置
- 在 Amazon EC2（包括 AppStream 2.0）上配置 M2DFUTIL 或 AWS 大型机现代化运行时
- 一般语法
- JCL 示例

支持的平台

您可以在以下任何平台上使用 M2DFUTIL：

- Windows 上的 Micro Focus ES（64 位和 32 位）
- Linux 上的 Micro Focus ES（64 位）

平台要求

M2DFUTIL 依赖于调用脚本来执行正则表达式测试。在 Windows 上，必须安装 Windows Services for Linux (WSL) 才能运行此脚本。

计划的未来支持

目前不可用但未来可用的大型机 ADRDSSU 实用程序功能包括：

- M2 托管
- VSAM
- 对文件重命名的 COPY 支持
- 对 RESTORE 的 RENAME 支持
- 多个 INCLUDE 和 EXCLUDE
- 通过 DSORG、CREDT、EXPDT 进行子查询的 BY 子句
- 用于重试入队失败的 MWAIT 子句
- 对 DUMP/RESTORE 的 S3 存储支持

资产位置

该实用程序的加载模块在 Linux 上称为 `M2DFUTIL.so`，在 Windows 上称为 `M2DFUTIL.dll`。可在以下位置找到此加载模块：

- Linux：`/opt/aws/m2/microfocus/utilities/64bit`
- Windows（32 位）：`C:\AWS\M2\MicroFocus\Utilities\32bit`
- Windows（64 位）：`C:\AWS\M2\MicroFocus\Utilities\64bit`

用于正则表达式测试的脚本称为 `compare.sh`。可在以下位置找到此脚本：

- Linux：`/opt/aws/m2/microfocus/utilities/scripts`
- Windows（32 位）：`C:\AWS\M2\MicroFocus\Utilities\scripts`

在 Amazon EC2（包括 AppStream 2.0）上配置 M2DFUTIL 或 AWS 大型机现代化运行时

使用以下内容配置您的 Enterprise Server 区域：

- 在 [ES-Environment] 中添加以下变量
  - M2DFUTILS_BASE_LOC - DUMP 输出的默认位置
  - M2DFUTILS_SCRIPTPATH - 资产位置中记录的 compare.sh 脚本的位置
  - M2DFUTILS_VERBOSE - [VERBOSE 或 NORMAL]。此变量控制 SYSPRINT 输出中的细节级别
- 验证加载模块路径是否已添加到 JES\Configuration\JES Program Path 设置中
- 验证实用程序目录中的脚本是否具有运行权限。您可以在 Linux 环境中使用 chmod + x <script name> 命令添加运行权限

一般语法

DUMP

提供将文件从当前编目位置复制到备份位置的功能。此位置当前必须是一个文件系统。

过程

DUMP 将执行下列操作：

1. 创建目标位置目录。
2. 将该目标位置目录归类为 PDS 成员。
3. 通过处理 INCLUDE 参数来确定要包含的文件。
4. 通过处理 EXCLUDE 参数来取消选择包含的文件。
5. 确定是否对转储的文件执行 DELETED 操作。
6. 将文件入队以待处理。
7. 复制这个文件。
8. 将已复制文件编目的 DCB 信息导出到目标位置中的副文件，以帮助将来执行 RESTORE 操作。

语法

```
DUMP
TARGET ( TARGET LOCATION  )    -
```

```
INCLUDE ( DSN. )
[ EXCLUDE ( DSN ) ]
[ CANCEL | IGNORE ]
[ DELETE ]
```

必需参数

以下是必需的 DUMP 参数：

- SYSPRINT DD NAME - 用于包含其他日志信息
- TARGET - 目标位置。它可以是：
  - 转储位置的完整路径
  - 在 M2DFUTILS_BASE_LOC 变量中定义的位置中创建的子目录名称
- INCLUDE - 要么是单个命名的 DSNAME，要么是有效的大型机 DSN 搜索字符串
- EXCLUDE - 要么是单个命名的 DSNAME，要么是有效的大型机 DSN 搜索字符串

可选参数

- CANCEL - 如果出现任何错误，则取消。已处理的文件将被保留
- （默认）IGNORE - 忽略任何错误并处理直到结束
- DELETE - 如果未出现 ENQ 错误，则会删除该文件并取消编目

删除

提供批量删除和取消编目文件的功能。文件未备份。

过程

DELETE 将执行下列操作：

1. 通过处理 INCLUDE 参数来确定要包含的文件。
2. 通过处理 EXCLUDE 参数来取消选择包含的文件。
3. 将文件入队以待处理。将处置设置为“OLD”、“DELETE”、“KEEP”。

语法

```
DELETE
```

```
INCLUDE ( DSN )
[ EXCLUDE ( DSN ) ]
[ CANCEL | IGNORE ]
[ DELETE ]
```

必需参数

以下是必需的 DELETE 参数：

- SYSPRINT DD NAME - 用于包含其他日志信息
- INCLUDE - 要么是单个命名的 DSNAME，要么是有效的大型机 DSN 搜索字符串
- EXCLUDE - 要么是单个命名的 DSNAME，要么是有效的大型机 DSN 搜索字符串

可选参数

- CANCEL - 如果出现任何错误，则取消。已处理的文件将被保留
- （默认）IGNORE - 忽略任何错误并处理直到结束

RESTORE

提供恢复以前使用 DUMP 备份的文件的功能。除非使用 RENAME 来更改已恢复的 DSNAME，否则文件将恢复到原始编目位置。

过程

RESTORE 将执行下列操作：

1. 验证源位置目录。
2. 通过处理目录导出文件来确定要包含的文件。
3. 通过处理 EXCLUDE 参数来取消选择包含的文件。
4. 将文件入队以待处理。
5. 未根据其导出信息编目的目录文件。
6. 如果已对文件进行编目并且导出目录信息相同，则当设置了 REPLACE 选项时，RESTORE 将替换编目的数据集。

语法

```
RESTORE
```

```
SOURCE ( TARGET LOCATION )
INCLUDE ( DSN )
[ EXCLUDE ( DSN ) ]
[ CANCEL | IGNORE ]
[ REPLACE]
```

必需参数

以下是必需的 RESTORE 参数：

- SYSPRINT DD NAME - 用于包含其他日志信息
- SOURCE - 源位置。它可以是：
  - 转储位置的完整路径
  - 在 M2DFUTILS_BASE_LOC 变量中定义的位置中创建的子目录名称
- INCLUDE - 要么是单个命名的 DSNAME，要么是有效的大型机 DSN 搜索字符串
- EXCLUDE - 要么是单个命名的 DSNAME，要么是有效的大型机 DSN 搜索字符串

可选参数

- CANCEL - 如果出现任何错误，则取消。已处理的文件已保留
- （默认）IGNORE - 忽略任何错误并处理直到结束
- REPLACE - 如果正在恢复的文件已经编目并且目录记录相同，则替换已编目文件

JCL 示例

DUMP 作业

此作业将创建一个名为 TESTDUMP 的子目录。这是 M2DFUTILS_BASE_LOC 变量指定的默认备份位置。它将为这个名为 M2DFUTILS.TESTDUMP 的备份创建一个 PDS 库。导出的目录数据存储在名为 CATDUMP.DAT 的备份目录中的行序文件中。所有选定文件都将复制到此备份目录。

```
//M2DFDMP JOB 'M2DFDMP',CLASS=A,MSGCLASS=X
//STEP001  EXEC PGM=M2DFUTIL
//SYSPRINT DD DSN=TESTDUMP.SYSPRINT,
//         DISP=(NEW,CATLG,DELETE),
//         DCB=(RECFM=LSEQ,LRECL=256)
//SYSIN     DD *
```

```
DUMP TARGET(TESTDUMP)               -
     INCLUDE(TEST.FB.FILE*.ABC)     -
 CANCEL
/*
//
```

DELETE 作业

此作业将从目录中删除与 INCLUDE 参数匹配的所有文件。

```
/M2DFDEL JOB 'M2DFDEL',CLASS=A,MSGCLASS=X
//STEP001  EXEC PGM=M2DFUTIL
//SYSPRINT DD DSN=TESTDEL.SYSPRINT,
//        DISP=(NEW,CATLG,DELETE),
//        DCB=(RECFM=LSEQ,LRECL=256)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
  DELETE                            -
     INCLUDE(TEST.FB.FILE*.ABC)     -
 CANCEL
/*
 //
```

RESTORE 作业

此作业将从 TESTDUMP 备份位置恢复与 INCLUDE 参数匹配的文件。如果编目文件与 CATDUMP 导出中的文件相同，并且指定了 REPLACE 选项，则编目文件将被替换。

```
//M2DFREST JOB 'M2DFREST',CLASS=A,MSGCLASS=X
//STEP001  EXEC PGM=M2DFUTIL
////SYSPRINT DD DSN=TESTREST.SYSPRINT,
//        DISP=(NEW,CATLG,DELETE),
//        DCB=(RECFM=LSEQ,LRECL=256)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
RESTORE SOURCE(TESTDUMP)            -
     INCLUDE(TEST.FB.FILE*.ABC)     -
 IGNORE
 REPLACE
/*
//
```

M2RUNCMD 批处理实用程序

您可以使用批处理实用程序 M2RUNCMD 直接从 JCL 运行 Micro Focus 命令、脚本和系统调用，而不必从终端或命令提示符处运行它们。命令的输出将记录到批处理作业的后台处理日志中。

主题

支持的平台

您可以在以下平台上使用 M2RUNCMD：

- Micro Focus 运行时（在 Amazon EC2 上）
- Micro Focus Enterprise Developer (ED) 和 Micro Focus Enterprise Server (ES) 产品的所有变体。

在 Amazon EC2（包括 AppStream 2.0）上为 AWS 大型机现代化运行时配置 M2RUNCMD

如果您迁移的应用程序在 Amazon EC2 AWS 的大型机现代化运行时上运行，请按如下方式配置 M2RUNCMD。

- 更改 Micro Focus JES 程序路径以包含批处理实用程序的二进制位置。如果您必须指定多个路径，请在 Linux 上使用冒号 (:) 分隔路径，在 Windows 上使用分号 (;) 分隔路径。
  - Linux：`/opt/aws/m2/microfocus/utilities/64bit`
  - Windows（32 位）：`C:\AWS\M2\MicroFocus\Utilities\32bit`
  - Windows（64 位）：`C:\AWS\M2\MicroFocus\Utilities\64bit`

JCL 示例

要测试安装，您可以使用以下示例 JCL 之一。

RUNSCRL1.jcl

此示例 JCL 创建一个脚本并运行它。第一步创建一个名为 /tmp/TEST_SCRIPT.sh 的脚本，其中包含来自 SYSUT1 流内数据的内容。第二步设置运行权限并运行第一步中所创建的脚本。您也可以选择仅执行第二步，来运行现有的 Micro Focus 和系统命令。

```
//RUNSCRL1 JOB 'RUN SCRIPT',CLASS=A,MSGCLASS=X,TIME=1440
//*
//*
//*--------------------------------------------------------------------*
//*  CREATE SCRIPT (LINUX)
//*--------------------------------------------------------------------*
//*
//STEP0010 EXEC PGM=IEBGENER
//*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//*
//SYSUT1   DD *
#!/bin/bash

set -x

## ECHO PATH ENVIRONMNET VARIABLE
echo $PATH

## CLOSE/DISABLE VSAM FILE
casfile -r$ES_SERVER -oc  -ed -dACCTFIL

## OPEN/ENABLE VSAM FILE
casfile -r$ES_SERVER -ooi -ee -dACCTFIL

exit $?
/*
//SYSUT2   DD DSN=&&TEMP,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(RECFM=LSEQ,LRECL=300,DSORG=PS,BLKSIZE=0)
//*MFE: %PCDSN='/tmp/TEST_SCRIPT.sh'
//*
//*--------------------------------------------------------------------*
//*   RUN SCRIPT (LINUX)                                               *
//*--------------------------------------------------------------------*
//*
//STEP0020 EXEC PGM=RUNCMD
//*
```

```
//SYSOUT  DD  SYSOUT=*
//*
//SYSIN   DD *
*RUN SCRIPT
 sh /tmp/TEST_SCRIPT.sh
/*
//
```

SYSOUT

运行的命令或脚本的输出将写入 SYSOUT 日志。对于每个已执行的命令，它都会显示命令、输出和返回代码。

```
************ CMD Start ************

CMD_STR: sh /tmp/TEST_SCRIPT.sh

CMD_OUT:


+ echo /opt/microfocus/EnterpriseServer/bin:/sbin:/bin:/usr/sbin:/usr/bin
/opt/microfocus/EnterpriseServer/bin:/sbin:/bin:/usr/sbin:/usr/bin
+ casfile -rMYDEV -oc -ed -dACCTFIL

-Return Code:   0

Highest return code:    0

+ casfile -rMYDEV -ooi -ee -dACCTFIL

-Return Code:   8

Highest return code:    8

+ exit 8


CMD_RC=8


************  CMD End  ************
```

RUNCMDL1.jcl

此 JCL 示例使用 RUNCMD 来运行多个命令。

```
//RUNCMDL1 JOB 'RUN CMD',CLASS=A,MSGCLASS=X,TIME=1440
//*
//*
//*--------------------------------------------------------------------*
//*   RUN SYSTEM COMMANDS                                              *
//*--------------------------------------------------------------------*
//*
//STEP0001 EXEC PGM=RUNCMD
//*
//SYSOUT  DD  SYSOUT=*
//*
//SYSIN   DD *
*LIST DIRECTORY
 ls
*ECHO PATH ENVIRONMNET VARIABLE
 echo $PATH
/*
//
```

AWS 使用 Precission 复制大型机现代化数据

AWS 大型机现代化提供各种亚马逊系统映像 (AMI)。这些 AMI 便于快速配置 Amazon EC2 实例，为从大型机系统复制到 AWS 使用 Precist 的数据创建了量身定制的环境。本指南提供访问和使用这些 AMI 所需的步骤。

先决条件

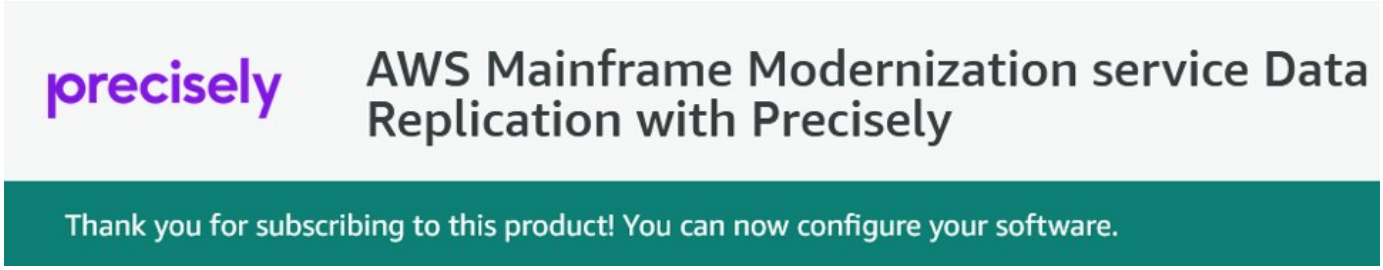
- 确保您拥有可以创建 Amazon EC2 实例的 AWS 账户的管理员访问权限。
- 确认 AWS 大型机现代化服务在您计划创建 Amazon EC2 实例的地区可用。查看按区域划分的可用 AWS 服务列表。
- 确定要在其中创建 Amazon EC2 实例的 Amazon Virtual Private Cloud（Amazon VPC）。
- 在 Amazon VPC 中创建 Amazon EC2 实例时，请确保关联的路由表具有互联网网关或 NAT 网关。

> **Note**
>
> 成功复制数据需要 AWS EC2 实例具有对 AWS Marketplace 的通信权限。如果 AWS Marketplace 出现连接问题，则复制过程将失败。

订阅亚马逊机器映像

当您订阅 AWS Marketplace 产品后，可以从该产品的 AMI 启动实例。

1. 登录 AWS Management Console 并打开 AWS Marketplace 控制台，网址为 https://console.aws.amazon.com/marketplace。
2. 选择管理订阅。
3. 将以下链接复制并粘贴到浏览器地址栏中：https://aws.amazon.com/marketplace/pp/prodview-en3xrbgzbs3dk
4. 选择继续订阅。
5. 如果可以接受条款和条件，请选择接受条款。处理订阅可能需要几分钟时间。
6. 等待出现感谢消息，如下所示。该消息确认您是否成功订阅了产品。



7. 在左侧导航窗格中，选择管理订阅。此视图显示您已订阅的所有订阅。

使用 Precist 启动 AWS 大型机现代化数据复制

1. 打开 AWS Marketplace 控制台，网址为 https://console.aws.amazon.com/marketplace。
2. 在左侧导航窗格中，选择管理订阅。
3. 找到您要启动的 AMI，选择启动新实例。
4. 在区域下，选择列为允许的区域。
5. 选择继续通过 EC2 启动。此操作将使您转入 Amazon EC2 控制台。
6. 输入服务器的名称。
7. 选择与您的项目性能和成本要求相匹配的实例类型。建议的实例大小的起点是 `c5.2xLarge`。
8. 选择现有密钥对或创建新的密钥对并保存。有关密钥对的信息，请参阅《适用于 Linux 实例的 Amazon EC2 用户指南》中的 Amazon EC2 密钥对和 Linux 实例。
9. 编辑网络设置并选择列为允许的 VPC 和相应的子网。
10. 选择现有安全组或者创建新的安全组。除了允许 SSH 访问（端口 22 默认允许）之外，对于使用 Precisely 服务器 EC2 实例进行的数据复制，通常还允许 TCP 流量进入其默认端口 2626。
11. 为 Amazon EC2 实例配置存储。
12. 审核摘要并选择启动实例。要成功启动，实例类型必须有效。如果启动失败，请选择编辑实例配置并选择其他实例类型。
13. 看到成功消息后，选择连接到实例。
14. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
15. 在左侧导航窗格中的实例菜单下，选择实例。
16. 在主窗格中，检查实例的状态。

创建 IAM 策略

要成功运行通过我们的 AWS Marketplace 列表部署 AWS 的大型机现代化 EC2 实例，您必须配置 IAM 角色和策略。这种专门定制的 IAM 设置不是可选的；它授权您的 Amazon EC2 实例与该服务进行交互。 AWS Marketplace IAM 角色和策略允许 AWS 大型机现代化准确记录使用数据，这对于精确计费至关重要。未能实现此配置可能会导致数据复制尝试失败和操作中断。

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧的导航窗格中，选择策略。
3. 如果这是您首次选择策略，则会显示欢迎访问托管式策略页面。选择开始使用。
4. 在页面的顶部，选择创建策略。
5. 在策略编辑器部分，选择 JSON 选项。
6. 输入以下 JSON 策略。

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

创建 IAM 角色

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择角色，然后选择创建角色。
3. 在可信实体类型部分中，选择 AWS 服务。
4. 在使用案例部分的服务或使用案例下，选择 Amazon EC2。
5. 选择下一步。
6. 在策略列表中，从按类型筛选下拉列表中选择客户托管，然后输入您创建的策略的名称。选中该策略名称旁边的复选框。
7. 选择下一步。

8. 输入角色的名称和（可选）描述。
9. 查看信任策略和权限，然后选择创建角色。

将 IAM 角色附加到 Amazon EC2 实例

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在导航窗格中，选择 Instances (实例)。
3. 选择您的 Amazon EC2 实例。
4. 从操作菜单上，选择安全，然后选择修改 IAM 角色。
5. 选择要附加到您的实例的角色，然后选择更新 IAM 角色。

Charon 集成

Charon-SSP 简介

1987 年，Sun Microsystems 发布了 SPARC V7 处理器，这是一款 32 位 RISC 处理器。随后，在 1990 年，又发布了 SPARC V8，对最初的 SPARC V7 进行了修订，其中最引人注目的是包括了硬件除法和乘法指令。SPARC V8 处理器成为了许多服务器和工作站的基础，例如 SPARCstation 5、10 和 20。继 SPARC V8 之后，1993 年，又推出了 64 位的 SPARC V9 处理器。它也成为了许多服务器和工作站的基础，例如 Enterprise 250 和 450。

由于硬件过时以及缺少备件或翻新部件，为这些基于 SPARC 的旧工作站和服务器开发的软件和系统变得更加难以维护。为了满足对某些 end-of-life 基于Sparc的系统的持续需求，Stromasys S.A. 开发了Charon-SSP系列的SPARC仿真器产品。以下产品是指定本机硬件 SPARC 系统的基于软件的虚拟机替代品。以下是仿真硬件系列的总体概述。

Charon-SSP/4M 仿真以下 SPARC 硬件：

- Sun-4m 系列（代表产品为 Sun SPARCstation 20）：最初是一个多处理器 Sun-4 变体，基于 SPARCServer 600MP 系列中引入的 MBus 处理器模块总线。后来，Sun-4m 架构还包括非 MBus 单处理器系统，例如采用 SPARC V8 架构处理器的 SPAR Cstation 5。从 SunOS 4.1.2 开始受支持，并且得到 Solaris 2.1 至 Solaris 9 的支持。在 Solaris 2.5.1 之后，SPARCServer 600MP 支持被废弃。

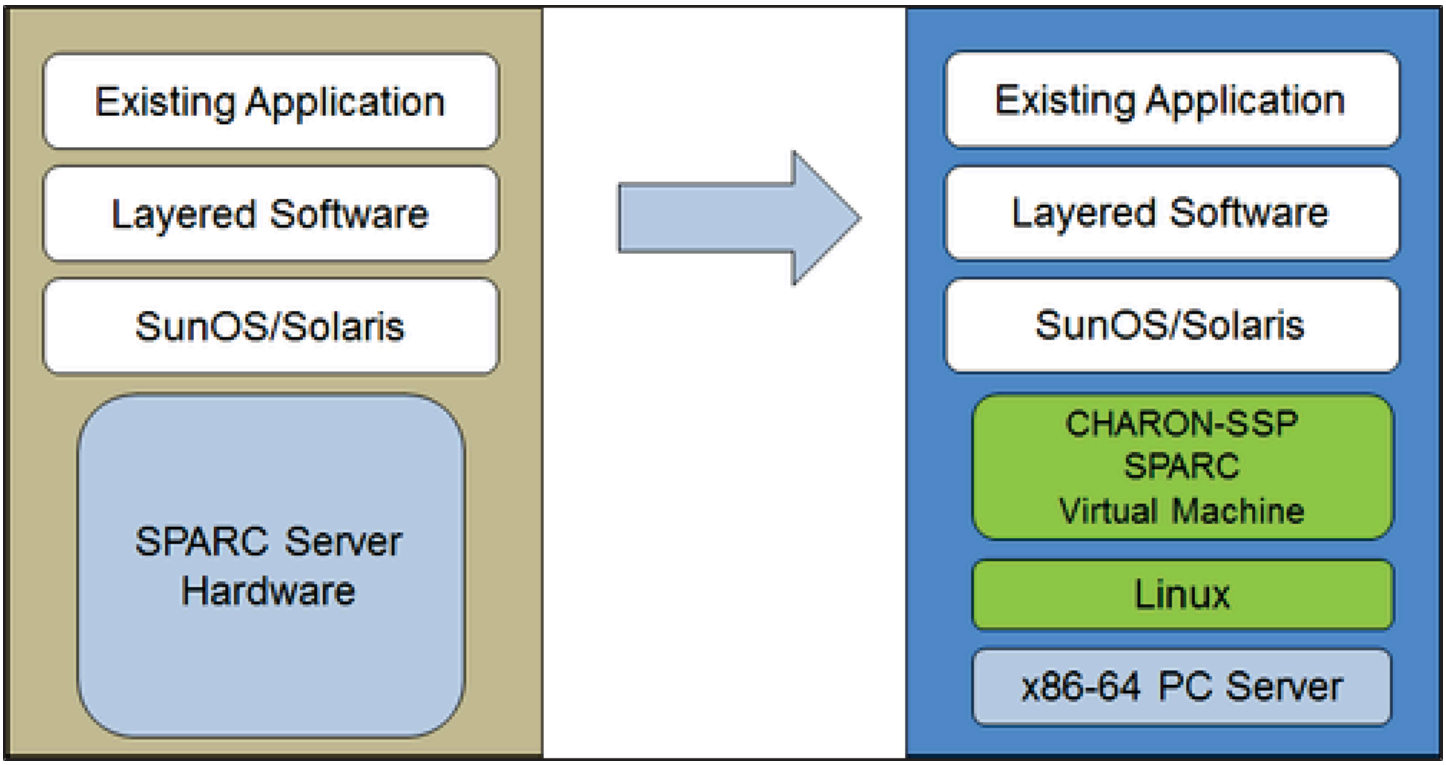
Charon-SSP/4U(+) 仿真以下 SPARC 硬件：

- Sun-4u 系列（代表产品为 Sun Enterprise 450）：（U 代表 UltraSPARC）- 该变体引入了 Sun Ultra 系列中首次使用的 64 位 SPARC V9 处理器架构和 UPA 处理器互连。得到自 2.5.1 版本开始的 32 位版本 Solaris 的支持。Sun-4u 的第一个 64 位 Solaris 版本是 Solaris 7。在 Solaris 9 之后，UltraSPARC I 支持被废弃。Solaris 10 支持从 UltraSP ARC II 到 UltraSPARC IV 的 Sun-4U 实现。

Charon-SSP/4V(+) 仿真以下 SPARC 硬件：

- Sun-4v 系列（代表产品为 SPARC T2 和 T4）：此变体在 Sun-4u 中增加了虚拟机管理程序处理器虚拟化；被引入到 Ultra SPARC T1 多核处理器中。从 3/05 HW2 版本开始，Solaris 版本 10 支持部分硬件（大多数型号，包括由 Charon-SSP 仿真的硬件，都需要更新版本的 Solaris 10）。还支持多个 Solaris 11 版本。

下图显示了将物理硬件迁移到仿真器的基本概念。



Charon-SSP 虚拟机允许使用基于 Sun 和 Oracle SPARC 的计算机的用户以一种简单的方式来更换其本机硬件，这种方式很少需要甚至不需要更改原始系统配置。这意味着您可以继续运行应用程序和数据，而无需切换或移植到其他平台。Charon-SSP 软件在商用 Intel 64 位系统上运行，确保持续保护您的投资。

Charon-SSP/4U+ 支持与 Charon-SSP/4U 相同的虚拟 SPARC 平台，Charon-SSP/4V+ 则支持与 Charon-SSP/4V 相同的平台。不过，4U+ 和 4V+ 版本在现代 CPU 中利用了 Intel 的 VTx/EPT 和 AMD 的 AMD-v/NPT 硬件辅助虚拟化技术，可提供更好的虚拟 CPU 性能。Charon-SSP/4U+ 和 Charon-SSP/4V+ 需要支持 VT-x/EPT 或 AMD-v/NPT 的 CPU，并且必须安装在专属主机系统上。不支持在虚拟机中（例如，在 VMware 上）运行这些产品变体。

> **Note**
>
> 如果您打算在云环境中运行 Charon-SSP/4U+ 或 4V+ ，请联系 Stromasys 或 Stromasys VAR 来讨论您的要求。

支持的客户机操作系统

Charon-SSP/4M 虚拟机支持以下来宾操作系统版本：

- SunOS 4.1.3 - 4.1.4
- Solaris 2.3 到 Solaris 9

Charon-SSP/4U(+) 虚拟机支持以下来宾操作系统版本：

- Solaris 2.5.1 到 Solaris 10

Charon-SSP/4V(+) 虚拟机支持以下来宾操作系统版本：

- Solaris 10（从 08/07 的更新 4 开始）和 Solaris 11.1 到 Solaris 11.4

对于 Charon-SSP/4V(+)，请注意以下几点：

- 对于仿真的 SPARC T4，受支持的 Solaris 10 版本为：Oracle Solaris 10 1/13、Oracle Solaris 10 8/11 和 Solaris 10 9/10，或者带有 Oracle Solaris 10 8/11 补丁集的 Solaris 10 10/09。
- 仿真的 SPARC T4 型号是在仿真器中运行 Solaris 11.4 的先决条件。
- 不支持 Solaris 内核区域。

Charon-SSP 云实例的先决条件

通过选择实例类型或形态，您可以选择将用于云中 Charon-SSP 主机实例的虚拟硬件。因此，实例类型或形态的选择决定了 Charon-SSP 虚拟主机硬件的硬件特性（例如，您的虚拟 Charon 主机系统将有多少 CPU 内核和多少内存）。

> **Note**
>
> 如果使用 Charon-SSP 市场映像来启动实例，将满足所有 Linux 主机操作系统要求。

最低硬件要求如下。

有关规格指南的要点：

- 以下的规格指南（特别是有关主机 CPU 内核数和主机内存量）显示了最低要求。必须审查每种部署情况，并根据需要调整实际的主机规格。例如，如果来宾应用程序产生高 I/O 负载，则必须增加可用于 I/O 的 CPU 内核数量。此外，具有许多仿真 CPU 的系统通常能够产生更高的 I/O 负载，因此可

能必须增加可用于 I/O 的 CPU 内核数量。在超线程环境中，为了获得最佳性能，CPU 内核（即实际/物理 CPU）的数量必须足以满足活动仿真器的 CPU 要求，从而避免高工作负载线程共享一个物理 CPU 内核。

- 仿真 CPU 的 CPU 内核分配以及用于 I/O 处理的 CPU 内核由配置决定。关于这一点以及用于 I/O 处理的 CPU 内核的默认分配的更多信息，请参阅一般性的《Charon-SSP 用户指南》中的 CPU 配置。

⚠ 重要的一般信息

- 为了便于将仿真器数据从一个云实例快速传输到另一个云实例，强烈建议您将所有相关的仿真器数据存储在单独的磁盘卷上，而该磁盘卷可以轻松地与旧实例分离并连接到新实例。
- 确保从一开始就形成正确的实例大小（检查下面的最低要求）。首次启动实例时，将创建 Charon-SSP AL 的 Charon-SSP 许可证。如果稍后改为其他实例大小/类型，进而改变 CPU 内核的数量，就会使许可证失效，阻止 Charon 实例启动（需要新实例）。如果计划在 AutoVE 模式下使用 Charon-SSP AL 实例，请务必在首次启动之前包含 AutoVE 服务器信息，否则将使用公共许可证服务器。Charon-SSP VE 的许可证是根据在许可证服务器上获取的指纹创建的。如果许可证服务器直接在仿真器主机上运行，而仿真器主机后来要求更改 CPU 内核数量，则该许可证将失效（需要新的许可证，还可能需要新的实例）。

实例先决条件

一般 CPU 要求：Charon-SSP 支持基于现代 x86-64 架构处理器的 Amazon EC2 实例。

Charon-SSP 的最低要求：

- 主机系统 CPU 内核的最小数量：
  - 主机操作系统至少有一个 CPU 内核，以及：
  - 对于每个仿真的 SPARC 系统：
    - 实例的每个仿真 CPU 有一个 CPU 内核，以及：
    - 至少增加一个用于 I/O 处理的 CPU 内核（如果使用服务器 JIT 优化，则至少需要两个 CPU 内核）。有关配置选项，请参阅上面提到的 CPU 配置部分。默认情况下，Charon 会将 Charon 主机可见的 1/3 的 CPU（最少 1 个；向下舍入）分配给 I/O 处理。
- 最低内存要求：

  - 对于 Linux 主机操作系统，4GB 或更大 RAM。实际要求可能更高，并将取决于在 Linux 主机上运行的非仿真器服务的要求。之前关于 Linux 主机至少具有 2GB RAM 的建议对许多系统仍然有效，但是随着对 Linux 操作系统和应用程序的要求不断提高，对于新安装的建议也更新了。以及：
  - 对于每个仿真的 SPARC 系统：
    - 仿真实例的配置内存，以及：
    - 2GB 内存（如果使用服务器 JIT，则为 6GB 内存），用于实现 DIT 优化、仿真器要求、运行时缓冲区、SMP 和图形仿真。
- 如果在现代 x86-64 CPU 上启用了超线程，则两个线程可以在一个物理 CPU 内核上运行，为主机操作系统提供两个逻辑 CPU。如果可能，请在 Charon-SSP 主机上禁用超线程。但是，在 VMware 和云环境中，这通常是不可能的，或者不清楚是否使用了超线程。Charon-SSP 超线程选项使 Charon-SSP 能够适应此类环境。有关详细的配置信息，请参阅上面提到的一般性《Charon-SSP 用户指南》中的 CPU 配置部分。请注意：为了获得最佳性能，Charon-SSP 线程不应共享物理 CPU 内核 - 主机系统上应有足够的物理核心，以便满足已配置仿真器的要求。
- 一个或多个网络接口，视客户要求而定。
- Charon-SSP/4U+ 和 Charon-SSP/4V+ 必须在支持 Intel VT-x/EPT 或 AMD-v/NPT（裸机实例）的物理硬件上运行，因此无法在所有云环境中运行。请查看您的云提供商的文档，了解此类硬件的可用性。此外，请注意以下几点：
  - Charon-SSP/4U+ 和 Charon-SSP/4V+ 只有在使用 Stromasys 支持的 Linux 内核时才可用。
  - 如果您需要这种类型的仿真 SPARC 硬件，请联系 Stromasys 或您的 Stromasys VAR，详细讨论您的需求。

为 Charon 创建和配置 AWS 云实例（新 GUI）

本节反映了202 AWS Management Console 2年春季的情况。如果您仍在使用较旧的主机，请参阅 Charon-SSP AWS 入门指南的附录。

一般先决条

此描述显示了 AWS中的 Linux 实例的基本设置。它没有列出具体的先决条件。但是，根据您的使用案例，可考虑以下先决条件：

- 亚马逊账户和 AWS Marketplace 订阅
  - 要在中设置 Linux 实例 AWS，您需要一个具有管理员访问权限的 AWS 帐户。
  - 确定您计划在哪个 AWS 地区启动您的实例。确保您计划使用的 AWS 服务在该区域可用。请参阅按区域划分的AWS 服务。

- 确定您计划在其中启动您的实例的 VPC 和子网。
- 如果您的实例需要互联网访问，请确保与您的 VPC 关联的路由表具有互联网网关。如果您的实例需要对本地网络进行 VPN 访问，请确保 VPN 网关可用。VPC 及其子网的确切配置将取决于您的网络设计和应用要求。
- 要订阅特定 AWS Marketplace 服务，请在中选择 AWS Marketplace 订阅， AWS Management Console 然后选择管理订阅。
- 搜索您计划使用的服务并订阅它。成功订阅后，您将在管理订阅部分中找到该订阅。从那里您可以直接启动一个新的实例。

- 根据实例的计划用途，实例的硬件和软件先决条件会有所不同：
  - 选项 1：实例将用作 Charon 仿真器主机系统：
    - 请参阅 Charon 产品的《用户指南》和/或《入门》指南中的硬件和软件先决条件部分，以确定 Linux 实例必须满足的确切硬件和软件先决条件。您用于启动实例的映像和您选择的实例类型决定了您的云实例的软件和硬件。
    - 运行仿真旧系统需要 Charon 产品许可证。请参阅 Charon 产品文档中的许可信息，或者联系您的 Stromasys 代表或 Stromasys VAR 了解更多信息。
  - 选项 2：实例将用作专用 VE 许可证服务器：
    - 有关详细的先决条件，请参阅《VE 许可证服务器指南》。
- 某些可在 Charon 仿真器产品提供的仿真系统中运行的传统操作系统需要操作系统原始供应商的许可。用户应对与旧版操作系统相关的任何许可义务负责，并且必须提供相应的许可证。

使用 AWS Management Console 启动新实例

创建新实例

1. 登录 AWS Management Console 并打开亚马逊 EC2 控制台，网址为 https://console.aws.amazon.com/ec2/。
2. 选择启动实例。
3. 为实例输入名称。
4. 选择 AMI。AMI 是用于启动云实例的预打包映像。它包括操作系统和适用的应用程序软件。如何选择 AMI 取决于您计划如何使用实例：
   - 如果要将实例用作 Charon 仿真器主机系统，则可以有多个 AMI 选择：
     - 从预打包的 Charon 市场映像安装 Charon 主机系统：它们包含底层操作系统和预装的 Charon 软件。

    - 请咨询您的 Stromasys 代表，了解您的云提供商市场中目前有哪些选项可用。
    - 根据云提供商和 Stromasys 产品发布计划，可能有两种变体：
        - 自动许可（AL）可用于 Stromasys 运营的公共许可证服务器，或用于客户运营的私有 AutoVE 许可证服务器
        - 虚拟环境（VE）可用于客户运营的私有 VE 许可证服务器
    - 使用传统的 Charon 仿真器安装（包含适用于 Linux 的 Charon 仿真器安装 RPM 软件包）来安装 Charon 主机系统：
        - 选择由您所选的 Charon 产品和版本提供支持的发行版的 Linux AMI。请参阅 Stromasys 文档网站上的产品用户指南。
- 如果要将实例用作专用 VE 许可证服务器，请参阅许可文档中的《VE 许可证服务器指南》，了解 Linux 实例的要求。

在决定需要哪个 AMI 之后，请选择匹配的 Linux 或 Charon 产品 AMI。如果您没有看到所需的 AMI，请选择浏览更多 AMI。选择与您计划使用实例的方式相匹配的 Linux AMI。它可以是下列项之一：

- 预打包的 Charon VE 市场映像。AMI 的名称将包含字符串“ve”。
- 自动许可或 AutoVE 的预打包的 Charon AL 市场映像。
- RPM 产品安装支持的 Linux 版本。
- VE 许可证服务器支持的 Linux 版本。

5. 选择一个实例类型。Amazon EC2 提供具有不同的 CPU、内存、存储和网络容量组合的实例类型。选择符合您要使用的 Charon 产品要求的实例类型。一些市场映像的实例类型选择有限。
6. 选择现有密钥对或创建新密钥对并保存。如果您选择现有密钥对，请确保您拥有匹配的私钥。否则，您将无法连接到您的实例。

> **Note**
>
> 如果您的管理系统支持它，则对于 RHEL 9.x、Rocky Linux 9.x 和 Oracle Linux 9.x，使用 SSH 密钥类型 ECDSA 或 ED25519。这些类型允许您使用 SSH 隧道连接到这些 Charon 主机 Linux 系统，而无需将 Charon 主机上的默认加密策略设置更改为不太安全的设置。例如，这对于 Charon-SSP Manager 很重要。请参阅 Red Hat 文档中的使用系统级加密策略。

7. 在网络设置部分中，选择编辑。选择与您的环境相对应的设置。

- 指定 VPC。
- 指定现有子网或创建新的子网。
- 启用或禁用为主接口自动分配公有 IP 地址的功能。只有当实例只有一个网络接口时，才能自动分配。
- 分配现有或新的自定义安全组。安全组必须至少允许 SSH 访问实例。还必须允许您计划在实例上运行的应用程序所需的任何端口。创建实例后，您可以随时修改安全组。

8. 在存储部分，为根卷（系统磁盘）选择适合您的环境的大小。建议 Linux 系统的最小系统磁盘大小为 30 GiB。要为虚拟磁盘容器和其他存储需求提供空间，您可以立即或在启动实例之后添加更多存储。但是系统磁盘大小必须满足 Linux 系统要求，包括您计划安装的所有应用程序和实用程序。

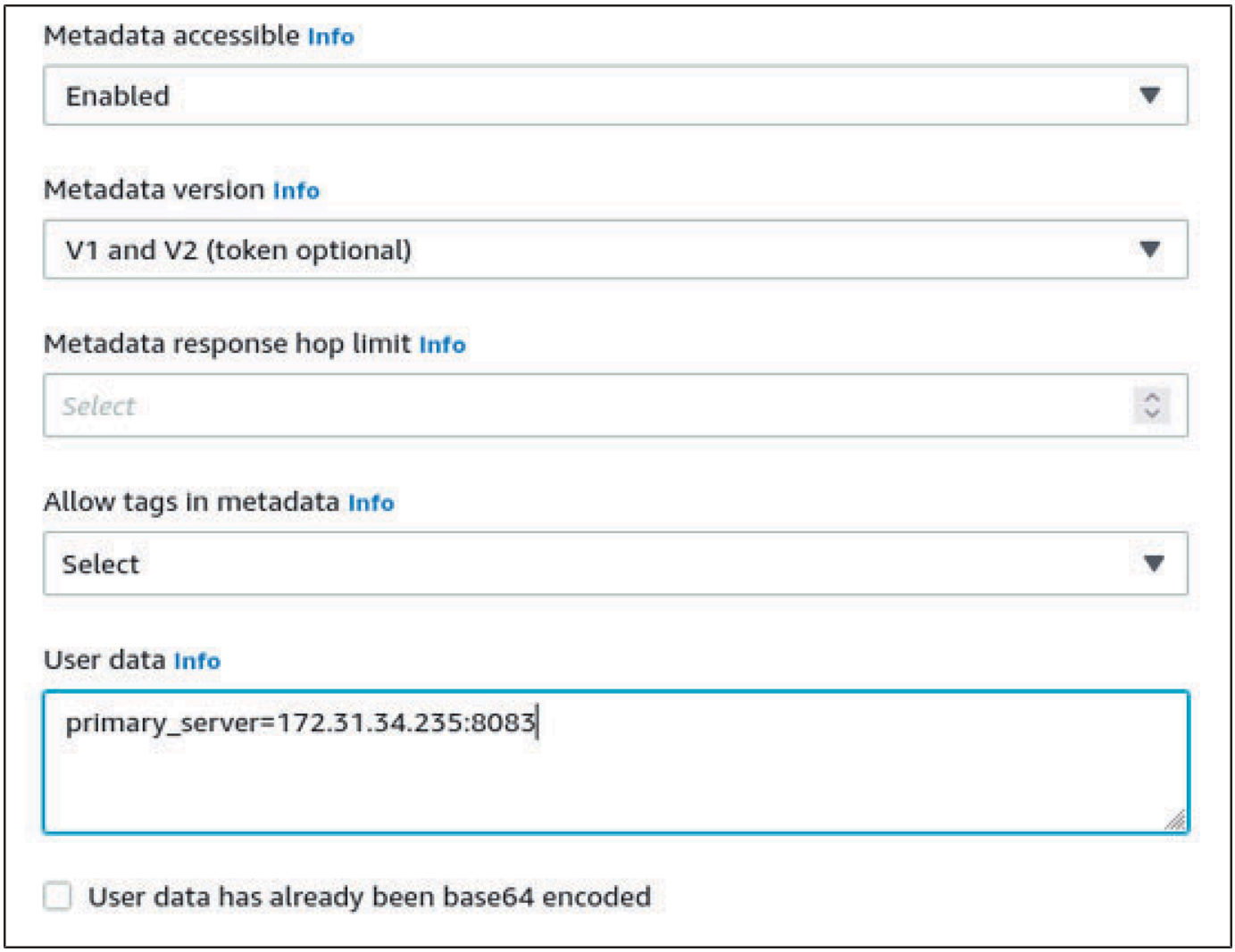
> **Note**
>
> 建议您为 Charon 应用程序数据（例如磁盘映像）创建单独的存储卷。如有必要，可以稍后将此类卷迁移到另一个实例。

9. 展开高级详细信息部分，向下滚动并选择指定 CPU 选项。作为示例，下图显示了三个更可能对 Charon 仿真器环境有用的选项。

10. 对于版本低于 1.1.23 的 VE 许可证服务器系统，您必须为实例分配所需的 IAM 角色。它必须是一个允许执行 `ListUsers` 操作的角色。要分配角色，请在展开的高级详细信息部分，在 IAM 实例配置文件下选择一个角色，或者选择创建新的 IAM 配置文件。有关更多信息，请参阅 Amazon EC2 的 IAM 角色。

11. 如果您的实例基于 Charon AL AWS Marketplace 映像，并且您计划使用 Stromasys 运营的公共许可证服务器，则必须在启动实例之前将相应信息添加到实例配置中。

    输入 AutoVE 许可证服务器的信息，如下图中所示。



    以下是有效的用户数据配置选项：

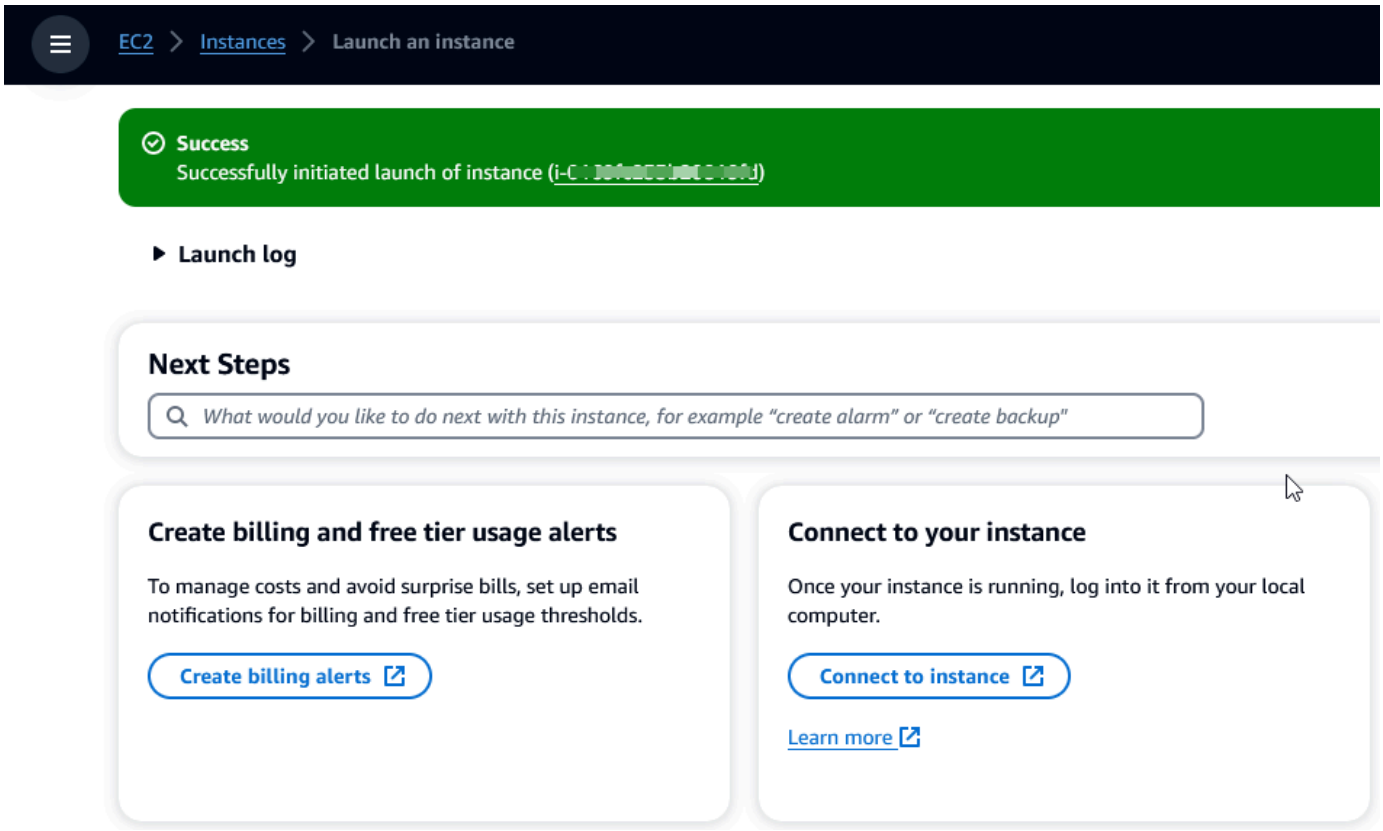
    - **primary_server=***<ip-address>*[**:***<port>*]
    - **backup_server=***<ip-address>*[**:***<port>*]

    位置

    - <ip-address> 代表主服务器和备份服务器（如适用）的 IP 地址。
    - <port> 代表用于与许可证服务器通信的非默认 TCP 端口（默认：TCP/8083）。

> **Note**
>
> 初次启动时必须至少配置一个许可证服务器才能启用 AutoVE 模式。否则，该实例将绑定到 Stromasys 运营的公共许可证服务器之一。

12. 在摘要部分，选择启动实例。过一会儿，您将看到以下成功消息：



13. 在屏幕的右下角，选择查看所有实例。
14. 要查看您的实例的详细信息，请在实例表中选中代表该实例的行左侧的复选框。您的实例详细信息将显示在屏幕的下半部分。有关如何连接您的实例的信息，请参阅《适用于 Linux 实例的 Amazon EC2 用户指南》中的连接。

AWS 使用 NTT DATA 实现大型机现代化改造平台

AWS 大型机现代化提供各种亚马逊系统映像 (AMI)。这些 AMI 有助于快速配置 Amazon EC2 实例，为使用 NTT 数据重新托管和重新构建大型机应用程序创建量身定制的 AWS 环境。本指南提供访问和使用这些 AMI 所需的步骤。

先决条件

- 确保您拥有可以创建 Amazon EC2 实例的 AWS 账户的管理员访问权限。
- 确认 AWS 大型机现代化服务在您计划创建 Amazon EC2 实例的地区可用。查看按区域划分的可用 AWS 服务列表。
- 确定您要在其中创建 Amazon EC2 实例的 Amazon VPC。

订阅亚马逊机器映像

当您订阅 AWS Marketplace 产品后，可以从该产品的 AMI 启动实例。

1. 登录 AWS Management Console 并打开 AWS Marketplace 控制台，网址为 https://console.aws.amazon.com/marketplace。
2. 选择管理订阅。
3. 将以下链接复制并粘贴到浏览器地址栏中：https://aws.amazon.com/marketplace/pp/prodview-eg227ymldsnx2
4. 选择继续订阅。
5. 如果可以接受条款和条件，请选择接受条款。处理订阅可能需要几分钟时间。
6. 等待感谢消息出现。该消息确认您是否成功订阅了产品。
7. 在左侧导航窗格中，选择管理订阅。此视图显示您的所有订阅。

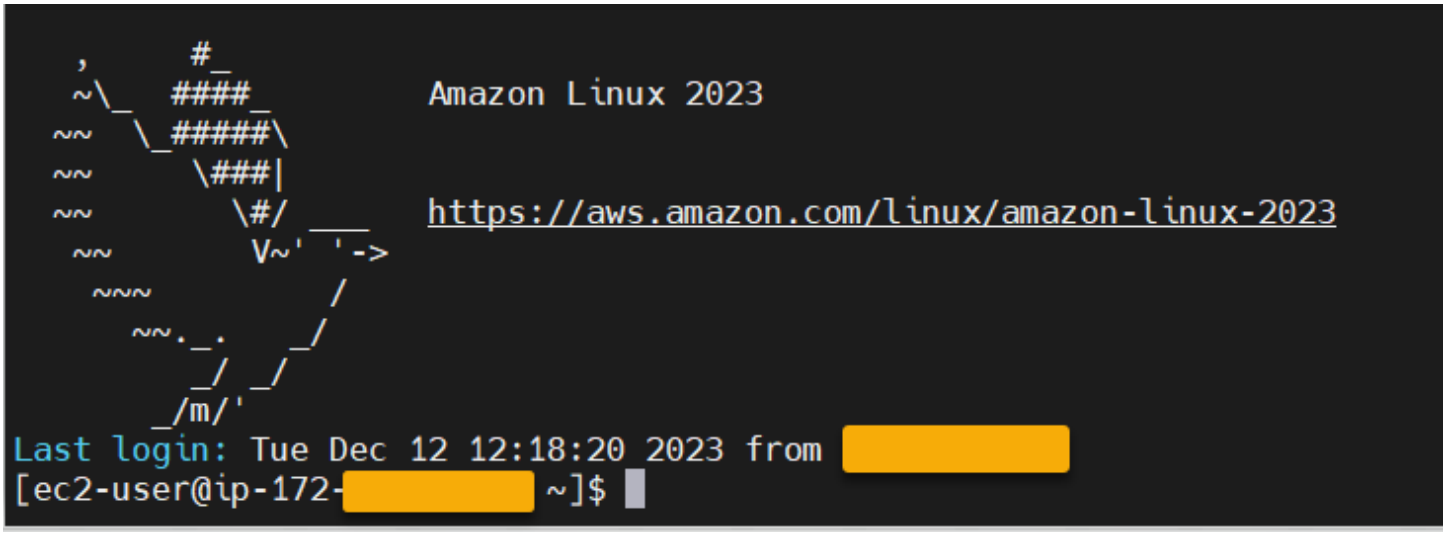
使用 NTT DATA 实例启动 AWS 大型机现代化改造平台

1. 打开 AWS Marketplace 控制台，网址为 https://console.aws.amazon.com/marketplace。
2. 在左侧导航窗格中，选择管理订阅。
3. 找到您要启动的 AMI，选择启动新实例。

4. 在区域下，选择列为允许的区域。
5. 选择继续通过 EC2 启动。此操作将使您转入 Amazon EC2 控制台。
6. 输入服务器的名称。
7. 选择与您的项目性能和成本要求相匹配的实例类型。建议的实例大小的起点是 `c5.2xLarge`。
8. 选择现有密钥对或创建新的密钥对并保存。有关密钥对的信息，请参阅《适用于 Linux 实例的 Amazon EC2 用户指南》中的 Amazon EC2 密钥对和 Linux 实例。
9. 编辑网络设置并选择列为允许的 VPC 和相应的子网。
10. 选择现有安全组或者创建新的安全组。如果这是 Enterprise Server Amazon EC2 实例，通常它会允许 TCP 流量进入端口 86 和 10086，以管理 Micro Focus 配置。
11. 为 Amazon EC2 实例配置存储。
12. 审核摘要并选择启动实例。要成功启动，实例类型必须有效。如果启动失败，请选择编辑实例配置并选择其他实例类型。
13. 看到成功消息后，选择连接到实例。
14. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
15. 在左侧导航窗格中的实例菜单下，选择实例。
16. 在主窗格中，检查实例的状态。

开始使用 NTT Data

预置好 Amazon EC2 实例后，使用用户名 `ec2-user` 通过 SSH 进入该实例。屏幕显示与下图类似。



在 `/opt/software/` 文件夹下，有一个名为 `UniKix_Product_Guides` 的文件夹，如下图所示。

```
[ec2-user@ip-172-          ~]$ ls -l /opt/software/
total 64
lrwxrwxrwx.  1 root     root        23 Oct 17 19:27 BPE -> /opt/software/BPE17.2.3
drwxr-xr-x.  6 ec2-user ec2-user 16384 Oct  4 16:38 BPE17.2.3
lrwxrwxrwx.  1 ec2-user ec2-user    32 Oct 17 19:27 COBOL -> /opt/software/NTT_DATA_COBOL_6.5
drwxr-xr-x. 11 ec2-user ec2-user 16384 Oct 17 19:27 NTT_DATA_COBOL_6.5
lrwxrwxrwx.  1 ec2-user ec2-user    36 Oct 17 19:28 NTT_DATA_TPE_Agent -> /opt/software/NTT_DATA_TPE_Agent_4.9
drwxr-xr-x.  8 ec2-user ec2-user 16384 Nov  9 01:59 NTT_DATA_TPE_Agent_4.9
lrwxrwxrwx.  1 ec2-user ec2-user    25 Oct 17 19:28 Secure -> /opt/software/Secure6.3.1
drwxr-xr-x.  8 ec2-user ec2-user   156 Oct 17 19:28 Secure6.3.1
lrwxrwxrwx.  1 ec2-user ec2-user    23 Oct 17 19:27 TPE -> /opt/software/TPE17.2.2
drwxr-xr-x. 12 ec2-user ec2-user 16384 Oct  4 16:34 TPE17.2.2
lrwxrwxrwx.  1 ec2-user ec2-user    20 Oct 17 19:28 UCM -> /opt/software/UCM2.1
drwxr-xr-x.  7 ec2-user ec2-user   173 Oct 17 19:28 UCM2.1
drwxr-xr-x.  2 ec2-user ec2-user     6 Dec 12 12:20 UniKix_Product_Guides
drwxr-xr-x.  2 ec2-user ec2-user     6 Oct 17 19:22 bin
drwxr-xr-x.  2 ec2-user ec2-user    34 Nov 10 17:03 license
drwxr-xr-x.  8 root     root        88 Oct 17 19:28 staging
```

`UniKix_Product_Guides` 文件夹包含安装在此 Amazon EC2 实例上的以下组件的文档：

- NTT DATA TPE
- NTT DATA BPE
- NTT DATA Enterprise COBOL
- NTT 数据安全 UniKix
- NTT 数据 UniKix 中心管理器

上图中显示的 `software` 文件夹包含上面列出的组件的二进制文件。

成功验证 Amazon EC2 实例后，请按照 NTT 数据文档开始使用带有 NTT 数据 AWS 的大型机现代化改造平台。

AWS Mainframe Modernization 中的应用程序

如果您是首次接触 AWS Mainframe Modernization，请参阅以下主题了解其用法：

- 什么是 AWS 大型机现代化？
- 设置 AWS 大型机现代化
- 教程：AWS Blu Age 的托管运行时
- 教程：Micro Focus 的托管运行时

AWS Mainframe Modernization 中的应用程序包含已迁移的大型机工作负载。应用程序类似于大型机上的工作负载，并且与运行时环境相关联。您可以向应用程序添加批处理文件和数据集，并在应用程序运行时对其进行监控。您可以为迁移的每个工作负载创建 AWS Mainframe Modernization 应用程序。创建 AWS Mainframe Modernization 应用程序时，需要指定运行该应用程序的引擎。如果您使用的是自动重构模式，请选择 AWS Blu Age；如果您使用的是更换平台模式，请选择 Micro Focus。

主题

- 创建 AWS Mainframe Modernization 应用程序
- 部署 AWS Mainframe Modernization 应用程序
- 更新 AWS Mainframe Modernization 应用程序
- 从环境中删除 AWS Mainframe Modernization 应用程序
- 删除 AWS Mainframe Modernization 应用程序
- 为 AWS Mainframe Modernization 应用程序提交批处理作业
- 为 AWS Mainframe Modernization 应用程序导入数据集
- 管理 AWS Mainframe Modernization 应用程序的事务
- 为迁移的应用程序创建 AWS 资源
- 配置托管应用程序
- AWS 大型机现代化应用程序定义参考
- AWS Mainframe Modernization 数据集定义参考

创建 AWS Mainframe Modernization 应用程序

使用 AWS Mainframe Modernization 控制台创建 AWS Mainframe Modernization 应用程序。

以下说明假定您已完成设置 AWS 大型机现代化 中的步骤。

创建应用程序

创建应用程序

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要在其中创建应用程序的区域。
3. 在应用程序页面上，选择创建应用程序。
4. 在指定基本信息页面的名称和描述部分，输入应用程序的名称。
5. （可选）在应用程序描述字段中，输入应用程序的描述。该描述有助于您及其他用户了解该应用程序的用途。
6. 在引擎类型部分，选择 Blu Age 进行自动重构，或选择 Micro Focus 来更换平台。
7. 如果要使用客户托管的 AWS KMS 密钥，在 KMS 密钥部分，选择自定义加密设置。有关更多信息，请参阅AWS 大型机现代化服务的静态数据加密。

   > **Note**
   >
   > 默认情况下，AWS Mainframe Modernization 使用 AWS Mainframe Modernization 为您保留和管理的 AWS KMS 密钥来加密您的数据。不过，您可以选择使用客户托管的 AWS KMS 密钥。

8. （可选）按名称或 Amazon 资源名称 (ARN) 选择 AWS KMS 密钥，或者选择创建 AWS KMS 密钥进入 AWS KMS 控制台并创建新的 AWS KMS 密钥。
9. （可选）在标签部分中，选择添加新标签以向应用程序添加一个或多个应用程序标签。应用程序标签是一种自定义属性标签，可帮助您组织和管理 AWS 资源。
10. 选择下一步。
11. 在资源和配置部分，使用内联编辑器输入应用程序定义。或者，选择在 Amazon S3 存储桶中使用应用程序定义 JSON 文件并提供要使用的应用程序定义的位置。有关更多信息，请参阅 AWS Blu Age 应用程序定义示例或 Micro Focus 应用程序定义。
12. 选择下一步。
13. 在审核和创建页面中，检查您输入的信息，然后选择创建应用程序。

部署 AWS Mainframe Modernization 应用程序

使用 AWS Mainframe Modernization 控制台部署 AWS Mainframe Modernization 应用程序。

以下说明假定您已完成设置 AWS 大型机现代化 中的步骤。

部署应用程序

要运行 AWS Mainframe Modernization 应用程序，必须先将其部署到运行时环境中。应用程序可以包含多个版本。应用程序的每个版本都有各自的应用程序定义。要部署应用程序，必须指定要部署的版本。

一次只能部署给定应用程序的一个版本。如果您部署应用程序的某个版本，然后决定改为部署其他版本，则必须先停止该应用程序（如果该应用程序正在运行）。

部署应用程序

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要在其中创建应用程序的区域。
3. 在应用程序页面上，选择要部署的应用程序。
4. 选择部署应用程序。
5. 在可用版本部分中，选择要部署的版本。
6. 在环境部分中，选择要在其中运行应用程序的运行时环境。
7. 选择部署。

部署已部署应用程序的不同版本

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要在其中创建应用程序的区域。
3. 在应用程序页面上，选择要部署的应用程序。
4. 从操作菜单中选择停止应用程序。
5. 应用程序停止后，选择部署应用程序。
6. 在可用版本部分中，选择要部署的版本。环境部分中已预先选择已部署该应用程序的环境。
7. 选择部署。

更新 AWS Mainframe Modernization 应用程序

使用 AWS Mainframe Modernization 控制台更新 AWS Mainframe Modernization 应用程序。

以下说明假定您已完成设置 AWS 大型机现代化 中的步骤。

更新应用程序

一个 AWS Mainframe Modernization 应用程序可以有多个版本，每个版本都有自己的应用程序定义。要更新应用程序，请提供新的应用程序定义。这样可以创建应用程序的新版本。

更新应用程序

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域 选择器中，选择要更新的应用程序创建时所选的区域。
3. 在应用程序页面上，选择要更新的应用程序。
4. 在应用程序详细信息页面的当前定义部分，选择编辑以更新当前的应用程序定义。
5. 在更新应用程序页面上，使用内联编辑器更新当前的应用程序定义。

   或者，选择在 Amazon S3 存储桶中使用应用程序定义 JSON 文件并提供要使用的应用程序定义的位置。有关更多信息，请参阅 AWS Blu Age 应用程序定义示例或 Micro Focus 应用程序定义。
6. 更新完应用程序定义后，选择更新。

> **Note**
>
> 更新应用程序后，必须将其重新部署。有关更多信息，请参阅部署 AWS Mainframe Modernization 应用程序。

从环境中删除 AWS Mainframe Modernization 应用程序

您可以使用 AWS Mainframe Modernization 控制台从环境中删除 AWS Mainframe Modernization 应用程序。

以下说明假定您已完成设置 AWS 大型机现代化 中的步骤。

从环境中删除应用程序

如果您需要删除正在运行的 AWS Mainframe Modernization 应用程序，请务必先将其停止。您可以在应用程序页面上查看应用程序的状态。

从环境中删除应用程序

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要从环境中删除的应用程序创建时所选的区域。
3. 在应用程序页面上，选择要从环境中删除的应用程序，然后选择操作。
4. （可选）如果应用程序的状态为 `Running`，选择停止应用程序。
5. 选择从环境中删除。

   删除过程会立即开始。

删除 AWS Mainframe Modernization 应用程序

使用 AWS Mainframe Modernization 控制台删除 AWS Mainframe Modernization 应用程序。

以下说明假定您已完成设置 AWS 大型机现代化 中的步骤。

删除应用程序

如果您需要删除正在运行的 AWS Mainframe Modernization 应用程序，请务必先将其停止。您可以在应用程序页面上查看应用程序的状态。

删除应用程序

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要删除的应用程序创建时所选的区域。
3. 在应用程序页面上，选择要删除的应用程序，然后选择操作。
4. （可选）如果应用程序的状态为 `Running`，选择停止应用程序。
5. 选择删除应用程序。
6. 在删除应用程序窗口中，输入 `delete` 以确认要删除该应用程序，然后选择删除。

为 AWS Mainframe Modernization 应用程序提交批处理作业

您可以在 AWS Mainframe Modernization 中为应用程序提交批处理作业。您可以提交或取消批处理作业，并查看有关批处理作业执行的详细信息。每次提交批处理作业时，AWS Mainframe Modernization 都会创建一个单独的批处理作业执行。您可以监控此作业的执行情况，也可以按名称搜索批处理作业，并向批处理作业提供 JCL 或脚本文件。

> **Important**
>
> 取消批处理作业不会删除该作业，而是会取消批处理作业的特定执行。您仍然可以使用批处理作业记录查看批处理作业执行的详细信息。

如果您的批处理作业需要访问一个或多个数据集，请使用 AWS Mainframe Modernization 控制台或 AWS Command Line Interface (AWS CLI) 导入数据集。有关更多信息，请参阅为 AWS Mainframe Modernization 应用程序导入数据集。

以下说明假定您已完成设置 AWS 大型机现代化 和创建 AWS Mainframe Modernization 应用程序中的步骤。

提交批处理作业

提交批处理作业

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要为其提交批处理作业的应用程序创建时所选的区域。
3. 在应用程序页面上，选择要为其提交批处理作业的应用程序。

   > **Note**
   >
   > 在向应用程序提交批处理作业之前，必须成功部署该应用程序。

4. 在应用程序详细信息页面上，选择批处理作业。
5. 选择提交作业。
6. 在选择脚本部分，选择一个脚本。您可以按名称搜索所需的脚本。
7. 选择提交作业。

为 AWS Mainframe Modernization 应用程序导入数据集

您可以使用 AWS Mainframe Modernization 导入数据集以用于您的应用程序。您可以指定 Amazon S3 存储桶中存储的 JSON 文件中的数据集，或者也可以单独指定数据集配置值。导入数据集后，可以查看导入任务的详细信息，以确认所需的数据集已导入。控制台中会一并列出应用程序的所有已编目数据集。

使用 AWS Mainframe Modernization 控制台为 AWS Mainframe Modernization 应用程序导入数据集。

以下说明假定您已完成设置 AWS 大型机现代化 和创建 AWS Mainframe Modernization 应用程序中的步骤。

导入数据集

导入数据集

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要为其导入数据集的应用程序创建时所选的区域。
3. 在应用程序页面上，选择要为其导入数据集的应用程序。
4. 在应用程序详细信息页面上，选择数据集。
5. 选择导入。
6. 请执行以下操作之一：
   - 选择使用 Amazon S3 存储桶中的数据集配置 JSON 文件，并提供数据集配置的位置。
   - 选择单独指定数据集配置值，并按引导配置操作。有关特定定义的详细信息，请参阅 the section called “数据集定义参考”。

     输入每个数据集配置值的名称、数据集组织（VSAM、GDG、PO、PS）、位置和外部 Amazon S3 位置以及参数设置。在引导配置中，您还可以选择生成 JSON 以查看您输入的 JSON 配置。
7. 选择提交。

管理 AWS Mainframe Modernization 应用程序的事务

借助 AWS Mainframe Modernization，您可以根据请求与许多其他用户（使用相同的文件和程序提交了运行相同应用程序的请求）同时运行应用程序。单个事务由一个或多个执行所需处理的应用程序组成。

以下说明假定您已完成设置 AWS 大型机现代化 和创建 AWS Mainframe Modernization 应用程序中的步骤。

管理应用程序事务

您可以显示和编辑应用程序事务。

管理应用程序事务

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要运行的应用程序创建时所选的 AWS 区域。
3. 在应用程序页面上，选择要为其管理事务的应用程序。
4. 在事务选项卡的事务资源下，从下拉列表中选择资源的显示方式。您可以根据事务资源、组、列表或 SIT 显示资源。
   - 事务资源允许您根据文件定义、事务定义、程序定义或临时数据队列定义来选择资源类型。

     > **Note**
     >
     > AWS Mainframe Modernization 托管事务支持的资源类型不止这些，但您无法在此处直接对其进行编辑。其他资源必须在外部编辑，并且需要使用更新的资源条目重新创建应用程序。

   - 组是事务资源的集合。您也可以选择要与事务资源关联的组。
   - 列表是按顺序排列的组集合。您可以在列表视图中查看所有事务资源和组。启动列表决定在服务器初始化时加载哪些资源。
     - 使用 Blu Age 重构引擎时，您可以在启动时指定要包含的列表，并且列表数量没有限制。
     - 使用 Micro Focus 更换平台引擎时，您最多可以在一个 SIT 中指定四个列表。
   - SIT（系统初始化表）显示所有可用的事务配置。您可以根据属性（名称、描述和启动列表）查找 SIT，也可以选择列表来与您所选的 SIT 关联。

     > **Note**
     >
     > SIT 仅适用于 Micro Focus 更换平台引擎。

5. 选择事务资源以显示所有资源信息。您还可以查看与您的事务资源关联的所有属性，并查看或编辑任何其他属性。

6. 如果要编辑与当前事务资源相关的任何信息，请选择编辑。

   a. 在编辑页面上，您可以添加或更改资源描述。

      1. 对于列表中显示的事务资源，您也可以根据需要对列表进行添加、删除或重新排序。
      2. 对于特定的资源类型（文件定义、事务定义、程序定义和临时数据队列定义），您可以编辑各个资源属性。这些属性因引擎和资源类型而异。

   b. 进行所需更改后，选择保存更改。

      成功更新资源后，您可以看到一条消息。

为迁移的应用程序创建 AWS 资源

为了在 AWS 中运行已迁移的应用程序，必须使用其他 AWS 服务创建一些 AWS 资源。必须创建的资源包括以下各项：

- 一个 S3 存储桶，用于存放应用程序代码、配置、数据文件和其他必需的构件。
- 一个 Amazon RDS 或 Amazon Aurora 数据库，用于保存应用程序所需的数据。
- 一个 AWS KMS key，这是 AWS Secrets Manager 创建和存储密钥所必需的。
- 一个 Secrets Manager 密钥，用于存放数据库凭证

> **Note**
>
> 每个迁移的应用程序均需要自己的一组上述资源。上述资源组是最低要求。您的应用程序可能还需要额外的资源，例如 Amazon Cognito 密钥或 MQ 队列。

所需的权限

请确保您具有以下权限：

- `s3:CreateBucket`, `s3:PutObject`
- `rds:CreateDBInstance`
- `kms:CreateKey`
- `secretsmanager:CreateSecret`

Amazon S3 存储桶

重构后的应用程序和更换平台的应用程序均需要一个按如下方式配置的 S3 存储桶：

```
bucket-name/root-folder-name/application-name
```

bucket-name

符合 Amazon S3 命名约束条件的任何名称。我们建议您将 AWS 区域名称作为存储桶名称的一部分。请确保在计划部署已迁移应用程序的同一区域中创建存储桶。

root-folder-name

名称需满足应用程序定义中的约束条件，应用程序定义是作为 AWS Mainframe Modernization 应用程序的一部分创建的。您可以使用 `root-folder-name` 来区分应用程序的不同版本，例如 V1 和 V2。

application-name

已迁移的应用程序的名称，例如， PlanetsDemo 或 BankDemo。

数据库

重构后的应用程序和更换平台后的应用程序都可能需要数据库。必须根据每个运行时引擎的特定要求来创建、配置和管理数据库。AWSMainframe Modernization 支持在此数据库上进行传输加密。如果在数据库上启用 SSL，请确保在数据库密钥中指定 `sslMode` 以及数据库的连接详细信息。有关更多信息，请参阅 AWS Secrets Manager 密钥。

如果您使用 AWS Blu Age 重构模式，并且需要 BluSam 数据库，则AWS蓝光时代运行时引擎需要一个 Amazon Aurora PostgreSQL 数据库，您必须创建、配置和管理该数据库。 BluSam 数据库是可选的。仅在您的应用程序需要时创建此数据库。要创建数据库，请按照《Amazon Aurora 用户指南》中创建 Amazon Aurora 数据库集群中的步骤进行操作。

如果您使用的是 Micro Focus 更换平台模式，则可以创建 Amazon RDS 或 Amazon Aurora PostgreSQL 数据库。要创建数据库，请按照《Amazon RDS 用户指南》中创建 Amazon RDS 数据库实例或《Amazon Aurora 用户指南》中创建 Amazon Aurora 数据库集群中的步骤进行操作。

对于这两个运行时引擎，您都必须将数据库凭证存储在 AWS Secrets Manager 中，并使用 AWS KMS key对其进行加密。

AWS Key Management Service 密钥

您必须将应用程序数据库的凭证安全地存储在 AWS Secrets Manager 中。要在 Secrets Manager 中创建密钥，必须创建 AWS KMS key。要创建 KMS 密钥，请参阅《AWS Key Management Service 开发人员指南》中的创建密钥。

创建密钥后，必须更新密钥策略以授予 AWS Mainframe Modernization 解密权限。添加以下策略语句：

```
{
   "Effect" : "Allow",
   "Principal" : {
   "Service" : "m2.amazonaws.com"
   },
   "Action" : "kms:Decrypt",
   "Resource" : "*"
   }
```

AWS Secrets Manager 密钥

您必须将应用程序数据库的凭证安全地存储在 AWS Secrets Manager 中。要创建密钥，请按《AWS Secrets Manager 用户指南》中创建数据库密钥的步骤操作。

AWS Mainframe Modernization 支持在此数据库上进行传输加密。如果在数据库上启用 SSL，请确保在数据库密钥中指定 `sslMode` 以及数据库的连接详细信息。可以为 `sslMode` 指定以下值之一：`verify-full`、`verify-ca` 或 `disable`。

在密钥创建过程中，选择资源权限 – 可选，然后选择编辑权限。在策略编辑器中，添加基于资源的策略（如下所示）以检索加密字段的内容。

```
{
   "Effect" : "Allow",
   "Principal" : {
   "Service" : "m2.amazonaws.com"
   },
   "Action" : "secretsmanager:GetSecretValue",
   "Resource" : "*"
   }
```

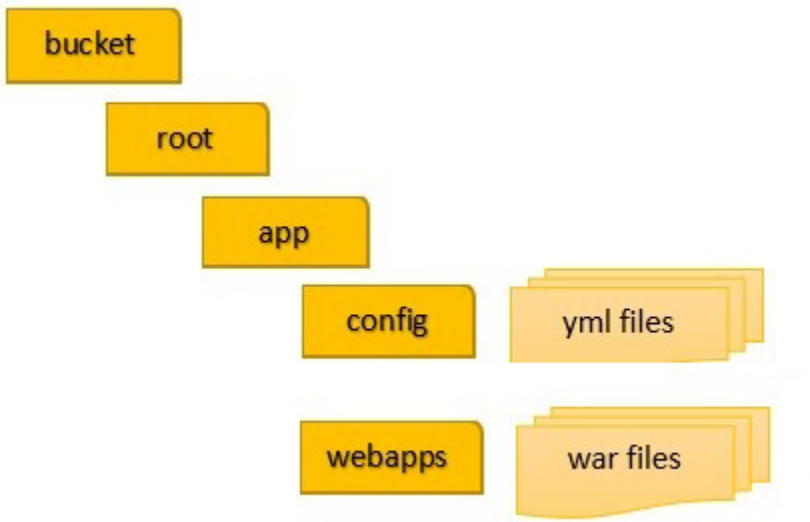
配置托管应用程序

您可以配置应用程序以包含对旧版实用程序的访问权限，也可以自定义其他属性。为了了解您可以配置什么以及在何处配置，了解 AWS 蓝时代现代化应用程序的整体结构会有所帮助。

主题

AWS Blu Age 托管应用程序的结构

如果您使用 AWS Blu Age 重构模式，则 AWS Blu Age 运行时引擎需要在 S3 存储桶的`application-name`文件夹中包含以下结构：



config

包含您的项目的 YAML 文件。这些是特定于您的应用程序的 YAML 文件，通常命名类似`application-planetsdemo.yaml`于 AWS 大型机现代化为您自动提供和设置的`application-main.yaml`文件。

webapps

包含您的应用程序的 `war` 文件。这些文件是在现代化过程中产生的。

应用程序还可以具有以下可选文件夹：

jics/sql

包含用于为应用程序初始化 JICS 数据库的 `initJics.sql` 脚本。

scripts

包含应用程序脚本，您也可以直接在 `war` 文件中提供这些脚本。

sql

包含应用程序 SQL 文件，您也可以直接在 `war` 文件中提供这些脚本。

lnk

包含应用程序 LNK 文件，您也可以直接在 `war` 文件中提供这些脚本。

管理应用程序的 Java 内存消耗

要管理应用程序的 Java 内存消耗，请将名为 `tomcat.properties` 的属性文件添加到 `application-name` 文件夹中。此文件可包含两个属性：`xms`（指定最小 Java 内存消耗）和 `xmx`（指定最大 Java 内存消耗）。以下是有效 `tomcat.properties` 文件的内容示例：

```
xms=512M
xmx=1G
```

您为这两个属性指定的值可以采用以下任何单位：

- 字节：不需指定单位。
- 千字节：在值后面加一个 K。
- 兆字节：在值后面加一个 M。
- 千兆字节：在值后面加一个 G。

为托管应用程序配置对实用程序的访问权限

使用 AWS Blu Age 重构大型机应用程序时，如果您的应用程序依赖于各种传统平台实用程序，例如 IDCAMS、INFUTILB、SORT 等，则可能需要为它们提供支持。 AWS Blu Age 重构通过与现代化应用程序一起部署的专用 Web 应用程序为这种访问提供了这种访问权限。此 Web 应用程序需要您提供配

置文件 application-utility-pgm.yml。如果您未提供此配置文件，则 Web 应用程序无法与您的应用程序一并部署，并且不可用。

主题

- 配置属性

本主题介绍了您可以在 application-utility-pgm.yml 配置文件中指定的所有可能属性及其默认值，包括必需属性和可选属性。以下是一个完整的配置文件示例，其中的属性按我们推荐的顺序列出。您可以将此示例为起点，设置自己的配置文件。

```
# If the datasource support mode is not static-xa, spring JTA transactions
 autoconfiguration must be disabled
 spring.jta.enabled: false
 logging.config: 'classpath:logback-utility.xml'

 # Encoding
 encoding: cp1047

 # Encoding to be used by INFUTILB and DSNUTILB to generate and read SYSPUNCH files
 sysPunchEncoding: cp1047

 # Utility database access
 spring.aws.client.datasources.primary.secret: `arn:aws:secretsmanager:us-
west-2:111122223333:secret:business-FfmXLG`

 treatLargeNumberAsInteger: false

 # Zoned mode : valid values = EBCDIC_STRICT, EBCDIC_MODIFIED, AS400
 zonedMode: EBCDIC_STRICT

 jcl.type: mvs

 # Unload properties
 # For date/time: if use database configuration is enabled, formats are ignored
 # For nbi; use hexadecimal syntaxe to specify the byte value
 unload:
  sqlCodePointShift: 384
  nbi:
    whenNull: "6F"
    whenNotNull: "00"
  useDatabaseConfiguration: false
  format:
```

```
    date: MM/dd/yyyy
    time: HH.mm.ss
    timestamp: yyyy-MM-dd-HH.mm.ss.SSSSSS
  chunkSize:500
  fetchSize: 500
  varCharIsNull: false
  columnFiller: space

# Load properties
# Batch size for DSNUTILB Load Task
load:
  sqlCodePointShift: 384
  batchSize: 500
  format:
    localDate: dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd
    dbDate: yyyy-MM-dd
    localTime: 'HH:mm:ss|HH.mm.ss'
    dbTime: 'HH:mm:ss'

table-mappings:
  TABLE_1_NAME : LEGACY_TABLE_1_NAME
  TABLE_2_NAME : LEGACY_TABLE_2_NAME
```

配置属性

您可以在配置文件中指定以下属性。

spring.jta.enabled

可选。控制是否启用 JTA 支持。对于实用程序，我们建议您将此值设置为 `false`。

```
spring.jta.enabled : false
```

logging.config

必需。指定专用记录器配置文件的路径。建议使用名称 `logback-utility.xml` 并在现代化的应用程序中使用此文件。组织这些文件的常用方法是将所有记录器配置文件放在同一个位置，通常放在子文件夹 `/config/logback` 中，其中 `/config` 是包含 yaml 配置文件的文件夹。有关更多信息，请参阅 Logback 文档中的第三章：Logback 配置。

```
logging.config : classpath:logback-utility.xml
```

encoding

必需。指定实用程序使用的字符集。在大多数情况下，从 z/OS 平台迁移时，此字符集是 EBCDIC 变体，应与为现代化应用程序配置的字符集相匹配。如果未设置，默认值为 ASCII。

```
encoding : cp1047
```

sysPunchEncoding

可选。指定 INFUTILB 和 DSNUTILB 用于生成和读取 SYSPUNCH 文件的字符集。如果您照原样使用来自旧平台的 SYSPUNCH 文件，则此值应该是 EBCDIC 变体。如果未设置，默认值为 ASCII。

```
sysPunchEncoding : cp1047
```

主要数据源配置

某些与数据库相关的实用程序（例如 LOAD 和 UNLOAD）需要通过数据来源访问目标数据库。与 AWS 大型机现代化中的其他数据源定义一样，此访问权限需要您使用 AWS Secrets Manager。指向 Secrets Manager 中正确密钥的属性如下所示：

spring.aws.client.datasources.primary.secret

可选。指定 Secrets Manager 中包含数据来源属性的密钥。

```
spring.aws.client.datasources.primary.secret: datasource-secret-ARN
```

spring.aws.client.datasources.primary.dbname

可选。如果数据库密钥中未直接提供数据库名称，则使用 dbname 属性指定目标数据库名称。

```
spring.aws.client.datasources.primary.dbname: target-database-name
```

treatLargeNumberAsInteger

可选。与 Oracle 数据库引擎细节和 DSNTEP2/DSNTEP4 实用程序的使用有关。如果将此标志设置为 true，则来自 Oracle 数据库的大数字 (NUMBER (38,0)) 将被作为整数处理。默认：false

```
treatLargeNumberAsInteger : false
```

zonedMode

可选。设置分区模式，以对分区数据类型进行编码或解码。此设置会影响有符号数字的表示方式。有效值如下：

- EBCDIC_STRICT：默认值。对符号处理使用严格的定义。根据字符集是 EBCDIC 还是 ASCII，有符号数字表示使用以下字符：
  - 与字节 (Cn+Dn) 对应的 EBCDIC 字符表示正数和负数范围（+0 至 +9、-0 至 -9）。字符显示为 {,A 至 I, }、J 至 R
  - 与字节 (3n+7n) 对应的 ASCII 字符表示正数和负数范围（+0 至 +9、-0 至 -9）。字符显示为 0 至 9、 p 至 y。
- EBCDIC_MODIFIED：实用修改后的定义进行符号处理。对于 EBDIC 和 ASCII，相同的字符列表表示有符号数字，即 +0 至 +9 映射为 { + A 至 I 并且 -0 至 -9 映射为 } + J 至 R. \
- AS400：用于来自 iSeries (AS400) 平台的现代化的传统资产。

```
zonedMode:EBCDIC_STRICT
```

jcl.type

可选。表示现代化 JCL 脚本的传统类型。如果调用的 JCL 类型为 vse，IDCAMS 实用程序将使用此设置来定制返回码。有效值如下所示：

- mvs（默认值）
- vse

```
jcl.type : mvs
```

数据库卸载实用程序相关属性

使用这些属性来配置用于将数据库表卸载到数据集的实用程序。以下所有属性均为可选属性。

此示例显示了所有可能的卸载属性。

```
# Unload properties
 # For date/time: if use database configuration is enabled, formats are ignored
 # For nbi; use hexadecimal syntaxe to specify the byte value
 unload:
 sqlCodePointShift: 0
 nbi:
```

```
whenNull: "6F"
whenNotNull: "00"
useDatabaseConfiguration: false
format:
date: MM/dd/yyyy
time: HH.mm.ss
timestamp: yyyy-MM-dd-HH.mm.ss.SSSSSS
chunkSize: 0
fetchSize: 0
varCharIsNull: false
columnFiller: space
```

sqlCodePoint移动

可选。指定一个整数值，该值表示数据上使用的 SQL 代码点转换。默认值是 0。这表示不进行代码点转换。将此设置与用于现代化应用程序的 SQL 代码点转换参数保持一致。当使用码点转换时，此参数的常用值为 384。

```
unload.sqlCodePointShift: 0
```

nbi

可选。指定一个空的指示符字节，以十六进制值（作为字符串）形式添加到数据值右侧的。两个可能的值如下所示：

- whenNull：当数据值为空时，添加十六进制值。默认值为 6`。有时会改用高值 FF。

  ```
  unload.nbi.whenNull: "6F"
  ```

- whenNotNull：当数据值不为空但列可以为空时，添加十六进制值。默认值为 00（低值）。

  ```
  unload.nbi.whenNotNull: "00"
  ```

useDatabaseConfiguration

可选。指定日期和时间格式化属性，用于处理 UNLOAD 查询中的日期/时间对象。默认值为 `false`。

- 如果设置为 `true`，则使用主配置文件 (`application-main.yml`) 中的 `pgmDateFormat`、`pgmTimeFormat` 和 `pgmTimestampFormat` 属性。
- 如果设置为 `false`，则使用以下日期和时间格式化属性：
  - `unload.format.date`：指定日期格式化模式。默认值为 `MM/dd/yyyy`。

- unload.format.time：指定时间格式化模式。默认值为 HH.mm.ss。
- unload.format.timestamp：指定时间戳格式化模式。默认值为 yyyy-MM-dd-HH.mm.ss.SSSSSS。

chunkSize

可选。指定用于创建 SYSREC 数据集的数据块的大小。这些数据集是数据集卸载操作的目标，可用于并行操作。默认值为 0（无数据块）。

```
unload.chunkSize:0
```

fetchSize

可选。指定数据提取大小。该值是使用数据块策略时一次要提取的记录数。默认值：0。

```
unload.fetchSize:0
```

varCharIs空

可选。指定如何处理不可为空的 varchar 列内容为空的情况。默认值为 false。

如果将此值设置为 true，则出于卸载目的，列内容将被作为空字符串（而不是单个空格字符串）进行处理。仅针对 Oracle 数据库引擎将此标志设置为 true。

```
unload.varCharIsNull: false
```

columnFiller

可选。指定用于填充 varchar 列中已卸载列的值。可能的值为 space 或低值。默认值为 space。

```
unload.columnFiller: space
```

与数据库加载相关的属性

使用以下属性来配置将数据集记录加载到目标数据库（例如 DSNUTILB）的实用程序。以下所有属性均为可选属性。

此示例显示了所有可能的加载属性。

```
# Load properties
```

```
# Batch size for DSNUTILB Load Task
load:
sqlCodePointShift: 384
batchSize: 500
format:
localDate: dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd
dbDate: yyyy-MM-dd
localTime: HH:mm:ss|HH.mm.ss
dbTime: HH:mm:ss

table-mappings:
TABLE_1_NAME : LEGACY_TABLE_1_NAME
TABLE_2_NAME : LEGACY_TABLE_2_NAME
```

sqlCodePoint移动

可选。指定一个整数值，该值表示数据上使用的 SQL 代码点转换。默认值为 0，这表示应用程序没有代码点转换。将此设置与用于现代化应用程序的 SQL 代码点转换参数保持一致。当使用代码点转换时，此参数的常用值为 384。

```
load.sqlCodePointShift : 384
```

batchSize

可选。指定一个整数值，该值表示在向数据库发送实际批处理语句之前要处理的记录数。默认值为 0。

```
load.batchSize: 500
```

format

可选。指定在数据库加载操作期间用于日期/时间转换的日期和时间格式化模式。

- `load.format.localDate`：本地日期格式化模式。默认值为 `dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd`。
- `load.format.dbDate`：数据库日期格式化模式。默认值为 `yyyy-MM-dd`。
- `load.format.localTime`：本地时间格式化模式。默认值为 `HH:mm:ss|HH.mm.ss`。
- `load.format.dbTime`：数据库时间格式化模式。默认值为 `HH:mm:ss`。

table-mappings

可选。指定客户提供的旧表名和现代表名之间的映射集合。DSNUTILB 实用程序会使用这些映射。

按以下格式指定值：MODERN_TABLE_NAME : LEGACY_TABLE_NAME

示例如下：

```
table-mappings:
 TABLE_1_NAME : LEGACY_TABLE_1_NAME
 TABLE_2_NAME : LEGACY_TABLE_2_NAME
 ...
 TABLE_*N*_NAME : LEGACY_TABLE_*N*_NAME
```

> **Note**
>
> 实用程序应用程序启动时，会显式记录所有提供的映射。

为 AWS Blu Age 引擎添加配置属性

你可以在重构后的应用程序的`config`文件夹中添加一个文件，这样你就可以访问 AWS Blu Age 运行时引擎中的新功能。您必须将此文件命名为 `user-properties.yml`。此文件不会替换应用程序定义，而是对其进行了扩展。本主题介绍可以在 `user-properties.yml` 文件中包含的属性。

> **Note**
>
> 您无法更改某些参数，因为它们要么由 AWS 大型机现代化控制，要么由应用程序定义控制。在应用程序定义中为应用程序定义的所有参数均比您在 `user-properties.yml` 中指定的参数优先级高。

有关重构应用程序结构的更多信息，请参阅AWS Blu Age 托管应用程序的结构。

下图显示了在 AWS Blu Age 示例应用程序结构中的哪个位置可以找到该`user-properties.yml`文件 PlanetsDemo。

```
PlanetsDemo-v1/
   ## config/
   #  ## application-PlanetsDemo.yml
   #  ## user-properties.yml
   ## jics/
   ## webapps/
```

配置属性参考

以下是可用属性的列表。所有参数都是可选的。

主题

Gapwalk 应用程序属性

bluesam.fileLoading.commitInterval

可选。bluesam 提交间隔。

类型：数字

默认值：100000

card.encoding

可选。卡片编码：与 `useControlMVariable` 一并使用。

类型：字符串

默认值：CP1145

checkinputfilesize

可选。指定在文件大小为记录大小倍数的情况下是否要进行检查。

类型：布尔值

默认值：false

database.cursor.overflow.allowed

可选。指定是否允许光标溢出。设置为 `true` 时，无论光标位置如何，均可在光标处执行下一次调用。设置为 `false` 时，会在光标处执行下一次调用之前检查光标是否位于最后一个位置。仅在光标可滚动（敏感或不敏感）的情况下将其启用

类型：布尔值

默认：True

数据简化器。 onInvalidNumeric数据

可选。解码无效的数字数据时如何应对。允许的值包括：`reject`、`toleratespaces`、`toleratespaceslowvalues`、`toleratemost`。

类型：字符串

默认：reject

defaultKeepExisting档案

可选。指定是否设置数据集的默认先前值。

类型：布尔值

默认：false

disposition.checkexistence

可选。指定是否针对配置 DISP SHR 或 OLD 的数据集检查文件是否存在。

类型：布尔值

默认：false

externalSort.threshold

可选。排序阈值：何时切换到外部（合并）排序。

类型：字符串

默认值：null

```
externalSort.threshold: 12MB
```

forceHR

可选。指定是否在控制台或文件输出上使用人类可读的 SYSPRINT。

类型：布尔值

默认：false

forcedDate

可选。在数据库中强制指定特定日期和时间。仅在开发和测试期间使用。

默认值：null

```
forcedDate: 2022-08-26T12:59:58.123456+01:57
```

frozenDate

可选。冻结数据库中的日期和时间。仅在开发和测试期间使用。

默认：false

```
frozenDate: false
```

ims.messages.extendedSize

可选。指定是否在 ims 消息上设置 extendedSize。

类型：布尔值

默认：false

lockTimeout

可选。无法在指定时间范围内获取锁时，事务的超时时间（单位为毫秒）。

类型：数字

默认值：500

mapTransfo.prefixes

可选。转换 controlM 变量时要使用的前缀列表。每一个前缀均用逗号隔开。

类型：字符串

默认值：&,@,%%

查询。 useConcatCondition

可选。指定键条件是否通过键连接构建。

类型：布尔值

默认：false

rollbackOnRTE

可选。指定是否在运行时异常时回滚隐式运行单元事务。

类型：布尔值

默认：false

sctThreadLimit

可选。用于触发脚本的线程限值。

类型：数字

默认：5

sqlCodePoint移动

可选。sql 代码点转换。对在将旧版 rdbms 数据迁移到现代 rdbms 时可能遇到的控制字符的代码点进行转换。例如，您可以指定 384 来匹配 unicode 字符 \u0180。

类型：数字

默认值：0

sqlIntegerOverflow允许

可选。指定是否允许 SQL 整数溢出，即是否允许在主机变量中放置更大的值。

类型：布尔值

默认：false

stepFailWhenAbend

可选。指定在步骤失败或完成执行时是否引发异常中止。

类型：布尔值

默认：True

stopExecutionWhenProgNotFound

可选。指定在找不到程序时是否停止运行。如果设置为 `true`，则在找不到程序时中断运行。

类型：布尔值

默认：True

uppercaseUserInput

可选。指定用户输入是否必须采用大写形式。

类型：布尔值

默认：True

useControlMVariable

可选。指定是否使用 control-M 规范进行变量替换。

类型：布尔值

默认：false

Gapwalk 批处理脚本属性

encoding

可选。批处理脚本项目中使用的编码（不是 groovy）。预期的有效编码包含 `CP1047`、`IBM930`、`ASCII`、`UTF-8`...

类型：字符串

默认值：ASCII

Gapwalk Blugen 属性

managers.trancode

可选。对话管理器转码映射。允许您将 JICS 事务代码映射到对话管理器。预期格式为 `trancode1:dialogManager1;trancode2:dialogManager2;`。

类型：字符串

默认值：null

```
managers.trancode: OR12:MYDIALOG1
```

Gapwalk CL 命令属性

commands-off

可选。要关闭的命令列表，以逗号分隔。允许的值包括：PGM_BASIC、RCVMSG、SNDRCVF、CHGVAR、QCLRDTAQ、RTVJOBA、ADDLFM、ADDPFM、RCVF、该属性在禁用或覆盖现有程序时非常有用。PGM_BASIC 是专为调试目的而设计的特定 Velocity 程序。

类型：字符串

默认值：null

spring.datasource.primary.jndi-name

可选。主要 Java 命名和目录接口 (jndi) 数据源。

类型：字符串

默认值：jdbc/primary

zonedMode

可选。对分区数据类型进行编码或解码的模式。允许的值包括：EBCDIC_STRICT、EBCDIC_MODIFIED、AS400。

类型：字符串

默认值：EBCDIC_STRICT

Gapwalk CL 运行程序属性

cl.configuration.context.encoding

可选。CL 文件的编码。预期的有效编码包含 CP1047、IBM930、ASCII、UTF-8...

类型：字符串

默认值：CP297

cl.zonedMode

可选。对控制语言 (CL) 命令进行编码或解码的模式。允许的值包括：EBCDIC_STRICT、EBCDIC_MODIFIED、AS400。

类型：字符串

默认值：EBCDIC_STRICT

Gapwalk JHDB 属性

ims.programs

可选。要使用的 IMS 程序列表。用分号 (;) 分隔每个参数，用逗号 (,) 分隔每个事务。例如：`ims.programs: PCP008,PCT008;PCP054,PCT054;PCP066,PCT066;PCP068,PCT068;`。

类型：字符串

默认值：null

jhdb.checkpointPath

可选。如果 `jhdb.checkpointPersistence` 不是 `none`，则此参数允许您设置检查点持久性路径（checkpoint.dat 文件存储位置），注册表中包含的所有检查点数据都将序列化并备份到所提供文件夹的文件 (checkpoint.dat) 中。请注意，此备份仅涉及检查点数据（scriptId、stepId、数据库位置和检查点区域）。

类型：字符串

默认值：file:./setup/

jhdb.checkpointPersistence

可选。检查点持久性模式。允许的值包括：none、add、end。使用 add 可在创建新检查点并将其添加到注册表后保留该检查点。使用 end 可在服务器关闭时保留检查点。任何其他值都会禁用持久性。请注意，每次向注册表中添加新的检查点时，所有现有的检查点都将被序列化并且文件会被擦除，而不是添加到文件中现有数据中。因此，根据检查点的数量，可能会对性能产生一些影响。

类型：字符串

默认值：none

jhdb.configuration.context.encoding

可选。Java 分层数据库 (JHDB) 编码。预期的有效编码字符串包含 `CP1047、IBM930、ASCII、UTF-8...`

类型：字符串

默认值：CP297

jhdb。 identificationCardData

可选。用于将某些“操作员识别卡数据”硬编码到 CARD 参数指定的 MID 字段。

类型：字符串

默认值：""

jhdb.lterm

可选。允许您在进行 IMS 仿真的情况下强制使用通用的逻辑终端 ID。如果未设置，则使用会话 ID。

类型：字符串

默认值：null

jhdb.metadata.extrapath

用于为 psbs 和 dbds 文件夹指定一个特定于运行时的额外根文件夹的配置参数。

类型：字符串

默认值：file:./setup/

> **Note**
>
> 目前，对于部署限制，您必须将 dbds 和 psbs 目录复制到应用程序的配置目录或配置目录的子目录中，例如 config/setup
>
> ```
> config
> |- setup
>    |- dbds
>    |- psbs
> ```
>
> 并在 application-jhdb.yml 中设置 `jhdb.metadata.extrapath: file: ./config/setup/`

jhdb.navigation.cachenexts

可选。RDBMS 分层导航中使用的缓存持续时间（以毫秒为单位）。

类型：数字

默认值：5000

jhdb.query。 limitJoinUsage

可选。指定是否在 RDBMS 图形上使用限制联接使用参数。

类型：布尔值

默认：True

jhdb。 use-db-prefix

可选。指定是否在 RDBMS 的分层导航中启用数据库前缀。

类型：布尔值

默认：True

Gapwalk JICS 属性

jics.data。 dataJsonInit地点

可选。分析器在解析 CSD 时准备的 json 文件的位置，该文件用于初始化 jics 数据库。

类型：字符串

默认值：""

jics.db。 dataScriptLocation

可选。initJics.sql 脚本的位置，该脚本由 Analyzer 在解析大型机的 CSD 导出时准备。

类型：字符串

默认值：""

jics.db。 dataTestQuery地点

可选。包含单个 sql 查询的 sql 脚本的位置，该查询应返回对象计数（例如：计算 jics 程序表中的记录数）。如果计数等于 0，则将使用 `jics.db.dataScriptLocation` 脚本加载数据库，否则将跳过数据库加载。

类型：字符串

默认值：""

jics.db。 ddlScriptLocation

可选。Jics ddl 脚本位置。允许您使用 .sql 脚本启动 jics 数据库架构。

类型：字符串

默认值：""

```
jics.db.ddlScriptLocation: ./jics/sql/jics.sql
```

jics.db。 schemaTestQuery地点

可选。sql 文件的位置，该文件应包含唯一查询，且该查询返回 jics 架构中对象的数量（如果有）。

类型：字符串

默认值：""

jics。 runUnitLauncherpool.enable

可选。指定是否在 JICS 中激活运行单元启动器池。

类型：布尔值

默认：false

jics。 runUnitLauncher泳池。大小

可选。JICS 中的运行单元启动器池大小。

类型：数字

默认值：20

jics。 runUnitLauncherpool. 验证间隔

可选：JICS 中运行单元启动器池的验证间隔，以毫秒为单位。

类型：数字

默认值：1000

jics.queues.sqs.region

可选。 AWS 区域 适用于 Amazon SQS 的，在 JICS 中使用。为了提高性能，建议将该属性设置为与已部署应用程序相同的区域，但并不强制。

类型：字符串

默认值：eu-west-1

jics.xa.agent.timeout

可选。定义负责管理分布式事务的 xa 代理用于完成其操作的最长持续时间。

类型：数字

默认值：null

mq.queues.sqs.region

可选。 AWS 区域 适用于亚马逊 SQS MQ 服务。

类型：字符串

默认值：eu-west-3

任务执行器。 allowCoreThreadTimeOut

可选。指定是否允许核心线程在 JCIS 中超时。即使与非零队列结合使用，也可以实现动态增长和缩小（因为只有在队列已满后，池最大大小才会增长）。

类型：布尔值

默认：false

任务执行器。 corePoolSize

可选。当通过 groovy 脚本启动终端中的事务时，会创建一个新线程。可以使用此参数设置核心池大小。

类型：数字

默认：5

任务执行器。 maxPoolSize

可选。当通过 groovy 脚本启动终端中的事务时，会创建一个新线程。使用此参数设置池的最大大小（并行线程的最大数量）。

类型：数字

默认值：10

taskExecutor.queueCapacity

可选。当通过 groovy 脚本启动终端中的事务时，会创建一个新线程。使用此参数设置队列大小。（= 达到 `taskExecutor.maxPoolSize` 时待处理事务的最大数量）

类型：数字

默认值：50

Gapwalk 运行时属性

cacheMetadata

可选。指定是否缓存数据库元数据。

类型：布尔值

默认：True

check-groovy-file

可选。指定是否在注册前检查 groovy 文件内容。

类型：布尔值

默认：True

databaseStatistics

可选。指定是否允许 SQL 生成器收集和显示统计信息。

类型：布尔值

默认：false

dateTimeFormat

可选。 dateTimeFormat 描述了如何将数据库日期时间戳类型泄漏到数据简化器实体中。允许的值包括： `ISO`、`EUR`、`USA`、`LOCAL`

类型：字符串

默认值：ISO

dbDateFormat

可选。数据库目标日期格式。

类型：字符串

默认值：yyyy-MM-dd

dbTimeFormat

可选。数据库目标时间格式。

类型：字符串

默认值：HH:mm:ss

dbTimestampFormat

可选。数据库目标时间戳格式。

类型：字符串

默认值：yyyy-MM-dd HH:mm:ss.SSSSSS

fetchSize

可选。游标的 fetchSize 值。在通过加载/卸载 utils 使用区块获取数据时使用。

类型：数字

默认值：10

强制禁用 SQL TrimStringType

可选。指定是否禁用所有 sql 字符串参数的 TRIM。

类型：布尔值

默认：false

localDateFormat

可选。本地日期格式列表。使用 | 分隔每种格式。

类型：字符串

localTimeFormat

可选。本地时间格式列表。使用 | 分隔每种格式。

类型：字符串

localTimestampFormat

可选。本地时间戳格式列表。使用 | 分隔每种格式。

类型：字符串

默认值：

pgmDateFormat

可选。程序中使用的日期时间格式。

类型：字符串

默认值：yyyy-MM-dd

pgmTimeFormat

可选。程序 (pgm) 执行的时间格式。

类型：字符串

默认值：HH.mm.ss

pgmTimestampFormat

可选。时间戳格式

类型：字符串

默认值：yyyy-MM-dd-HH.mm.ss.SSSSSS

Gapwalk 实用程序属性

jcl.type

可选。`.jcl` 文件类型。允许的值包括：`jcl`、`vse`。如果非 vse jcl 的文件为空，IDCAMS 实用程序 PRINT/REPRO 命令将返回 4。

类型：字符串

默认值：mvs

listcat.variablelengthpreprocessor.enabled

可选。指定是否为 LISTCAT 命令启用可变长度预处理器。

类型：布尔值

默认：false

listcat.variablelengthpreprocessor.type

可选。listcat 文件中包含的对象的类型（如果启用 `listcat.variablelengthpreprocessor.enabled`）。允许的值包括：`rdw`、`bdw`。

类型：字符串

默认为：rdw

load.batchSize

可选。加载实用程序批次大小。

类型：数字

默认值：0

load.format.dbDate

可选。要使用的加载实用程序数据库格式。

类型：字符串

默认值：yyyy-MM-dd

load.format.dbTime

可选。要使用的加载实用程序数据库时间。

类型：字符串

默认值：HH:mm:ss

load.format.localDate

可选。要使用的加载实用程序本地日期格式。

类型：字符串

默认值：dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd

load.format.localTime

可选。要使用的加载实用程序本地时间格式。

类型：字符串

默认值：HH:mm:ss|HH.mm.ss

加载。 sqlCodePoint移动

可选。加载实用程序的 SQL 代码点转换。运行字符转换进程。如果来自 DB2 目标数据库的是 Postgresql，则该属性是必需的。

类型：数字

默认值：0

sysPunchEncoding

可选。syspunch 编码字符集。支持的值为 `Cp1047`、`ASCII`。

类型：字符串

默认值：ASCII

treatLargeNumberAsInteger

可选。指定是否将大数字作为 `Integer` 处理。默认情况下，大数字作为 `BigDecimal` 处理。

类型：布尔值

默认：false

unload.chunkSize

可选。卸载实用程序使用的区块大小。

类型：数字

默认值：0

unload.columnFiller

可选。卸载实用程序列填充内容。

类型：字符串

默认值：space

unload.fetchSize

可选。允许您在卸载实用程序中处理光标时调整提取大小。

类型：数字

默认值：0

unload.format.date

可选。卸载实用程序中使用的日期格式（如果启用 `unload.useDatabaseConfiguration`）。有关更多信息，请参阅 unload.format.date。

类型：字符串

默认值：MM/dd/yyyy

unload.format.time

可选。卸载实用程序中使用的时间格式（如果启用 `unload.useDatabaseConfiguration`）。

类型：字符串

默认值：HH.mm.ss

unload.format.timestamp

可选。卸载实用程序中使用的时间戳格式（如果启用 `unload.useDatabaseConfiguration`）。

类型：字符串

默认值：yyyy-MM-dd-HH.mm.ss.SSSSSS

unload.nbi。 whenNotNull

可选。数据库中的值不为空时，要添加的空字节指示符 (nbi) 值。

类型：十六进制

默认值：00

unload.nbi.whenNull

可选。数据库中的值为空时，要添加的空字节指示符 (nbi) 值。

类型：十六进制

默认值：6F

unload.nbi。 writeNullIndicator

可选。指定是否在卸载输出文件中写出空指示符。

类型：布尔值

默认：false

卸载。 sqlCodePoint移动

可选。卸载实用程序的 SQL 代码点转换。运行字符转换进程。如果来自 DB2 目标数据库的是 Postgresql，则该属性是必需的。

类型：数字

默认值：0

卸载。 useDatabaseConfiguration

可选。指定是否在卸载实用程序中使用 application-main.yml 中的日期或时间配置。

类型：布尔值

默认：false

卸载。 varCharIs空

可选。在 INFTILB 程序中使用此参数，如果设置为 `true`，则所有具有空（空格）值的不可为空的字段都将返回一个空字符串。

类型：布尔值

默认：false

其他属性

qtemp.cleanup.threshold.hours

可选。指定何时启用 `qtemp.dblog`。数据库分区的生命周期（以小时为单位）。

类型：数字

默认值：0

qtemp.dblog

可选。是否启用 QTEMP 数据库日志记录。

类型：布尔值

默认：false

qtemp.uuid.length

可选。QTEMP 的唯一 ID 长度。

类型：数字

默认值：9

quartz.scheduler。 stand-by-if-error

可选。指定当作业调度程序处于待机模式时是否触发作业执行。如果设置为 true，则启用时不会触发作业执行。

类型：布尔值

默认：false

warmUpCache

可选。指定是否在服务器启动时将所有数据通信表数据加载到预热缓存中。

类型：布尔值

默认：false

AWS 大型机现代化应用程序定义参考

在 AWS 大型机现代化中，您可以在应用程序定义 JSON 文件中配置迁移的大型机应用程序，该文件特定于您选择的运行时引擎。应用程序定义既包含一般信息，也包含特定于引擎的信息。本主题介绍了 AWS Blu Age 和 Micro Focus 应用程序的定义，并确定了所有必需元素和可选元素。

主题

- 定义区段概述
- AWS Blu Age 应用程序定义示例
- AWS Blu Age 定义详情
- Micro Focus 应用程序定义
- Micro Focus 定义详细信息

一般头部区段

每个应用程序定义都首先指定有关模板版本和源位置的一般信息。应用程序定义的当前版本为 2.0。尽管版本 1 仍然有效，但它正被逐渐弃用。我们建议您在创建或更新应用程序时使用版本 2。

使用以下结构来指定模板版本和源位置。

```
"template-version": "2.0",
    "source-locations": [
        {
            "source-id": "s3-source",
            "source-type": "s3",
            "properties": {
                "s3-bucket": "mainframe-deployment-bucket-aaa",
                "s3-key-prefix": "v1"
            }
        }
    ]
```

> **Note**
>
> 如果要将 S3 ARN 作为 s3 存储桶输入，则可以使用以下语法：
>
> ```
> "template-version": "2.0",
>   "source-locations": [
>       {
>           "source-id": "s3-source",
>           "source-type": "s3",
>           "properties": {
>               "s3-bucket": "arn:aws:s3:::mainframe-deployment-bucket-aaa",
>               "s3-key-prefix": "v1"
>           }
>       }
> ```

```
]
```

template-version

必需。指定应用程序定义文件的版本。请勿更改此值。目前唯一允许的值为 2.0。使用字符串指定 `template-version`。

source-locations

指定应用程序在运行时所需的文件和其他资源的位置。

属性

提供源位置的详细信息。每个属性均使用字符串指定。

- `s3-bucket` – 必需。存储文件的 Amazon S3 存储桶名称。
- `s3-key-prefix` – 必需。指定存储文件的 Amazon S3 存储桶的文件夹的名称。

> **Note**
>
> 请务必指定 Amazon S3 存储桶的名称，而不是存储桶 ARN。请勿指定存储桶中资源的绝对路径。

定义区段概述

指定应用程序运行所需的服务、设置、数据和其他典型资源的资源定义。更新应用程序定义时，AWS Mainframe Modernization 会通过比较应用程序定义 JSON 文件的先前版本和当前版本中的 `source-locations` 和 `definition` 列表来检测是否发生更改。

定义区段是特定于引擎的，可能会发生变化。以下各节针对两个引擎显示了特定于引擎的应用程序定义示例。

AWS Blu Age 应用程序定义示例

```
{
 "template-version": "2.0",
 "source-locations": [
        {
            "source-id": "s3-source",
            "source-type": "s3",
```

```
            "properties": {
                "s3-bucket": "mainframe-deployment-bucket-aaa",
                "s3-key-prefix": "v1"
            }
        }
    ],
    "definition" : {
        "listeners": [{
            "port": 8194,
            "type": "http"
        }],
        "ba-application": {
            "app-location": "${s3-source}/murachs-v6/"
        },
        "blusam": {
            "db": {
                "nb-threads": 8,
                "batch-size": 10000,
                "name": "blusam",
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:blusam-FfmXLG"
            },
            "redis": {
                "hostname": "blusam.c3geul.ng.0001.usw2.cache.amazonaws.com",
                "port": 6379,
                "useSsl": true,
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:bluesamredis-nioefm"
            }
        }
    }
}
```

AWS Blu Age 定义详情

侦听器 – 必需

指定用于通过 AWS 大型机现代化创建的 Elastic Load Balancing 访问应用程序的端口。请使用以下结构。

```
"listeners": [{
            "port": 8194,
            "type": "http"
```

```
        }],
```

port

必需。除了公认端口 0 到 1023 之外，您可以使用任何可用端口。我们建议使用的端口范围为 8192 至 8199。请确保没有其他侦听器或应用程序使用此端口。

type

必需。目前仅支持 `http`。

AWS Blu Age 应用程序-必填

使用以下结构指定引擎获取应用程序映像文件的位置。

```
"ba-application": {
            "app-location": "${s3-source}/murachs-v6/",
            "files-directory": "/m2/mount/myfolder",
            "enable-jics": <true|false>,
            "shared-app-location": "${s3-source}/shared/"
        },
```

app-location

Amazon S3 中存储应用程序映像文件的特定位置。

files-directory

可选。批处理的输入/输出文件的位置，必须是环境级别的 Amazon EFS 或 Amazon FSx 挂载点设置的子文件夹。

enable-jics

可选。指定是否启用 JICS。默认值为 true。将其设置为 false 可防止生成 JICS 数据库。

shared-app-location

可选。Amazon S3 中存储共享应用程序元素的进一步位置，其中可以包含与 app-location 中相同的应用程序结构。

BluSAM – 可选

使用以下结构指定 BluSAM 数据库和 Redis 缓存。

```
"blusam": {
            "db": {
                "nb-threads": 8,
                "batch-size": 10000,
                "name": "blusam",
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:blusam-FfmXLG"
            },
            "redis": {
                "hostname": "blusam.c3geul.ng.0001.usw2.cache.amazonaws.com",
                "port": 6379,
                "useSsl": true,
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:bluesamredis-nioefm"
            }
        }
```

db

指定与应用程序搭配使用的数据库的属性。该数据库必须为 Aurora PostgreSQL 数据库。您可以指定以下属性：

- nb-threads – 可选。指定 blusam 引擎所依赖的回写机制使用多少个专用线程。默认值为 8。
- batch-size – 可选。指定回写机制用于启动批量存储操作的阈值。该阈值表示已修改记录的数量，这些记录将启动批量存储操作，以确保修改后的记录得以保存。触发器本身基于所耗时间（一秒钟）和批量大小（以先达到者为准）的组合。默认值为 10000。
- name – 可选。指定数据库的名称。
- secret-manager-arn – 指定包含数据库凭证的密钥的 Amazon 资源名称 (ARN)。有关更多信息，请参阅步骤 4：创建和配置AWS Secrets Manager数据库密钥。

redis

指定 Redis 缓存的属性，应用程序使用该缓存将临时数据存储在中央位置以提高性能。我们建议您同时对 Redis 缓存进行加密和实施密码保护。

- hostname – 指定 Redis 缓存的位置。
- port – 指定 Redis 缓存发送和接收数据的通信端口，通常为 6379。
- useSsl – 指定 Redis 缓存是否已加密。如果缓存未加密，则将 useSsl 设置为 false。
- secret-manager-arn – 指定包含 Redis 缓存密码的密钥的 Amazon 资源名称 (ARN)。如果 Redis 缓存没有密码保护，请勿指定 secret-manager-arn。有关更多信息，请参阅步骤 4：创建和配置AWS Secrets Manager数据库密钥。

AWS Blu Age 消息队列——可选

指定 Blu Age 应用程序的 JMS-MQ 连接详细信息。 AWS

```
"message-queues": [
  {
      "product-type": "JMS-MQ",
      "queue-manager": "QMgr1",
      "channel": "mqChannel1",
      "hostname": "mqserver-host1",
      "port": 1414,
      "user-id": "app-user1",
      "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:sample/mq/test-279PTa"
  },
  {
      "product-type": "JMS-MQ",
      "queue-manager": "QMgr2",
      "channel": "mqChannel2",
      "hostname": "mqserver-host2",
      "port": 1412,
      "user-id": "app-user2",
      "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:sample/mq/test-279PTa"
  }
]
```

product-type

必需。指定产品类型。目前，这只能是 Blu Age 应用程序的 “JMS-MQ”。 AWS

queue-manager

必需。指定队列管理器的名称。

channel

必需。指定服务器连接通道的名称。

hostname

必需。指定消息队列服务器的主机名。

port

必需。指定服务器正在监听的侦听器端口号。

user-id

可选。指定允许在指定通道上执行消息队列操作的用户账户 ID。

secret-manager-arn

可选。指定 Secrets Manager 的 Amazon 资源名称 (ARN)，Secrets Manager 提供指定用户的密码。

Micro Focus 应用程序定义

以下示例定义区段适用于 Micro Focus 运行时引擎，包含必需元素和可选元素。

```
{
 "template-version": "2.0",
 "source-locations": [
        {
            "source-id": "s3-source",
            "source-type": "s3",
            "properties": {
                "s3-bucket": "mainframe-deployment-bucket-aaa",
                "s3-key-prefix": "v1"
            }
        }
    ],
    "definition" : {
        "listeners": [{
            "port": 5101,
            "type": "tn3270"
        }],
        "dataset-location": {
            "db-locations": [{
                "name": "Database1",
                "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456"
            }]
        },
        "cognito-auth-handler": {
            "user-pool-id": "cognito-idp.us-west-2.amazonaws.com/us-west-2_rvYFnQIxL",
            "client-id": "58k05jb8grukjjsudm5hhn1v87",
            "identity-pool-id": "us-west-2:64464b12-0bfb-4dea-ab35-5c22c6c245f6"
        },
        "ldap-ad-auth-handler": {
            "ldap-ad-connection-secrets": [LIST OF AD-SECRETS]
```

```
        },
        "batch-settings": {
            "initiators": [{
                "classes": ["A", "B"],
                "description": "initiator...."
            }],
            "jcl-file-location": "${s3-source}/batch/jcl"
        },
        "cics-settings": {
            "binary-file-location": "${s3-source}/cics/binaries",
            "csd-file-location": "${s3-source}/cics/def",
            "system-initialization-table": "BNKCICV"
        },
        "xa-resources" : [{
            "name": "XASQL",
            "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456",
            "module": "${s3-source}/xa/ESPGSQLXA64.so"
        }]
    }
}
```

Micro Focus 定义详细信息

Micro Focus 应用程序定义文件中定义区段的内容会有所不同，具体取决于迁移的大型机应用程序在运行时所需的资源。

侦听器 – 必需

使用以下结构指定侦听器：

```
"listeners": [{
    "port": 5101,
    "type": "tn3270"
}],
```

port

对于 tn3270，默认值为 5101。对于其他类型的服务侦听器，端口会有所不同。除了公认端口 0 到 1023 之外，您可以使用任何可用端口。每个侦听器都应使用一个独有的端口。侦听器不应共享端口。有关更多信息，请参阅 Micro Focus Enterprise Server 文档中的 Listener Control。

type

指定服务侦听器的类型。有关更多信息，请参阅 Micro Focus Enterprise Server 文档中的侦听器。

数据集位置 – 必需

使用以下结构指定数据集的位置。

```
"dataset-location": {
            "db-locations": [{
                "name": "Database1",
                "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456"
            }],
 }
```

db-locations

指定迁移的应用程序创建的数据集的位置。目前，AWS 大型机现代化仅支持来自单个 VSAM 数据库的数据集。

- `name` – 指定包含已迁移应用程序创建的数据集的数据库实例的名称。
- `secret-manager-arn` – 指定包含数据库凭证的密钥的 Amazon 资源名称 (ARN)。

Amazon Cognito 身份验证和授权处理程序 – 可选

AWS 大型机现代化使用 Amazon Cognito 对迁移的应用程序进行身份验证和授权。使用以下结构指定 Amazon Cognito 身份验证处理程序。

```
"cognito-auth-handler": {
            "user-pool-id": "cognito-idp.Region.amazonaws.com/Region_rvYFnQIxL",
            "client-id": "58k05jb8grukjjsudm5hhn1v87",
            "identity-pool-id": "Region:64464b12-0bfb-4dea-ab35-5c22c6c245f6"
}
```

user-pool-id

指定 AWS 大型机现代化用于对迁移应用程序的用户进行身份验证的 Amazon Cognito 用户池。用户池 AWS 区域 的应与 AWS 大型机现代化应用程序 AWS 区域 的相匹配。

client-id

指定经过身份验证的用户可访问的已迁移应用程序。

identity-pool-id

指定 Amazon Cognito 身份池，经过身份验证的用户可以在其中交换用户池令牌以获取允许用户访问 AWS 大型机现代化的证书。身份池 AWS 区域 的应与 AWS 大型机现代化应用程序 AWS 区域 的相匹配。

LDAP 和 Active Directory 处理程序 - 可选

您可以将您的应用程序与 Active Directory（AD）或任何类型的 LDAP 服务器集证进行授权和身份验证。

将应用程序与 AD 集成

1. 按照 Micro Focus Enterprise Server 文档中为 Enterprise Server 安全配置 Active Directory 中描述的步骤进行操作。
2. 使用您的 AD/LDAP 详细信息为要用于应用程序的每台 AD/LDAP 服务器创建一个 AWS Secrets Manager 密钥。有关如何创建密钥的信息，请参阅 AWS Secrets Manager 用户指南中的创建 AWS Secrets Manager 密钥。对于密钥类型，请选择其他类型的密钥，并包括以下键值对。

```
{
    "connectionPath"       : "<HOST-ADDRESS>:<PORT>",
    "authorizedId"         : "<USER-FULL-DN>",
    "password"             : "<PASSWORD>",
    "baseDn"               : "<BASE-FULL-DN>",
    "userClassDn"          : "<USER-TYPE>",
    "userContainerDn"      : "<USER-CONTAINER-DN>",
    "groupContainerDn"     : "<GROUP-CONTAINER-DN>",
    "resourceContainerDn"  : "<RESOURCE-CONTAINER-DN>"
}
```

⚠ 安全建议

- 对于connectionPath，AWS 大型机现代化支持 LDAP 和基于 SSL 的 LDAP (LDAPS) 协议。建议您使用 LDAPS，因为它更安全，并且可以防止凭证出现在网络传输中。

- 对于 authorizedId 和 password，建议您依据以下原则为用户指定凭证：权限不超过应用程序运行所需的最严格的只读和验证权限。
- 建议您定期变换 AD/LDAP 凭证。
- 不要使用用户名 awsuser 或 mfuser 创建 AD 用户。这两个用户名保留供 AWS 使用。

以下是示例。

```
{
    "connectionPath" : "ldaps://msad4.m2.example.people.aws.dev:636",
    "authorizedId" :
 "CN=LDAPUser,OU=Users,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev",
    "password" : "ADPassword",
    "userContainerDn" : "CN=Enterprise Server Users,CN=Micro Focus,CN=Program
 Data,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev",
    "groupContainerDn" : "CN=Enterprise Server Groups,CN=Micro Focus,CN=Program
 Data,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev",
    "resourceContainerDn" : "CN=Enterprise Server Resources,CN=Micro
 Focus,CN=Program Data,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev"
}
```

使用客户托管的 KMS 密钥创建密钥。您必须向 AWS 大型机现代化授予密钥的GetSecretValue和DescribeSecret权限，Decrypt以及 KMS 密钥的DescribeKey权限。有关更多信息，请参阅《 AWS Secrets Manager 用户指南》中的 KMS 密钥权限。

3. 将以下内容添加到您的应用程序定义中。

```
"ldap-ad-auth-handler": {
    "ldap-ad-connection-secrets": [LIST OF AD/LDAP SECRETS]
}
```

以下是示例。

```
"ldap-ad-auth-handler": {
    "ldap-ad-connection-secrets": ["arn:aws:secrets:1234:us-east-1:secret:123456"]
}
```

LDAP/AD 身份验证处理程序可用于 Micro Focus 8.0.11 及更高版本。

批处理设置 – 必需

使用以下结构指定作为应用程序一部分运行的批处理作业所需的详细信息。

```
"batch-settings": {
            "initiators": [{
                "classes": ["A","B"],
                "description": "initiator...."
            }],
            "jcl-file-location": "${s3-source}/batch/jcls"
 }
```

initiators

指定一个批处理启动器，该启动器在迁移的应用程序成功启动时启动，并应用程序停止运行前持续运行。您可以为每个启动器定义一个或多个类，也可以定义多个启动器。例如：

```
"batch-settings": {
            "initiators": [
                {
                  "classes": ["A", "B"],
                  "description": "initiator...."
                },
                {
                  "classes": ["C", "D"],
                  "description": "initiator...."
                }
            ],
            "jcl-file-location": "${s3-source}/batch/jcls"
 }
```

有关更多信息，请参阅 Micro Focus Enterprise Server 文档中的定义批处理启动器或打印机 SEP。

- `classes` – 指定启动器可以运行的作业类。最多可以使用 36 个字符。您可以使用以下字符：A-Z 或 0-9。
- `description` – 描述启动器的用途。
- `jcl-file-location` – 指定迁移应用程序运行的批处理作业所需的 JCL 文件的位置。

CICS 设置 – 必需

使用以下结构指定作为应用程序一部分运行的 CICS 事务所需的详细信息。

```
"cics-settings": {
            "binary-file-location": "${s3-source}/cics/binaries",
            "csd-file-location": "${s3-source}/cics/def",
            "system-initialization-table": "BNKCICV"
        }
```

binary-file-location

指定 CICS 事务程序文件的位置。

csd-file-location

指定此应用程序的 CICS 资源定义 (CSD) 文件的位置。有关更多信息，请参阅 Micro Focus Enterprise Server 文档中的 CICS 资源定义。

system-initialization-table

指定迁移的应用程序使用的系统初始化表 (SIT)。SIT 表的名称最多可包含 8 个字符。您可以使用 A-Z、0-9、$、@ 和 #。有关更多信息，请参阅 Micro Focus Enterprise Server 文档中的 CICS 资源定义。

XA 资源 – 必需

使用以下结构指定应用程序所需的 XA 资源所需的详细信息。

```
"xa-resources" : [{
            "name": "XASQL",
            "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456",
            "module": "${s3-source}/xa/ESPGSQLXA64.so"
        }]
```

name

必需。指定 XA 资源的名称。

secret-manager-arn

指定包含用于连接数据库的凭证的密钥的 Amazon 资源名称 (ARN)。

module

指定 RM 交换机模块可执行文件的位置。有关更多信息，请参阅 Micro Focus Enterprise Server 文档中的规划和设计 XAR。

AWS Mainframe Modernization 数据集定义参考

如果您的应用程序需要处理多个数据集，在 AWS Mainframe Modernization 控制台中逐个输入这些数据集时效率低下。建议您创建一个 JSON 文件来指定每个数据集。尽管许多参数都很常见，但在 JSON 中，要以不同的方式指定不同的数据集类型。本文档描述了导入不同类型数据集所需的 JSON 的详细信息。

> **Note**
>
> 在导入任何数据集之前，必须将数据集从大型机传输到 AWS。然后，必须确保将数据集从大型机格式转换为 AWS 可以使用的格式。如有必要，请按需转换数据，并将转换后的数据集存储在 Amazon S3 中。在数据集定义 JSON 文件中指定存储桶和文件夹的名称。
>
> 如果您使用的是 Micro Focus 运行时引擎，则可以使用 DFCONV 实用程序来转换数据集。我们在 Micro Focus Enterprise Developer 和 Enterprise Server 映像中提供此实用程序。有关更多信息，请参阅《Micro Focus Enterprise Developer》文档中的 DFCONV 批处理文件转换。

主题

常用属性

部分参数是所有数据集通用的。这些参数涵盖以下区域：

- 有关数据集的信息（`datasetName`、`datasetOrg`、`recordLength`、`encoding`）。
- 有关从何处导入的信息，即数据集的源位置。该位置不是大型机上的位置，而是您上传数据集的 Amazon S3 位置的路径 (`externalLocation`)。
- 有关导入到何处的信息，即数据集的目标位置。该位置可以是数据库或文件系统，具体取决于您的运行时引擎。（`storageType` 和 `relativePath`）。
- 有关数据集类型的信息（特定数据集类型、格式、编码等）。

每个数据集定义都具有相同的 JSON 结构。以下示例 JSON 显示了所有这些通用参数。

```
{
    "dataSet": {
        "storageType": "Database",
        "datasetName": "MFI01V.MFIDEMO.BNKACC",
        "relativePath": "DATA",
        "datasetOrg": {
            "type": {
                type-specific properties
                    ...
            },
        },
    },
}
```

以下属性是所有数据集的通用属性。

storageType

必需。应用于目标位置。指定数据集是存储在数据库中还是文件系统中。可能的值为 `Database` 或 `FileSystem`。

- AWS Blu Age 运行时引擎：不支持文件系统。您必须使用数据库。
- Micro Focus 运行时引擎：同时支持数据库和文件系统。您可以针对数据库使用 Amazon Relational Database Service 或 Amazon Aurora，也可以针对操作系统使用 Amazon Elastic File System 或适用于 Lustre 的 Amazon FSx。

datasetName

必需。指定在大型机上显示的数据集的完全限定名称。

relativePath

必需。应用于目标位置。指定数据集在数据库或文件系统中的相对位置。

datasetOrg

必需。指定数据集的类型。可能的值为 `vsam`、`gdg`、`ps`、`po` 或 `unknown`。

- AWS Blu Age 运行时引擎：仅支持 VSAM 类型的数据集。
- Micro Focus 运行时引擎：支持 VSAM、GDG、PS、PO 或未知类型的数据集。

> **Note**
>
> 如果您的应用程序需要的文件不是 COBOL 数据文件而是 PDF 或其他二进制文件的文件，则可以按如下方式指定它们：
>
> ```
> "datasetOrg": {
>             "type": PS {
>                 "format": U
>             },
> ```

VSAM 数据集请求格式示例

- AWS Blu Age 运行时引擎：支持。
- Micro Focus 运行时引擎：支持。

如果要导入 VSAM 数据集，请将 `datasetOrg` 指定为 `vsam`。您的 JSON 应与以下示例类似：

```
{
    "storageType": "Database",
    "datasetName": "AWS.M2.VSAM.KSDS",
    "relativePath": "DATA",
    "datasetOrg": {
        "vsam": {

            "encoding": "A",
            "format": "KS",
            "primaryKey": {
                "length": 11,
                "offset": 0
            }
        }
    },
    "recordLength": {
        "min": 300,
        "max": 300
    }
},
```

```
"externalLocation": {
    "s3Location": "s3://$M2_DATA_STORE/catalog/data/AWS.M2.VSAM.KSDS.DAT"
}
```

VSAM 数据集支持以下属性。

encoding

必需。指定数据集的字符集编码。可能的值为 ASCII (A)、EBCDIC (E) 和未知 (?)。

format

必需。指定 VSAM 数据集类型和记录格式。

- AWS Blu Age 运行时引擎：可能的值为 ESDS (ES)、KSDS (KS) 和 RRDS (RR)。记录格式可以是固定的，也可以是可变的。
- Micro Focus 运行时引擎：可能的值为 ESDS (ES)、KSDS (KS) 和 RRDS (RR)。VSAM 定义包括记录格式，因此您无需单独指定它。

primaryKey

仅适用于 VSAM KSDS 数据集。指定主键。包括主键名称、键偏移和键长度。`name` 为可选项，`offset` 和 `length` 为必需项。

recordLength

必需。指定记录的长度。对于固定长度的记录格式，这些值必须匹配。

- AWS Blu Age 运行时引擎：对于 VSAM ESDS、KSDS 和 RRDS，`min` 为可选项，`max` 为必需项。
- Micro Focus 运行时引擎：`min` 和 `max` 均为必需项。

externalLocation

必需。指定源位置：即您上传数据集的 Amazon S3 存储桶。

Blu Age 引擎特有的属性

AWS Blu Age 运行时引擎支持 VSAM 数据集压缩。以下示例说明如何在 JSON 中指定此属性。

```
{
    common properties
        ...
        "datasetOrg": {
            "vsam": {
```

```
                common properties
                ...
                "compressed": boolean,
                common properties
                ...
            }
        }
}
```

按如下方式指定压缩属性：

compression

可选。指定此数据集的索引是否存储为压缩值。如果您的数据集很大（通常大于 100 Mb），请考虑将此标志设置为 `true`。

GDG Base 数据集请求格式示例

- AWS Blu Age 运行时引擎：支持。
- Micro Focus 运行时引擎：支持。

如果要导入 GDG base 数据集，请将 `datasetOrg` 指定为 `gdg`。您的 JSON 应与以下示例类似：

```
{
    "storageType": "Database",
    "datasetName": "AWS.M2.GDG",
    "relativePath": "DATA",
    "datasetOrg": {
        "gdg": {
            "limit": "3",
            "rollDisposition": "Scratch and No Empty"
        }
    }
}
```

GDG base 数据集支持以下属性。

限制

必需。指定活跃生成数或偏差。对于 GDG base 集群，最大值为 255。

rollDisposition

可选。指定在达到或超过最大值时如何处理生成数据集。可能的值为 No Scratch and No Empty、Scratch and No Empty、Scratch and Empty 或 No Scratch and Empty。默认值为 Scratch and No Empty。

PS 或 GDG 生成的数据集请求格式示例

- AWS Blu Age 运行时引擎：支持。
- Micro Focus 运行时引擎：支持。

如果要导入 PS 或 GDG 生成数据集，请将 datasetOrg 指定为 ps。您的 JSON 应与以下示例类似：

```
{
    "storageType": "Database",
    "datasetName": "AWS.M2.PS.FB",
    "relativePath": "DATA",
    "datasetOrg": {
        "ps": {
            "format": "FB",
            "encoding": "A"
        }
    },
    "recordLength": {
        "min": 300,
        "max": 300
    }
},
"externalLocation": {
    "s3Location": "s3://$M2_DATA_STORE/catalog/data/AWS.M2.PS.LSEQ"
}
}
```

PS 或 GDG 生成数据集支持以下属性。

format

必需。指定数据集记录的格式。可能的值为 F、FA、FB、FBA、FBM、FBS、FM、FS、LSEQ、U、V、VA、VB、VBA、VBM、VBS、VM 和 VS。

encoding

必需。指定数据集的字符集编码。可能的值为 ASCII (A)、EBCDIC (E) 和未知 (?)。

recordLength

必需。指定记录的长度。必须指定记录的最小 (min) 和最大 (max) 长度。对于固定长度的记录格式，这些值必须匹配。

externalLocation

必需。指定源位置：即您上传数据集的 Amazon S3 存储桶。

PO 数据集请求格式示例

如果要导入 PO 数据集，请将 datasetOrg 指定为 po。您的 JSON 应与以下示例类似：

```
{
    "storageType": "Database",
    "datasetName": "AWS.M2.PO.PROC",
    "relativePath": "DATA",
    "datasetOrg": {
        "po": {
            "format": "LSEQ",
            "encoding": "A",
            "memberFileExtensions": ["PRC"]
        }
    },
    "recordLength": {
        "min": 80,
        "max": 80
    }
},
"externalLocation": {
    "s3Location": "s3://$M2_DATA_STORE/source/proc/"
}
}
```

PO 数据集支持以下属性。

format

必需。指定数据集记录的格式。可能的值为 F、FA、FB、FBA、FBM、FBS、FM、FS、LSEQ、U、V、VA、VB、VBA、VBM、VBS、VM 和 VS。

encoding

必需。指定数据集的字符集编码。可能的值为 ASCII (A) 、EBCDIC (E) 和未知 (?) 。

memberFileExtensions

必需。指定包含一个或多个文件扩展名的数组，允许您指定要包含哪些文件作为 PDS 成员。

recordLength

可选。指定记录的长度。记录的最小 (min) 和最大 (max) 长度都是可选的。对于固定长度的记录格式，这些值必须匹配。

externalLocation

必需。指定源位置：即您上传数据集的 Amazon S3 存储桶。

> **Note**
>
> Micro Focus 运行时引擎的当前实现将 PDS 条目添加为动态数据集。

AWS Mainframe Modernization 中的托管运行时环境

如果您是首次接触 AWS Mainframe Modernization，请参阅以下主题了解其用法：

- 什么是 AWS 大型机现代化？
- 设置 AWS 大型机现代化
- AWS 大型机现代化入门
- 教程：AWS Blu Age 的托管运行时
- 教程：Micro Focus 的托管运行时

AWS Mainframe Modernization 中的运行时环境是 AWS 计算资源、运行时引擎和您指定的配置详细信息的组合名称。运行时环境托管一个或多个应用程序。AWS Mainframe Modernization 中的应用程序包含已迁移的大型机工作负载。您可以为所创建的环境选择运行时引擎。如果您使用的是自动重构模式，请选择 AWS Blu Age；如果您使用的是更换平台模式，请选择 Micro Focus。您还可以选择适合您的应用程序的计算资源量，也可以选择将存储附加到运行时环境。AWS大型机现代化为您启用 Amazon CloudWatch 指标和日志记录，以便您可以监控运行时环境。

主题

- 创建 AWS Mainframe Modernization 运行时环境
- 更新 AWS Mainframe Modernization 运行时环境
- 停止 AWS Mainframe Modernization 运行时环境
- 重新启动 AWS Mainframe Modernization 运行时环境
- 删除 AWS Mainframe Modernization 运行时环境

创建 AWS Mainframe Modernization 运行时环境

使用 AWS Mainframe Modernization 控制台创建 AWS Mainframe Modernization 环境。

以下说明假定您已完成设置 AWS 大型机现代化 中的步骤。

创建运行时环境

创建运行时环境

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。

2. 在 AWS 区域选择器中，选择要创建环境的区域。
3. 在环境页面上，选择创建环境。
4. 在指定基本信息页面上，提供以下信息：
   a. 在名称和描述部分，输入环境的名称。
   b. （可选）在环境描述字段中，输入环境的描述。该描述有助于您及其他用户了解该运行时环境的用途。
   c. 在引擎选项部分，选择 Blu Age 进行自动重构，或选择 Micro Focus 来更换平台。
   d. 为所选引擎选择一个版本。
   e. （可选）在标签部分中，选择添加新标签以向环境添加一个或多个环境标签。环境标签是一种自定义属性标签，可帮助您组织和管理 AWS 资源。
   f. 选择下一步。
5. 在指定配置页面上，提供以下信息：
   a. 在可用性部分中，选择独立运行时环境或高可用性集群。

      可用性模式决定了您的应用程序在运行时的可用性。独立模式适用于开发用途。高可用性模式适用于必须始终可用的应用程序。
   b. 在资源中，选择实例类型和所需容量。

      这些资源是 AWS Mainframe Modernization 托管的 Amazon EC2 实例，用于托管您的运行时环境。独立运行时环境提供两种实例类型选择，但仅允许一个实例。高可用性运行时环境提供两种实例类型选择，并最多允许两个实例。

      有关更多信息，请参阅 Amazon EC2 实例类型，并联系 AWS 大型机专家寻求指导。
6. 在安全和网络部分，执行以下操作：
   a. 如果您希望应用程序可公开访问，请选择允许部署到此环境的应用程序被公开访问。
   b. 选择一个虚拟私有云（VPC）。
   c. 如果您使用高可用性模式，请选择两个或更多子网。如果您使用独立模式和 AWS Blu Age 引擎，请选择两个或更多子网。如果使用独立模式和 Micro Focus 引擎，则可以指定一个子网。
   d. 为所选的 VPC 选择安全组。

> **Note**
>
> AWS Mainframe Modernization 为您创建了一个网络负载均衡器，用于将连接分发到您的运行时环境。请确保您的安全组入站规则允许从 IP 地址访问您在应用程序定义的 `listener` 属性中指定的端口。有关更多信息，请参阅《网络负载均衡器用户指南》中的注册目标。

e. 如果要使用客户托管的 AWS KMS key 密钥，在 KMS 密钥字段中，选择自定义加密设置。有关更多信息，请参阅 AWS 大型机现代化服务的静态数据加密。

> **Note**
>
> 默认情况下，AWS Mainframe Modernization 使用 AWS Mainframe Modernization 为您保留和管理的 AWS KMS key来加密您的数据。不过，您可以选择使用客户托管的 AWS KMS key。

f. （可选）按名称或 Amazon 资源名称（ARN）选择 AWS KMS key。或者，选择创建 AWS KMS key，进入 AWS KMS 控制台并创建一个新的 AWS KMS key。

g. 选择下一步。

7. （可选）在附加存储页面上，选择一个或多个 Amazon EFS 或 Amazon FSx 文件系统，然后选择下一步。

8. 在维护时段部分，选择何时将待处理的更改应用于环境。

   - 如果选择无首选项，则 AWS Mainframe Modernization 将为您选择优化的维护时段。
   - 如果要指定特定的维护时段，请选择选择新的维护时段。然后，选择维护时段的开始日期、开始时间和持续时间。

   有关维护时段的更多信息，请参阅 AWS Mainframe Modernization 维护时段。

   选择下一步。

9. 在审核和创建页面中，检查您输入的信息，然后选择创建环境。

更新 AWS Mainframe Modernization 运行时环境

使用 AWS Mainframe Modernization 控制台更新 AWS Mainframe Modernization 运行时环境。您可以更新运行时引擎的次要版本或托管运行时环境的实例类型。您可以选择是要立即应用更新，还是在首选维护时段内应用更新。

以下说明假定您已完成设置 AWS 大型机现代化 中的步骤。

更新运行时环境

更新运行时环境

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要更新的环境创建时所选的区域。
3. 在环境页面上，选择要更新的环境。
4. 在该环境的详细信息页面上，选择操作，然后选择编辑环境。
5. 进行以下任何更改：
   - 在引擎选项部分，选择所需的引擎版本。
   - 在资源部分，选择所需的实例类型。
   - 在维护时段 部分，选择所需的日期、时间和持续时间。

   > **Note**
   >
   > 您可以选择在维护时段内应用的唯一更改是对引擎版本的更改。因此，必须立即应用所有其他更改。

6. 选择下一步。
7. 在何时应用更改中，选择立即或在下一个维护时段内。然后，选择更新环境。

如果选择立即，则当环境完成更新时，您会看到一条消息。

AWS Mainframe Modernization 维护时段

每个运行时环境每周都有一小时的维护时段。在此时段内，会应用所有系统更改。您可以利用维护时段控制何时进行修改和软件修补。如果在指定周内计划了维护事件，则维护会在一小时的维护时段内启

动。大部分维护事件也会在一小时的维护时段内完成，但较大的维护事件可能需要一小时以上的时间才能完成。

一小时维护时段是从每个区域的 8 小时时间段中随机选择的。如果创建运行时环境时未指定维护时段，则 AWS Mainframe Modernization 将在随机选择的一星期的某一天中分配 1 小时维护时段。

AWS Mainframe Modernization 在应用维护时，会使用环境实例中的部分资源。在维护过程中，您可能会观察到对性能的影响甚微，也可能会看到应用程序中断。

下表显示了为每个区域分配维护时段时的默认时间段。

区域名称	区域	时间段
美国东部（弗吉尼亚州北部）	us-east-1	03:00–11:00 UTC
美国西部（俄勒冈州）	us-west-2	06:00–14:00 UTC
亚太地区（孟买）	ap-south-1	06:00–14:00 UTC
亚太地区（新加坡）	ap-southeast-1	14:00–22:00 UTC
亚太地区（悉尼）	ap-southeast-2	12:00–20:00 UTC
亚太地区（东京)	ap-northeast-1	13:00–21:00 UTC
加拿大（中部）	ca-central-1	03:00–11:00 UTC
欧洲地区（法兰克福）	eu-central-1	21:00–05:00 UTC
欧洲地区（爱尔兰）	eu-west-1	22:00–06:00 UTC
欧洲地区（伦敦）	eu-west-2	22:00–06:00 UTC
欧洲地区（巴黎）	eu-west-3	23:59–07:29 UTC
南美洲（圣保罗）	sa-east-1	00:00–08:00 UTC

停止 AWS Mainframe Modernization 运行时环境

使用 AWS Mainframe Modernization 控制台停止 AWS Mainframe Modernization 运行时环境。停止环境时，当前的应用程序部署将保留，并且在环境重新启动之前，您无需支付该环境的费用。

以下说明假定您已完成设置 AWS 大型机现代化 中的步骤。

停止运行时环境

如果您需要停止 AWS Mainframe Modernization 运行时环境，请按照与更新环境部分类似的步骤进行操作。

使用 AWS Mainframe Modernization 控制台停止 AWS Mainframe Modernization 运行时环境。停止环境时，当前的应用程序部署将保留，并且在环境重新启动之前，您无需支付该环境的费用。

停止运行时环境

要停止 AWS Mainframe Modernization 运行时环境，请按照与更新环境部分类似的步骤进行操作。

> **Note**
>
> 必须先停止所有应用程序，然后才能停止环境。

停止运行时环境

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要停止的环境创建时所选的区域。
3. 在环境页面上，选择要停止的环境。
4. 在该环境的详细信息页面上，选择操作，然后选择编辑环境。
5. 在编辑环境页面上，找到资源部分，然后将所需容量更新为零。

   > **Note**
   >
   > 要停止环境，仅能选择立即停止。

6. 选择下一步。

7. 在何时应用更改中，选择立即。然后，选择更新环境。

环境容量更新后，您会看到一条消息。

重新启动 AWS Mainframe Modernization 运行时环境

使用 AWS Mainframe Modernization 控制台重新启动 AWS Mainframe Modernization 运行时环境。重新启动运行时环境时，将恢复对该环境的计费。

重新启动运行时环境

要重新启动 AWS Mainframe Modernization 运行时环境，请按照与停止环境部分类似的步骤进行操作。

重新启动运行时环境

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要重新启动的环境创建时所选的区域。
3. 在环境页面上，选择要重新启动的环境。
4. 在该环境的详细信息页面上，选择操作，然后选择编辑环境。

> **Note**
>
> 独立环境的所需容量仅能更新为 1。要重新启动运行时环境，仅能选择立即重新启动。

5. 在编辑环境页面上，找到资源部分，然后将所需容量从零更新为所需值。
6. 选择下一步。
7. 在何时应用更改中，选择立即。然后，选择更新环境。

环境容量更新完成并且环境重新启动后，您会看到一条消息。

删除 AWS Mainframe Modernization 运行时环境

使用 AWS Mainframe Modernization 控制台删除 AWS Mainframe Modernization 运行时环境。

以下说明假定您已完成设置 AWS 大型机现代化 中的步骤。

删除运行时环境

如果您需要删除 AWS Mainframe Modernization 运行时环境，请务必先从该环境中删除所有已部署的应用程序。您无法删除已部署应用程序的运行时环境。

删除环境

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域选择器中，选择要删除的环境创建时所选的区域。
3. 在环境页面上，选择要删除的环境，然后依次选择操作和删除环境。
4. 在删除环境窗口中，输入 `delete` 以确认要删除该运行时环境，然后选择删除。

AWS 大型机现代化中的应用程序测试

AWS 应用程序测试处于 AWS 大型机现代化的预览版，可能会发生变化。建议您仅将此功能用于测试数据和应用程序，而不要在生产环境中使用。

AWS 大型机现代化应用程序测试为您的迁移项目提供自动功能等效性测试。

主题

什么是 AWS Mainframe Modernization 应用程序测试？

AWS 应用程序测试包含在 AWS Mainframe Modernization 的预览版中，可能会发生变化。建议您仅将此功能用于测试数据和应用程序，而不要在生产环境中使用。

测试会显著影响迁移项目。它会消耗高达 70% 的迁移、现代化或增强项目时间和精力。AWS应用程序测试是 AWS Mainframe Modernization 的一项功能，为迁移的应用程序提供自动化的功能等效性测试。功能等效性测试帮助您验证 AWS Cloud 上的应用程序是否与大型机上的应用程序等效。AWS 应用程序测试自动比较大型机与 AWS 之间数据集、数据库记录和在线 3270 屏幕的变化。此外，应用程序测试允许可重复的测试，因此，当您更新目标架构、解决问题以及向完全迁移的应用程序迈进时，可以多次运行测试场景。迁移后，您可以继续使用应用程序测试进行回归测试，以确保对运行时引擎或其他组件的更新不会导致回归。应用程序测试具有成本效益：目标测试环境是使用用户提供的 CloudFormation 模板创建的，利用了基础设施即代码（IaC）概念。应用程序测试利用云弹性加快了迁移项目的速度。您可以根据需要在多个并行环境中运行独立的测试场景，从而缩短测试时间。

主题

您是首次使用应用程序测试吗？

如果您是首次使用应用程序测试，建议您先阅读以下章节：

应用程序测试的好处

应用程序测试具有多种好处，可为您的迁移过程提供帮助：

- 测试加速、敏捷性和灵活性
- “在大型机上记录一次，即可在 AWS 中多次重放”的测试概念
- 通过用户提供的 CloudFormation 模板进行目标环境的 IaC 创建
- 测试可重复性高
- 专为云而构建，考虑了可扩展性和弹性
- 自动化程度高的大规模测试
- 成本效益

与 AWS CloudFormation 集成

应用程序测试将基础设施即代码与 AWS CloudFormation 结合起来。这种设计选择简化并改善了您的测试体验。AWS CloudFormation 使您能够自主、独立地根据需求定义更好的基础设施。您可以为许多参数（实例大小、RDS 实例、最佳安全组）独立选择或定义。您可以添加资源，例如添加使您的应用程序在测试条件下正常工作所需要的 Amazon SQS 队列。

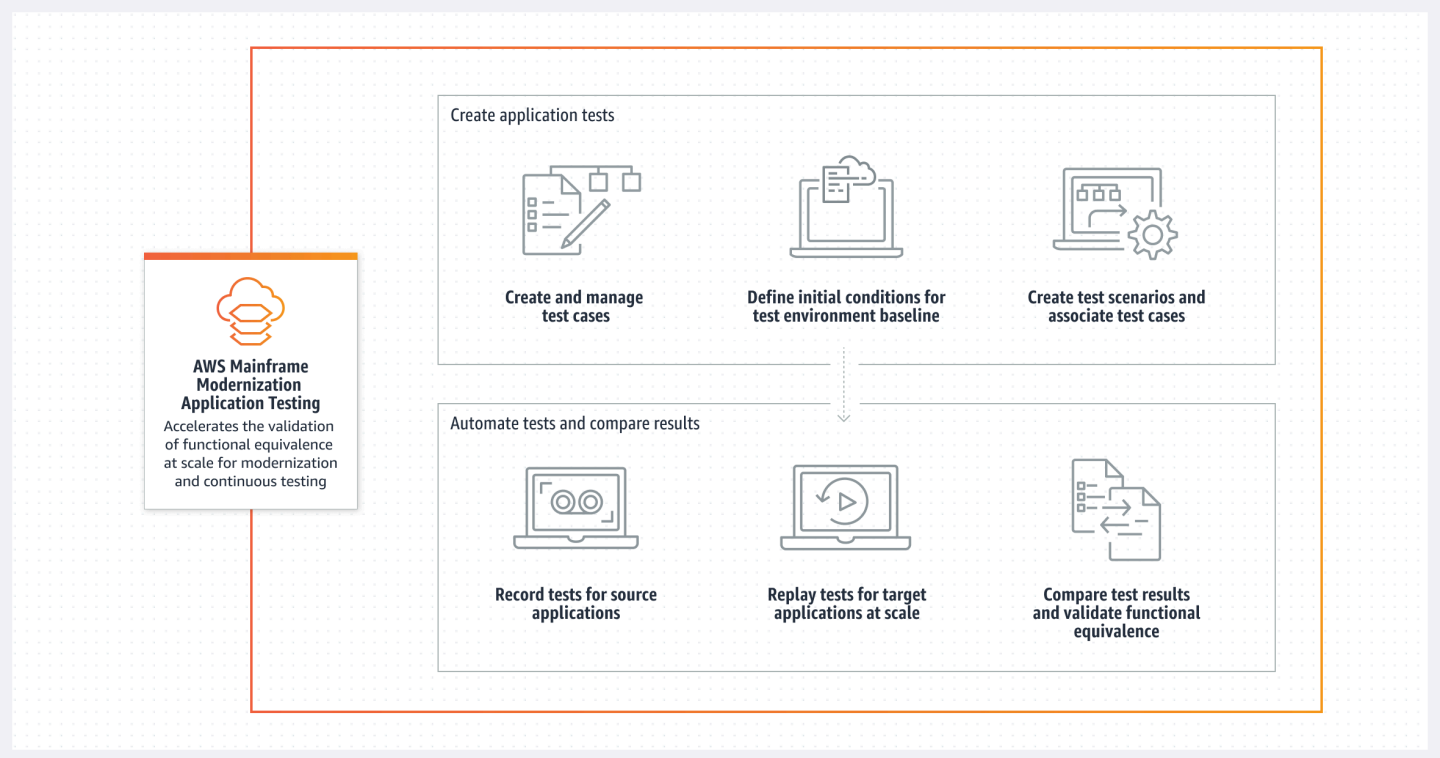
在供下载的 AWS CloudFormation 模板中，您会注意到一些常见功能：

- 应用程序测试创建一个完全隔离的堆栈，包括 AWS Mainframe Modernization 运行时环境和应用程序，并具有它自己的网络和安全定义。此隔离堆栈可提供弹性，因为同一个 AWS 账户中的其他参与者无法干扰测试活动。它还可以避免发生系统操作员修改默认 VPC 或安全组而导致测试活动故障的情况。
- 安全组还允许您控制外部访问测试中使用的资源。例如，数据库可能包含机密数据。
- 完全隔离则能防止共享 VPC 的其他参与者窥探流量。
- 它能增强性能。例如，模板创建的 AWS Mainframe Modernization 应用程序与其 Amazon RDS 数据库之间的通信发生在单独的网络（私有 VPC）上，这将避免其他参与者减慢流量。

建议您在创建的 AWS CloudFormation 模板中也实施这些功能。

应用程序测试的工作原理

下图概述了应用程序测试中的功能等效性测试的工作原理。



- - 您可以使用 AWS Mainframe Modernization 文件传输或您偏爱的大型机数据传输工具将输入数据从源传输到 AWS。
- 您在源和目标上运行相同的业务逻辑。
- 应用程序测试自动比较来自源和目标的输出数据（数据集、关系数据库更改、3270 屏幕和用户交互）。当您在大型机上运行测试场景后，可以捕获输出数据并将其传输到 AWS，然后在目标上重放

测试场景。应用程序测试会自动比较来自 AWS 上运行的测试的输出数据与来自源的输出数据。您可以一眼看出哪些记录相同、相等、不同或缺失。此外，您可以定义等效规则，让那些虽不完全相同但具有相同业务含义的记录被理解为等效。

您在应用程序测试中执行的工作流程包括以下步骤：

1. 创建测试用例。测试用例是测试操作的最小单位。当您创建一个测试用例时，也确定了要比较的数据类型，这些数据类型最能代表源和目标之间的功能等效性。
2. 创建测试场景。测试场景将相关测试用例分组到一个特定序列中来运行。
3. 创建初始条件。初始条件解释了如何在大型机上记录测试，并使用 AWS CloudFormation 在 AWS 上自动创建相同的状态。
4. 在源上记录以及在目标上重放。捕获大型机上的输入和输出数据集，并将它们上传到 AWS。然后在 AWS 上重放测试场景。
5. 比较源和目标数据集。应用程序测试会自动比较来自源和目标的输出数据集，因此，您可以一眼看出哪些是正确的，哪些是不正确的。

测试场景的最终操作和整个过程的目的都是确定源和目标测试运行之间的差异。应用程序测试会比较测试运行期间在所有交互渠道上捕获的数据的源版本和目标版本。它还会比较相关数据的最终状态（如测试用例中所定义的）。

相关服务

应用程序测试是 AWS Mainframe Modernization 的一项功能。它也将基础设施即代码与 AWS CloudFormation 结合使用来确保测试的可重复性、自动化和成本效益。有关更多信息，请参阅：

- AWS Mainframe Modernization
- AWS CloudFormation

访问应用程序测试

您可以通过在左侧导航栏中选择应用程序测试，从 AWS Mainframe Modernization 控制台访问应用程序测试。

应用程序测试的定价

可以在 AWS Mainframe Modernization 定价中找到应用程序测试的定价。

AWS 大型机现代化应用程序测试概念

AWS 应用程序测试处于 AWS 大型机现代化的预览版，可能会发生变化。建议您仅将此功能用于测试数据和应用程序，而不要在生产环境中使用。

AWS 应用程序测试使用的术语与其他测试服务或软件包可能使用的术语含义略有不同。以下各节说明了 AWS 大型机现代化应用程序测试如何使用此术语。

主题

- 源应用程序
- 目标应用程序

测试用例

一个测试用例就是测试工作流程中一个最具原子性的操作单元。通常，一个测试用例用于表示修改数据的一个独立业务逻辑单元。将对每个测试用例进行比较。测试用例添加到测试场景中。测试用例包含有关测试用例修改的数据构件（数据集、数据库）以及测试用例执行期间触发的业务功能（批处理任务、3270 交互式对话等）的元数据。例如，数据集的名称和代码页。

输入数据 → 测试用例 → 输出数据

测试用例可以是在线类型，或者是批处理类型：

- 在线测试用例是用户执行交互式屏幕对话（3270）来读取、修改或生成新的业务数据（数据库和/或数据集记录）的测试用例。
- 批处理测试用例是要求提交批处理来读取、处理、修改或生成新的业务数据（数据集和/或数据库记录）的测试用例。

测试场景

测试场景是一系列按顺序逐一运行的测试用例。重放是在测试场景级别完成的。当重放测试场景时，测试场景中的所有测试用例都在目标测试环境中运行。如果比较参考测试构件和重放测试构件后存在差异，则会在测试用例级别显示差异。

例如，测试场景 A：

测试用例 1、测试用例 2、测试用例 3 等。

测试项目

测试项目代表达成所需测试里程碑的测试场景的集合。例如，可以将迁移一个特定应用程序视为单个测试项目。测试场景分组为测试项目将允许测试经理跟踪测试项目状态，包括通过/失败的测试。

初始条件

初始条件包含一组必须创建的资源（计算、数据存储等），以及运行测试场景之前必须在这些创建的资源上恢复的应用程序数据。这将创建目标测试环境基准。它允许您提供 AWS CloudFormation 模板。

您可以使用该模板创建目标测试环境，并且，如果您的测试场景修改了数据库记录，还可以使用该模板创建源数据库的 DDL 摘录。每个测试场景都将与一个初始条件相关联。一个初始条件可以关联到多个测试场景。为确保结果一致的可重复性，以及为避免由于已经更改的数据而导致的误报，您必须在运行每个测试场景之前恢复初始条件。

对于包含修改数据库记录的测试用例的测试场景，初始条件还引用源数据库架构和表的 DDL 导出。

记录（捕获）

记录是在测试场景级别完成的。在记录过程中，您必须提供一个包含来自源大型机的构件、数据集和关系数据库 CDC 日志的 Amazon S3 位置，以便用于比较。这些内容将被视为来自源大型机的参考数据。在重放期间，生成的重放数据将与记录的参考数据进行比较，以确保应用程序的等效性。

重放

重放是在测试场景级别上完成的。在重播期间， AWS 大型机现代化应用程序测试使用关联的初始条件中引用的 CloudFormation脚本来创建目标测试环境并运行应用程序。系统将捕获在重放期间修改的数据集和数据库记录，并与来自大型机的参考数据进行比较。通常，您将在大型机上记录一次，然后重放多次，直到达到功能等效为止。

比较

成功完成重放后，系统会自动进行比较。在比较过程中，会将您在记录阶段上传和捕获的参考数据与重放阶段生成的重放数据进行比较。将在单个测试用例级别上为数据集、数据库记录和在线屏幕独立完成比较。

数据库比较

在比较源和目标应用程序之间数据库记录的变化时，应用程序测试利用状态进度匹配功能。状态进度匹配功能比较每条运行的 INSERT、UPDATE 和 DELETE 语句的差异，这与在过程结束时比较表行不同。状态进度匹配功能比其他方法更高效，它仅比较更改的数据并检测事务流中的自我更正错误，因此速度更快，也更准确。通过使用 CDC（更改数据捕获）技术，应用程序测试可以检测单个关系数据库的变化，并在源和目标之间进行比较。

关系数据库更改由经过测试的应用程序代码使用 DML（数据修改语言）语句（如 SQL INSERT、UPDATE 或 DELETE）在源和目标上生成，不仅如此，当应用程序使用存储过程，或在某些表上设置了数据库触发器，或当使用 CASCADE DELETE 来保证引用完整性，从而自动触发额外删除时，也能间接生成关系数据库更改。

数据集比较

应用程序测试会自动比较在源（记录）和目标（重放）系统上生成的参考数据集和重放数据集。

比较数据集

1. 从源和目标上相同的输入数据（数据集、数据库）开始。
2. 在源系统（大型机）上运行您的测试用例。
3. 捕获生成的数据集并将其上传到 Amazon S3 存储桶。您可以将输入数据集从源传输到 AWS 使用 CDC 日记、屏幕和数据集。
4. 指定记录测试用例时上传大型机数据集的 Amazon S3 存储桶的位置。

重放完成后，应用程序测试会自动比较输出参考和目标数据集，显示记录是相同、等效、不同还是缺失。例如，与工作负载执行时刻有关的日期字段（日期 + 1、当月结束等）会自动被视为等效。此外，您还可以选择定义等效规则，以便那些尽管不是完全相同但仍具有相同业务含义的记录被标记为等效。

比较状态

应用程序测试使用以下比较状态：相同、等效和不同。

相同

源数据和目标数据完全相同。

等效

源数据和目标数据包含被视为等效的虚假差异，例如与工作负载执行时刻有关但不影响功能等效性的日期或时间戳。您可以定义等效规则来确定这些差异是什么。当所有重放的测试场景与其参考测试场景比较后显示状态为“相同”或“等效”时，证明您的测试场景功能等效。

不同

源数据和目标数据包含差异，例如数据集中的记录数量不同或同一记录中的值不同。

等效规则

确定虚假差异可被视为等效结果的一组规则。离线功能等效性测试（OFET）不可避免地会导致源系统和目标系统之间某些结果存在差异。例如，更新时间戳因设计而异。等效规则解释了如何为这些差异做调整并避免在比较时出现误报。例如，如果在一个特定数据列中某个日期为运行时 + 2 天，则等效规则会描述它，并接受目标系统上一个运行时 + 2 天的时间，而不是严格等于参考记录中同一列的值。

最终状态数据集比较

已创建或修改的数据集的结束状态，包括从数据集初始状态起对数据集所做的所有更改或更新。对于数据集，应用程序测试会在测试用例运行结束时查看这些数据集中的记录，并比较结果。

状态进度数据库比较

系统会比较若干个 DML（删除、更新、插入）语句序列对数据库记录所做的更改。应用程序测试将比较从源数据库到目标数据库的各种更改（插入、更新或删除表的行），并将确定每个单独更改的差异。例如，一条 INSERT 语句可能被用来在表中插入一行，而源数据库相比目标数据库使用不同的值。

功能等效（FE）

如果两个系统被给予同样的输入数据，它们在所有可观察的操作中都产生同样的结果，则这两个系统被认为是功能等效的。例如，有两个应用程序，如果同样的输入数据生成相同的输出数据（通过屏幕、数据集更改或数据库更改），则这两个应用程序被视为功能等效。

在线 3270 屏幕比较

将目标系统在 AWS Blu Age 运行时下运行时，将大型机 3270 屏幕的输出与现代化应用程序网页屏幕的输出进行比较。 AWS Cloud当目标系统在 AWS Cloud中的 Micro Focus 运行时下运行时，比较大型机的 3270 屏幕输出与更换主机的应用程序的 3270 屏幕。

记录

恢复认可的数据状态，然后在源系统上捕获或记录参考测试场景（关于一个或多个连续测试用例）的参考数据的操作。

重放数据

重放数据用于描述在目标测试环境中重放测试场景时生成的数据。例如，重播数据是在 AWS 大型机现代化服务应用程序上运行测试场景时生成的。然后，将重放数据与从源捕获的参考数据进行比较。每次在目标环境中重放工作负载时，都会生成新一代的重放数据。

参考数据

参考数据用于描述在源大型机上捕获的数据。它是重放（目标）生成的数据要比较的参考。通常，对于大型机上创建参考数据的每条记录，都会有很多重放。这是因为，用户通常会在大型机上捕获应用程

序的正确状态，然后在目标现代化应用程序上重放测试用例，以验证等效性。如果发现错误，会进行修复，并再次重放测试用例。通常，要经历重放、修复错误、再次重放以验证的多次循环。这被称为一次捕获，多次重放的测试范式。

记录、重放和比较

应用程序测试执行三种步骤：

- 记录：捕获大型机上为测试场景的每个测试用例创建的参考数据。其中包括 3270 在线屏幕、数据集和数据库记录。
  - 对于在线的 3270 屏幕，您必须使用 Blu Insights 终端仿真器来捕获源工作负载。有关更多信息，请参阅 Blu Insights 文档。
  - 对于数据集，您需要使用常用工具（例如 FTP 或大型机现代化中的数据集传输服务部分）来捕获大型机上每个测试用例生成的数据集。 AWS
  - 对于数据库更改，请参阅使用 Precisely 进行 AWS Mainframe Modernization 数据复制文档，来捕获和生成包含更改的 CDC 日志。
- 重放：在目标环境中重放测试场景。测试场景中指定的所有测试用例都会运行。将自动捕获由各个测试用例创建的指定数据类型，例如数据集、关系数据库更改或 3270 屏幕。这些数据被称为重放数据，将与记录阶段捕获的参考数据进行比较。

> **Note**
>
> 关系数据库的更改需要在初始条件 CloudFormation 模板中提供特定于 DMS 的配置选项。

- 比较：比较源测试参考数据与目标重放数据，显示的结果为相同、不同、等效或缺失的数据。

差异

表示通过数据比较，检测到参考数据集和重放数据集之间存在差异。例如，在线 3270 屏幕中的一个字段显示，从业务逻辑角度看，源大型机和目标现代化应用程序之间的值不同，因此，该字段将被视为一个差异。又如，数据集中有一条记录，它在源和目标应用程序之间不相同。

等效

等效记录是参考数据集和重放数据集之间不同的记录，但从业务逻辑角度看，不应将其视为不同。例如，一个包含数据集生成时间（工作负载执行时间）的时间戳的记录。使用可自定义的等效规则，您可以指示应用程序测试将此类误报的不同视为等效，即使它显示参考数据和重放数据之间的值不同。

源应用程序

要比较的源大型机应用程序。

目标应用程序

新的或修改后的应用程序，在其上完成测试，并将与源应用程序进行比较，以检测任何缺陷并实现源应用程序和目标应用程序之间的功能等效。目标应用程序通常在 AWS 云端运行。

教程：设置 CardDemo 应用程序示例

> AWS 应用程序测试包含在 AWS Mainframe Modernization 的预览版中，可能会发生变化。建议您仅将此功能用于测试数据和应用程序，而不要在生产环境中使用。

在本教程中，您将创建一个 AWS CloudFormation 堆栈，它可以帮助您设置 CardDemo 应用程序示例，利用 AWS Mainframe Modernization 托管服务上的 Micro Focus 以及包括 AWS Mainframe Modernization 应用程序测试在内的功能实现平台更换。本教程介绍可用于创建堆栈的示例 AWS CloudFormation 模板。我们还提供了包含必要应用程序构件的压缩文件。示例模板预置了一个数据库、一个运行时环境、一个应用程序和一个完全隔离的网络环境。

该模板创建多个 AWS 资源。如果您从该模板创建堆栈，则需为资源付费。

先决条件

- 下载并解压缩 `IC3-card-demo-zip` 和 `datasets_Mainframe_ebcdic.zip`。这些文件包含用于 AWS 应用程序测试的 CardDemo 示例和示例数据集。
- 创建一个 Amazon S3 存储桶来存放 CardDemo 文件和其他构件。例如，`my-carddemo-bucket`。

步骤 1：准备设置 CardDemo

上传 CardDemo 示例文件并编辑将创建 CardDemo 应用程序的 AWS CloudFormation 模板。

1. 将您之前解压缩的 `datasets_Mainframe_ebcdic` 和 `IC3-card-demo` 文件夹上传到您的存储桶。
2. 从您的存储桶下载 `aws-m2-math-mf-carddemo.yaml` AWS CloudFormation 模板。它位于 `IC3-card-demo` 文件夹中。

3. 按如下所示编辑 aws-m2-math-mf-carddemo.yaml AWS CloudFormation 模板：

   - 将 BucketName 参数更改为您之前定义的存储桶的名称，例如 my-carddemo-bucket。
   - 将 ImportJsonPath 更改为 mf-carddemo-datasets-import.json 文件在存储桶中的位置。例如，s3://my-carddemo-bucket/IC3-card-demo/mf-carddemo-datasets-import.json。更新此值，确保输出 M2ImportJson 具有正确的值。
   - （可选）调整 EngineVersion 和 InstanceType 参数以符合您的标准。

   > **Note**
   >
   > 请勿修改 M2EnvironmentId 和 M2ApplicationId 输出。应用程序测试使用这些值来定位它将交互的资源。

步骤 2：创建所有必要的资源

运行您的自定义 AWS CloudFormation 模板，创建成功完成本教程所需的所有资源。此模板设置 CardDemo 应用程序，以便您可以在测试中使用它。

1. 登录 AWS CloudFormation 控制台，选择创建堆栈，然后选择使用新资源（标准）。
2. 在先决条件 – 准备模板中，选择模板已就绪。
3. 在指定模板中，选择上传模板文件，然后选择选择文件。
4. 导航到您下载 aws-m2-math-mf-carddemo.yaml 的地方并选择该文件，然后选择下一步。
5. 在指定堆栈详细信息中，提供堆栈的名称，以便可以轻松地在列表中找到它，然后选择下一步。
6. 在配置堆栈选项中，保持默认值，并选择下一步。
7. 在审核中，检查 AWS CloudFormation 正在为您创建的内容，然后选择提交。

AWS CloudFormation 创建堆栈约需 10–15 分钟时间。

> **Note**
>
> 模板设置为在其创建的资源名称后面附加一个唯一的后缀。这意味着您可以并行创建此堆栈模板的多个实例，这是应用程序测试的一项关键功能，允许您同时运行多个测试场景。

步骤 3：部署和启动应用程序

部署 AWS CloudFormation 为您创建的 CardDemo 应用程序并确保其正在运行。

1. 打开 AWS Mainframe Modernization 控制台，从左侧导航窗格选择应用程序。
2. 选择 CardDemo 应用程序，其名称类似于 `aws-m2-math-mf-carddemo-abc1d2e3`。
3. 选择操作，然后选择部署应用程序。
4. 在环境中，选择与应用程序对应的运行时环境。它将在名称末尾附加同样的唯一标识符。例如，`aws-m2-math-mf-carddemo-abc1d2e3`。
5. 选择部署。等待应用程序部署成功并处于“就绪”状态。
6. 选择应用程序，然后选择操作和启动应用程序。等待直到应用程序处于正在运行状态。
7. 在应用程序详细信息页面中，复制端口和 DNS 主机名，您需要它们以便连接到正在运行的应用程序。

步骤 4：导入初始数据

要使用 CardDemo 应用程序示例，必须导入一组初始数据。完成以下步骤。

1. 下载 `mf-carddemo-datasets-import.json` 文件。
2. 使用您常用的文本编辑器编辑该文件。
3. 找到 `s3Location` 参数并更新值，以指向您创建的 Amazon S3 存储桶。
4. 对所有出现的 `s3Location` 进行此同样的更改，然后保存文件。
5. 登录 Amazon S3 控制台并导航到您之前创建的存储桶。
6. 上传自定义 `mf-carddemo-datasets-import.json` 文件。
7. 打开 AWS Mainframe Modernization 控制台，从左侧导航窗格选择应用程序。
8. 选择 CardDemo 应用程序。
9. 选择数据集，然后选择导入。
10. 导航到 Amazon S3 中您上传自定义 JSON 文件的位置，然后选择提交。

此任务导入 23 个数据集。要监控导入任务的结果，请检查控制台。成功导入所有数据集后，连接到应用程序。

> **Note**
>
> 当您在应用程序测试中使用该模板时，输出 `M2ImportJson` 会自动处理导入过程。

步骤 5：连接到 CardDemo 应用程序

使用您选择的 3270 仿真器连接到 CardDemo 应用程序示例。

- 当该应用程序运行时，使用 3270 仿真器连接该应用程序，必要时指定 DNS 主机名和端口名。

  例如，如果您使用的是开源 c3270 仿真器，则命令如下所示：

  ```
  c3270 -port port-number DNS-hostname
  ```

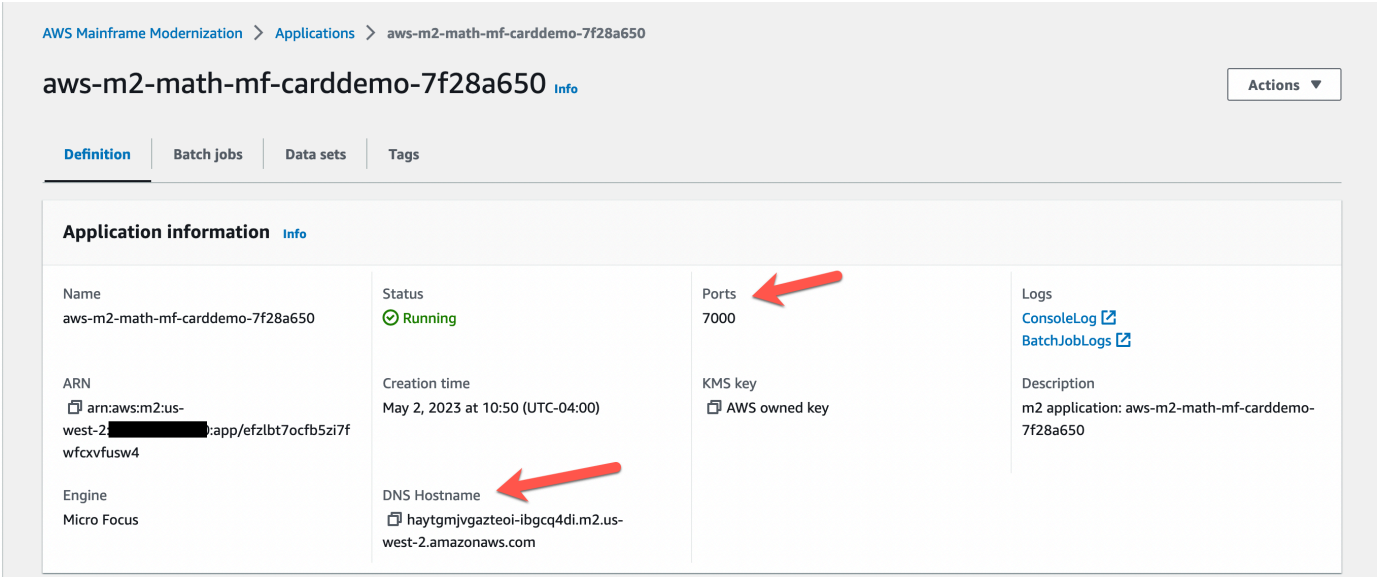
  port

  应用程序详细信息页面上指定的端口。例如，6000。

  Hostname

  应用程序详细信息页面上指定的 DNS 主机名。

  下图显示在哪里可以找到端口和 DSN 主机名。



教程：使用 CardDemo 部署在 A AWS mazon EC2 上的 AWS Blu Age 的大型机现代化应用程序测试重播和比较

AWS 应用程序测试包含在 AWS Mainframe Modernization 的预览版中，可能会发生变化。建议您仅将此功能用于测试数据和应用程序，而不要在生产环境中使用。

在本教程中，您将完成必要的步骤，将测试工作负载与部署在 Amazon EC2 上的 AWS Blu Age 上运行的 CardDemo 应用程序进行重放和比较。

第 1 步：获取 AWS Blu Age 亚马逊 EC2 亚马逊机器映像 (AMI)

按照 AWSAWSBlu Age Runtime（在 Amazon EC2 上）安装教程中的说明进行操作，了解在 Amazon EC2 AMI 上访问 AWS Blu Age 所需的入门步骤。

第 2 步：使用 AWS Blu Age AMI 启动亚马逊 EC2 实例

1. 设置您的 AWS 凭证。
2. 确定 Amazon S3 存储桶中 3.5.0 Amazon EC2 AMI 二进制文件（仅限 CLI/AWSBlu Age 版本）的位置：

```
aws s3 ls s3://aws-bluage-runtime-artifacts-xxxxxxx-eu-west-1/
aws s3 ls s3://aws-bluage-runtime-artifacts-xxxxxxx-eu-west-1/3.5.0/AMI/
```

> **Note**
>
> 应用程序测试功能仅在产品中的以下 4 个区域可用：us-east-1、sa-east-1、eu-central-1 和 ap-southeast-2。

3. 使用以下命令，恢复账户中的 AMI：

```
aws ec2 create-restore-image-task --object-key 3.5.0/AMI/ami-0182ffe3b9d63925b.bin
 --bucket aws-bluage-runtime-artifacts-xxxxxxx-eu-west-1 --region eu-west-1 --name
 "AWS BLUAGE RUNTIME AMI"
```

> **Note**
>
> 替换 AMI bin 文件名称以及要在其中创建 AMI 的区域。

4. 创建 Amazon EC2 实例后，您可以在 Amazon EC2 映像目录中找到从 Amazon S3 存储桶恢复的 AMI 的正确 AMI ID。

> **Note**
>
> 在本教程中，AMI ID 为 ami-0d0fafcc636fd1e6d，您必须将不同配置文件中的这一 ID 改为提供给您的值。

1. 如果 a ws ec2 create-restore-image-task 出现故障，请使用以下命令检查您的 Python 和 CLI 版本：

```
aws --version
```

> **Note**
>
> Python 版本必须为 >= 3，CLI 版本必须为 >= 2。

2. 如果这些版本已过时，则必须更新 CLI。要更新 CLI，执行以下步骤：
   a. 按照安装或更新 AWS CLI 的最新版本中的指示操作。
   b. 使用以下命令删除 CLI v1：

```
sudo yum remove awscli
```

   c. 使用以下命令安装 CLI v2：

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
 "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

   d. 最后，使用以下命令检查 Python 和 CLI 的版本：

```
aws --version
```

3. 然后你可以重做 aw s ec2 create-restore-image-task。

步骤 3：将 CardDemo 依赖文件上传到 S3

复制文件夹 databases、file-system 和 userdata 的内容。下载并解压缩 CardDemo 应用程序。这三个文件夹都必须复制到本文档中称为 your-s3-bucket 的存储桶之一中。

步骤 4：加载数据库并初始化 CardDemo应用程序

创建一个临时 Amazon EC2 实例，该实例将用作计算资源，为 CardDemo应用程序生成所需的数据库快照。此 EC2 实例不会运行 CardDemo 应用程序本身，而是生成数据库快照供以后使用。

首先编辑提供的名为 “load-and-create-ba-snapshots.yml” 的 CloudFormation 模板。此 CloudFormation 模板用于创建用于生成数据库快照的 Amazon EC2 实例。

1. 生成并提供将用于 EC2 实例的 EC2 密钥对。有关更多信息，请参阅创建密钥对。

   例如：

```
Ec2KeyPair:
    Description: 'ec2 key pair'
    Default: 'm2-tests-us-west-2'
    Type: String
```

2. 指定上一步中存放 database 文件夹的文件夹的 Amazon S3 路径：

```
S3DBScriptsPath:
    Description: 'S3 DB scripts folder path'
    Type: String
    Default: 's3://your-s3-bucket/databases'
```

3. 指定上一步中存放 file-system 文件夹的文件夹的 Amazon S3 路径：

```
S3ApplicationFilesPath:
    Description: 'S3 application files folder path'
    Type: String
    Default: 's3://your-s3-bucket/file-system'
```

4. 指定上一步中存放 userdata 文件夹的文件夹的 Amazon S3 路径：

```
S3UserDataPath:
    Description: 'S3 userdata folder path'
    Type: String
    Default: 's3://your-s3-bucket/userdata'
```

5. 另外，指定一个用来保存结果文件的 Amazon S3 路径，后续步骤中将用到结果文件。

```
S3SaveProducedFilesPath:
    Description: 'S3 path folder to save produced files'
    Type: String
    Default: 's3://your-s3-bucket/post-produced-files'
```

6. 使用以下模板将 AMI ID 更改为在本教程前面部分中获得的正确 ID：

```
  BaaAmiId:
    Description: 'ami id (AL2) for ba anywhere'
    Default: 'ami-0bd41245734fd20d9'
    Type: String
```

   - 您可以选择更改将在运行加载数据库时创建的三个快照的名称 withCloudFormation。它们将在堆栈创建时在 CloudFormation 堆栈中可见，并将在本教程的后面部分使用。记着记下用于数据库快照的名称。

```
SnapshotPrimary:
    Description: 'Snapshot Name DB BA Primary'
    Type: String
    Default: 'snapshot-primary'

  SnapshotBluesam:
    Description: 'Snapshot Name DB BA Bluesam'
    Type: String
    Default: 'snapshot-bluesam'

  SnapshotJics:
    Description: 'Snapshot Name DB BA Jics'
    Type: String
    Default: 'snapshot-jics'
```

> **Note**
>
> 在本文档中，我们假定快照的名称保持一致。

7. 使用 “ CloudFormation 创建堆栈” 按钮和向导通过 CLI 或AWS控制台运行。在该过程结束时，您会在 RDS 控制台中看到三个快照，使用了您选择的名称，后面跟一个唯一 ID。在下一步骤中将需要这些名称。

> **Note**
>
> RDS 将在 AWS CloudFormation 模板中定义的快照名称后添加后缀。在继续下一步之前，请务必从 RDS 获取完整快照名称。

示例 CLI 命令

```
aws cloudformation create-stack --stack-name load-and-create-ba-snapshots --
template-url https://your-apptest-bucket.s3.us-west-2.amazonaws.com/load-and-
create-ba-snapshots.yml --capabilities CAPABILITY_NAMED_IAM
```

您还可以在为 S3 提供的 Amazon S 3 SaveProducedFilesPath 路径中检查数据集是否已正确创建。

第 5 步：启动 AWS Blu Age 运行时 CloudFormation

CloudFormation 用于运行带有 CardDemo AWS Blu Age 应用程序的 Amazon EC2 实例。您必须`m2-with-ba-using-snapshots-https-authentication.yml`通过编辑 YAML 文件或在 CFN 启动期间在控制台中修改值来替换 CloudFormation 命名中的某些变量。

1. 使用以下命令修改`AllowedVpcEndpointPrincipals`以指定哪个账户将到达用于访问 AWS Blu Age 运行时的 VPC 终端节点：

```
AllowedVpcEndpointPrincipals:
    Description: 'comma-separated list of IAM users, IAM roles, or AWS accounts'
    Default: 'apptest.amazonaws.com'
    Type: String
```

2. 将变量 SnapshotPrimaryDb SnapshotBlusamDb、和的值更改 SnapshotJicsDb 为快照的名称。上一步中创建了快照后，还可从 RDS 获取快照名称。

```
SnapshotPrimary:
    Description: 'Snapshot DB cluster for DB Primary'
    Type: String
    Default: 'snapshot-primary87d067b0'

  SnapshotBluesam:
    Description: 'Snapshot DB cluster for DB Bluesam'
    Type: String
    Default: 'snapshot-bluesam87d067b0'

  SnapshotJics:
    Description: 'Snapshot DB cluster for DB Jics'
    Type: String
    Default: 'snapshot-jics87d067b0'
```

> **Note**
>
> RDS 将在快照名称后添加自己的后缀。

3. 使用以下命令为 EC2 实例提供您的 Amazon EC2 密钥对：

```
Ec2KeyPair:
    Description: 'ec2 key pair'
    Default: 'm2-tests-us-west-2'
    Type: String
```

4. 使用以下方法提供您在 AMI 注册过程中为变量BaaAmiId获取的 AMI ID：

```
BaaAmiId:
    Description: 'ami id (AL2) for ba anywhere'
    Default: 'ami-0d0fafcc636fd1e6d'
    Type: String
```

5. 使用以下命令，提供您在上一步中用于保存生成的文件的 Amazon S3 文件夹路径：

```
S3ApplicationFilesPath:
    Description: 'bucket name'
    Type: String
```

```
    Default: 's3://your-s3-bucket/post-produced-files'
```

6. 最后，提供 s 3- 的文件夹路径userdata-folder-path：

```
S3UserDataPath:
    Description: 'S3 userdata folder path'
    Type: String
    Default: 's3://your-s3-bucket/userdata'
```

- （可选）您可以为 tomcat 启用 HTTPS 模式和基本 HTTP 身份验证。尽管默认设置也可以使用。

> **Note**
>
> 默认情况下，HTTPS 模式处于禁用状态，并在参数中设置为 HTTP 模式 BacHttpsMode：

例如：

```
BacHttpsMode:
    Description: 'http or https for Blue Age Runtime connection mode '
    Default: 'http'
    Type: String
    AllowedValues: [http, https]
```

- （可选）要启用 HTTPS 模式，您必须将该值更改为 HTTPS，并通过更改变量 ACM 的值来提供 ACM 证书 ARN：CertArn

```
ACMCertArn:
    Type: String
    Description: 'ACM certificate ARN'
    Default: 'your arn certificate'
```

- （可选）默认情况下，基本身份验证处于禁用状态，参数WithBacBasicAuthentication设置为 false。您可以通过将该值设置为 true 来启用它。

```
WithBacBasicAuthentication:
    Description: 'false or true for Blue Age Runtime Basic Authentication '
    Default: false
    Type: String
```

```
    AllowedValues: [true, false]
```

7. 完成配置后，您可以使用编辑后的 CloudFormation 模板创建堆栈。

第 6 步：测试 AWS Blu Age 亚马逊 EC2 实例

手动运行 CloudFormation 模板为 CardDemo 应用程序创建 AWS Blu Age Amazon EC2 实例，以确保其启动时没有错误。这样做是为了在使用带有应用程序测试功能的 CloudFormation 模板之前，验证 CloudFormation 模板和所有先决条件是否有效。然后，您可以使用应用程序测试在重播期间自动创建目标 AWS Blu Age Amazon EC2 实例，并通过初始条件进行比较。

1. 运行 CloudFormation 创建堆栈命令创建 AWS Blu Age Amazon EC2 实例，提供您在上一步中编辑的 m2-with-ba-using-snapshots-https-authentication.yml 模板 CloudFormation ：

```
aws cloudformation create-stack --stack-name load-and-create-ba-snapshots —-
template-url https://apptest-ba-demo.s3.us-west-2.amazonaws.com/m2-with-ba-using-
snapshots-https-authentication.yml --capabilities CAPABILITY_NAMED_IAM --region us-
west-2
```

> **Note**
>
> 请记住指定恢复 AWS Blu Age AMI 的正确区域。

2. 确保一切工作是否正常，方法是，查看控制台，找到正在运行的 Amazon EC2 实例。使用 Session Manager 连接它。
3. 连接到 Amazon EC2 实例后，使用以下命令：

```
sudo su
cd /m2-anywhere/tomcat.gapwalk/velocity/logs
cat catalina.log
```

4. 确保日志中没有异常或错误。
5. 接下来，使用该命令检查应用程序是否正在响应：

```
curl http://localhost:8080/gapwalk-application/
```

您会看到消息：“Jics 应用程序正在运行。”

步骤 7：验证之前的步骤是否正确完成

在接下来的几个步骤中，我们将使用AWS大型机现代化应用程序测试来重放和比较 CardDemo 应用程序创建的数据集。这些步骤依赖于成功完成本教程中所有前面的步骤。在继续之前，先验证以下事项：

1. 您已通过AWS CloudFormation模板成功创建了 Amazon EC2 上的 AWS Blu Age 实例。
2. 亚马逊 EC2 上的 AWS Blu Age 上的 Tomcat 服务已启动并运行，无一例外。

当您让 EC2 实例与应用程序一起运行时，请在 CardDemo 应用程序测试控制台上完成以下步骤，对批处理数据集执行重放和比较。

步骤 8：创建初始条件

在此步骤中，您将通过提供用于在 Amazon EC2 上部署 AWS Blu Age CardDemo 应用程序的 CloudFormation 模板来创建初始条件。

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在左侧导航窗格中，选择应用程序测试。
3. 在应用程序测试中，选择创建初始条件。
4. 使用包含修改后的 Amazon S3 路径和指向您的资源的快照 ID 的本地副本。

步骤 9：创建测试用例

在此步骤中，您将创建用于比较在 Card Demo 应用程序中创建的数据集的测试用例。

1. 创建新的测试用例。给出名称和描述。
2. 指定 `CREASTMT.JCL` 为 JCL 名称。
3. 将以下数据集添加到测试用例定义中：

名称	CCSID	RecordFormat	RecordLength
AWS.M2.CARDDEMO.STATEMNT.PS	"037"	FB	80

AWS.M2.CA RDDEMO.ST ATEMNT.HTML	"037"	FB	100

> **Note**
>
> 您的 JCL 名称和数据集详细信息必须匹配。

步骤 10：创建测试场景

1. 创建一个新的测试场景，并为其提供名称和描述。
2. 将您在上一步中创建的测试用例添加到测试场景中。
3. 创建测试场景后，在测试场景概述页面中选择步骤 1 中创建的初始条件。

步骤 11：记录您的测试场景

在此步骤中，您将在源上运行测试用例。为此，请执行以下步骤：

1. 下载并运行源自 CardDemo 应用程序大型机运行的数据集。
2. 将解压后的文件夹上传到 Amazon S3 存储桶。此 Amazon S3 存储桶必须与您的其他应用程序测试资源位于同一区域。

> **Note**
>
> 应该有两个文件，它们的名称与前面测试用例中传递的数据集名称相匹配。

3. 在测试场景概述页面上，选择记录按钮。
4. 在测试场景记录页面上，指定您将获取自源大型机的数据集上传到的 Amazon S3 位置。
5. 单击提交开始记录流程。

> **Note**
>
> 等待记录完成后再执行重放和比较。

步骤 12：重放和比较

在 Amazon EC2 上的目标 AWS AWS Blu Age 环境中运行测试场景和测试用例。应用程序测试将捕获重放生成的数据集，并将它们与大型机上记录的参考数据集进行比较。

1. 选择重放和比较。创建 CloudFormation 堆栈、执行比较和删除堆栈大约需要三分钟。

所有工作都完成后，您将得到比较结果，其中包含为本演示目的特意创建的一些差异。

AWS 大型机现代化应用程序测试支持的数据集代码页

AWS 应用程序测试处于 AWS 大型机现代化的预览版，可能会发生变化。建议您仅将此功能用于测试数据和应用程序，而不要在生产环境中使用。

使用下表确定 AWS 应用程序测试是否支持数据的编码字符集标识符 (CCSID)。如果您的数据使用不受支持的 CCSID，建议您将其转换为支持的 CCSID，或者联系我们以寻求帮助。

CCSID	字符集	描述
37	IBM037、IBM-037、Cp037	主机：美国、加拿大（ESA）、荷兰、葡萄牙、巴西、澳大利亚、新西兰
273	IBM273、IBM-273、Cp273	主机：奥地利、德国
277	IBM277、IBM-277、Cp277	主机：丹麦、挪威
278	IBM278、IBM-278、Cp278	主机：芬兰、瑞典
280	IBM280、IBM-280、Cp280	主机：意大利
284	IBM284、IBM-284、Cp284	主机：西班牙、拉丁美洲（西班牙语）
285	IBM285、IBM-285、Cp285	主机：英国
297	IBM297、IBM-297、Cp297	主机：法国

CCSID	字符集	描述
300	IBM-300	JAPAN DB EBCDIC
301	IBM-301	PC 数据：Japan DB
437	IBM437、IBM-437、US-ASCII、ASCII、Cp437、US-ASCII	PC 数据：PC Base USA，许多其他国家/地区
500	IBM500、IBM-500、Cp500	主机：比利时、加拿大（AS/400）、瑞士、国际拉丁-1
720	IBM-720	MSDOS ARABIC
737	IBM-737、x-IBM737	MSDOS GREEK
775	IBM775、IBM-775	MSDOS BALTIC
808	IBM-808	PC 数据：西里尔文（俄国），含欧元符
813	ISO-8859-7、ISO8859_7	ISO 8859-7：希腊
819	ISO-8859-1、ISO8859_1	ISO 8859-1：拉丁-1 国家/地区
833	IBM-833	KOREAN EBCDIC
834	IBM-834、x-IBM834	KOREAN DB EBCDIC
835	IBM-835	T-CHINESE DB EBCD
836	IBM-836	S-CHINESE EBCDIC
837	IBM-837	S-CHINESE EBCDIC
850	IBM850、IBM-850、Cp850	PC 数据：拉丁-1 国家/地区
855	IBM855、IBM-855、Cp855	PC 数据：西里尔文

CCSID	字符集	描述
856	IBM-856、x-IBM856、Cp856	PC 数据：希伯来语
858	IBM00858、IBM-858、Cp858	PC 数据：拉丁-1 国家/地区，含欧元符
859	IBM-859	PC 数据：LATIN-9
860	IBM860、IBM-860	PC 数据：葡萄牙语
861	IBM861、IBM-861	PC 数据：冰岛
862	IBM862、IBM-862、Cp862	PC 数据：希伯来语（迁移）
863	IBM863、IBM-863	PC 数据：加拿大
865	IBM865、IBM-865、Cp865	PC 数据：丹麦/挪威
866	IBM866、IBM-866、Cp866	PC 数据：西里尔文（俄国）
867	IBM-867	PC 数据：希伯来语，含欧元符
870	IBM870、IBM-870、Cp870	主机：拉丁-2 多语种
871	IBM871、IBM-871、Cp871	主机：冰岛
874	x-IBM874	PC 数据：泰语
875	IBM-875、x-IBM875、Cp875	主机：希腊
897	IBM-897	PC 数据：Japan SB
912	ISO-8859-2、ISO8859_2	ISO 8859-2：拉丁-2 多语种
915	ISO-8859-5、ISO8859_5	ISO 8859-5：西里尔文
916	ISO-8859-8、ISO8859_8	ISO 8859-8：希伯来语
918	IBM918、IBM-918、Cp918	主机：乌尔都语

CCSID	字符集	描述
920	ISO-8859-9、ISO8859_9	ISO 8859-9：拉丁 -5（ECMA-128、土耳其 TS-5881）
921	IBM-921、x-IBM921、Cp921	PC 数据：拉脱维亚、立陶宛
922	IBM-922、x-IBM922、Cp922	PC 数据：爱沙尼亚
923	ISO-8859-15、Cp923、ISO8859_15_FDIS	ISO 8859-15：拉丁-9
924	IBM-924	ISO 8859-15：拉丁-9
927	IBM-927	PC 数据：繁体中文
930	IBM-930、x-IBM930、Cp930	片假名主机：扩展版 SBCS。汉字主机：DBCS，包括 4370 个用户定义的字符
932	IBM-932	PC 数据：Japan Mix
933	IBM-933、x-IBM933、Cp933	主机：扩展版 SBCS。主机：DBCS，包括 1880 个用户定义的字符和 11172 个全角韩文字符
935	IBM-935、x-IBM935、Cp935	主机：扩展版 SBCS。主机：DBCS，包括 1880 个用户定义的字符。
937	IBM-937、x-IBM937、Cp937	主机：扩展版 SBCS。主机：DBCS，包括 6204 个用户定义的字符
939	IBM-939、x-IBM939、Cp939	拉丁语主机：扩展版 SBCS。汉字主机：DBCS，包括 4370 个用户定义的字符。

CCSID	字符集	描述
942	IBM-942、IBM-942C、x-IBM942、x-IBM942C、Cp942、Cp942C	PC 数据：扩展版 SBCS。PC 数据：DBCS，包括 1880 个用户定义的字符
943	IBM-943、IBM-943C、Shift_JIS、windows-31j、windows-932、x-IBM943、x-IBM943C、Cp943、Cp943C、MS932	PC 数据：SBCS。PC 数据：适用于开放环境的 DBCS，包括 1880 个 IBM﷿ 用户定义的字符
947	IBM-947	T-CHINESE BIG-5
948	IBM-948、x-IBM948、Cp948	PC 数据：扩展版 SBCS。PC 数据：DBCS，包括 6204 个用户定义的字符
949	IBM-949、IBM-949C、x-IBM949、x-IBM949C、Cp949、Cp949C	IBM KS 代码 - PC 数据：SBCS。IBM KS 代码 - PC 数据：DBCS，包括 1880 个用户定义的字符
950	Big5、IBM-950、x-IBM950、Cp950	PC 数据：SBCS（IBM BIG5）。PC 数据：DBCS，包括 13493 个中国台湾地区字符、566 个 IBM 精选字符、6204 个用户定义的字符
951	IBM-951	PC 数据：IBM KS
954	EUC-JP、IBM-954、IBM-954C	G0：JIS X201 Roman。G1：JIS X208-1990。G1：JIS X201 Katakana。G1：JIS X212
964	EUC-TW、IBM-964、x-IBM964、Cp964	G0：ASCII。G1：CNS 11643 plane 1。G1：CNS 11643 plane 2。

CCSID	字符集	描述
970	EUC-KR、x-IBM970、Cp970	G0：ASCII。G1：KSC X5601-1989，包括 1880 个用户定义的字符
971	IBM-971	KOREAN EUC
1006	IBM-1006、x-IBM1006、Cp1006	ISO-8：乌尔都语
1025	IBM-1025、x-IBM1025、Cp1025	主机：西里尔文多语种
1026	IBM1026、IBM-1026、Cp1026	主机：拉丁-5（土耳其）
1027	IBM-1027	JAPAN LATIN EBCD
1041	IBM-1041	PC 数据：日本
1043	IBM-1043	PC 数据：繁体中文
1046	IBM-1046、IBM-1046S、x-IBM1046	ARABIC - PC
1047	IBM1047、IBM-1047	主机：拉丁-1
1051	hp-roman8	HP EMULATION
1088	IBM-1088	PC 数据：Korea KS
1089	ISO-8859-6、ISO8859_6	ISO 8859-6：阿拉伯语
1097	IBM-1097、x-IBM1097、Cp1097	主机：波斯语
1098	IBM-1098、x-IBM1098、Cp1098	PC 数据：波斯语

CCSID	字符集	描述
1112	IBM-1112、x-IBM1112、Cp1112	主机：拉脱维亚、立陶宛
1114	IBM-1114	PC 数据：T-CH SB
1115	IBM-1115	PC 数据：S-CH GB
1122	IBM-1122、x-IBM1122、Cp1122	主机：爱沙尼亚
1123	IBM-1123、x-IBM1123、Cp1123	主机：西里尔文（乌克兰）
1124	IBM-1124、x-IBM1124、Cp1124	8 位：西里尔文（白俄罗斯）
1140	IBM01140、IBM-1140、Cp1140	主机：美国、加拿大（ESA）、荷兰、葡萄牙、巴西、澳大利亚、新西兰，含欧元符
1141	IBM01141、IBM-1141、Cp1141	主机：奥地利、德国，含欧元符
1142	IBM01142、IBM-1142、Cp1142	主机：丹麦、挪威，含欧元符
1143	IBM01143、IBM-1143、Cp1143	主机：芬兰、瑞典，含欧元符
1144	IBM01144、IBM-1144、Cp1144	主机：意大利，含欧元符
1145	IBM01145、IBM-1145、Cp1145	主机：西班牙、拉丁美洲（西班牙语），含欧元符
1146	IBM01146、IBM-1146、Cp1146	主机：英国，含欧元符

CCSID	字符集	描述
1147	IBM01147、IBM-1147、Cp1147	主机：法国，含欧元符
1148	IBM01148、IBM-1148、Cp1148	主机：比利时、加拿大（AS/400）、瑞士、国际拉丁-1，含欧元符
1149	IBM01149、IBM-1149、Cp1149	主机：冰岛，含欧元符
1200	UTF-16BE	含字符集 65535 的 Unicode。在没有字节顺序标记（BOM）的情况下，假定为 UTF-16 BE（big-endian）。
1202	UTF-16LE	含 IBM PUA 的 UTF-16 LE
1204	UTF-16	含 IBM PUA 的 UTF-16
1208	UTF-8、UTF-8J、UTF8	含字符集 65535 的 Unicode。UTF-8
1232	UTF-32BE	含 IBM PUA 的 UTF-32 BE
1234	UTF-32LE	含 IBM PUA 的 UTF-32 LE
1236	UTF-32	含 IBM PUA 的 UTF-32
1351	IBM-1351	JAPAN OPEN
1362	IBM-1362	KOREAN MS-WIN
1363	IBM-1363、IBM-1363C、windows-949、MS949	PC 数据：MS Windows Korean SBCS。PC 数据：MS Windows Koran DBCS，包括 11172 个全角韩文字符

CCSID	字符集	描述
1364	IBM-1364	主机：扩展版 SBCS。主机：DBCS，包括 1880 个用户定义的字符和 11172 个全角韩文字符
1370	IBM-1370	PC 数据：扩展版 SBCS，含欧元符。PC 数据：DBCS，包括 6204 个用户定义的字符，含欧元符
1371	IBM-1371	主机：扩展版 SBCS，含欧元符。主机：DBCS，包括 6204 个用户定义的字符，含欧元符
1375	Big5-HKSCS	Mixed Big-5 Ext for HKSCS
1380	IBM-1380	PC 数据：S-CH GB
1381	IBM-1381、x-IBM1381、Cp1381	PC 数据：扩展版 SBCS（IBM GB）。PC 数据：DBCS（IBM GB），包括 31 个 IBM 精选字符、1880 个用户定义的字符
1382	IBM-1382	S-CHINESE EUC
1383	EUC-CN、GB2312、IBM-1383、x-IBM1383、Cp1383	G0：ASCII。G1：GB 2312-80 集
1385	IBM-1385	PC 数据：S-CH GBK
1386	GBK、IBM-1386、windows-936、MS936	PC 数据：S-Chinese GBK 和 T-Chinese IBM BIG-5。PC 数据：S-Chinese GBK
1388	IBM-1388	主机：扩展版 SBCS。主机：DBCS，包括 1880 个用户定义的字符

CCSID	字符集	描述
1390	IBM-1390	片假名主机：扩展版 SBCS，含欧元符。汉字主机：DBCS，包括 6205 个用户定义的字符
1399	IBM-1399	拉丁语主机：扩展版 SBCS，含欧元符。汉字主机：DBCS，包含 4370 个用户定义的字符，含欧元符
5050	JIS0201、JIS0208、JIS0212、JIS0201、JIS0208、JIS0212	G0：JIS X201 Roman。G1：JIS X208-1990。G1：JIS X201 Katakana。G1：JIS X212
5054	ISO-2022-JP	JAPANESE TCP
5346	windows-1250、Cp1250	MS Windows：拉丁-2，第 2 版，含欧元符
5347	windows-1251、Cp1251	MS Windows：西里尔文，第 2 版，含欧元符
5348	windows-1252、Cp1252	MS Windows：拉丁-1 国家，第 2 版，含欧元符
5349	windows-1253、Cp1253	MS Windows：希腊，第 2 版，含欧元符
5350	windows-1254、Cp1254	MS Windows：土耳其，第 2 版，含欧元符
5351	windows-1255、Cp1255	MS Windows：希伯来语，第 2 版，含欧元符
5352	windows-1256、Windows-1256S、Cp1256	MS Windows：阿拉伯语，第 2 版，含欧元符

CCSID	字符集	描述
5353	windows-1257、Cp1257	MS Windows：Baltic Rim，第 2 版，含欧元符
5354	windows-1258、Cp1258	MS Windows：越南语，第 2 版，含欧元符
5488	GB18030	GB18030、1 字节数据 GB18030、2 字节数据 GB18030、4 字节数据
9030	IBM-838、Cp838	主机：泰语扩展版 SBCS
9066	IBM-874、Cp874	PC 数据：泰语扩展版 SBCS
9400	CESU-8	含 IBM PUA 的 CESU-8
25546	ISO-2022-KR	KOREAN TCP
33722	IBM-33722、IBM-33722C	IBMeucJP

AWS 大型机现代化中的文件传输

AWS 大型机现代化文件传输允许您将大型机数据集传输并转换到 Amazon S3，以用于大型机现代化、迁移和增强用例。

主题

什么是 AWS Mainframe Modernization 文件传输？

借助 AWS 大型机现代化文件传输，您可以使用完全托管的服务传输和转换数据集和文件，从而加快和简化AWS大型机现代化服务和 Amazon S3 的现代化、迁移和增强用例。

主题

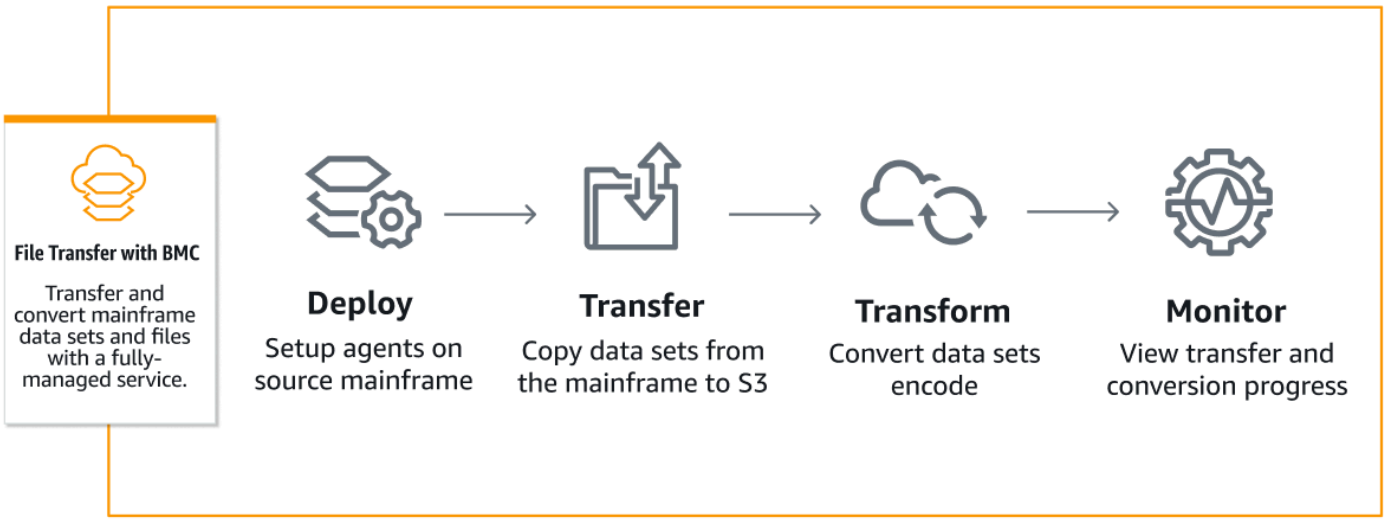
AWS Mainframe Modernization 文件传输功能的优势

借助 AWS Mainframe Modernization 文件传输功能，您可以将数据集从大型机传输到 Amazon S3。一些优势包括：

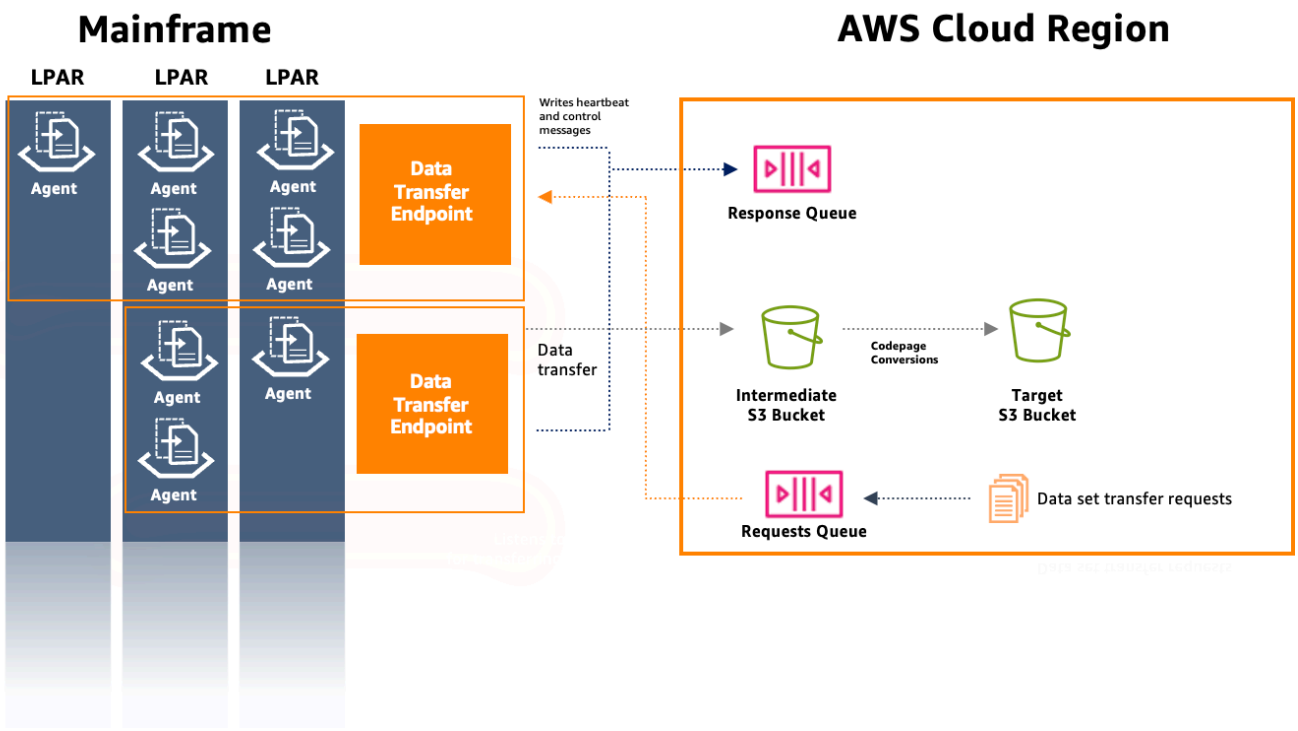
- 发现源大型机数据集和构件
- 自动传输和数据集转换
- 可扩展性、效率和速度提升，可更快地将数据集传输到 AWS

AWS Mainframe Modernization 文件传输功能的工作原理

下图从概念层面概述了 AWS Mainframe Modernization 文件传输功能的工作原理。



下图是 AWS Mainframe Modernization 文件传输功能的架构概述。



安装文件传输代理

按照本指南 step-by-step 完成在源主机上安装代理和配置代理的先决条件。

主题

步骤 1：登录 ISPF

登录您的 ISPF（Interactive System Productivity Facility）会话。通常通过 3270 终端仿真器完成登录。

步骤 2：为 z/FS 分配数据集

使用 ISPF 的数据集实用程序，为 z/FS 分配一个新的数据集。通常，这是在步骤 1 中的数据集列表实用程序中完成的。

1. 转到选项 3.4（实用程序 --> 数据集）。
2. 按键“C”创建一个新的数据集。
3. 输入该数据集的名称（例如“yourhlq.M2AGENT.ZFS”）。
4. 将数据集类型指定为“大格式”，主要大小为 1000 个柱体，次要大小为 200。

5. 将数据集组织（DSORG）设置为 PS，将记录格式（RECFM）设置为“U”（未定义）。
6. 完成创建过程

步骤 3：将数据集格式化为 z/FS

创建数据集后，将其格式化为 z/FS 文件系统。

一种方法是使用以下作业控制语言（JCL）：

```
//FORMAT  EXEC PGM=IOEAGFMT,PARM='AGGRNAME(yourhlq.M2AGENT.ZFS),FORMAT,AGGRSIZE(1200)'
//SYSPRINT DD  SYSOUT=A
```

提交此作业并检查其是否成功完成。

步骤 4：为 z/OS 定义文件系统

使用 DEF FILESYSTEM 命令或通过批处理作业为 z/OS 定义 z/FS。使用以下命令：

```
DEF FILESYSTEM('yourhlq.M2AGENT.ZFS') TYPE(ZFS) MODE(R/W) MOUNTPOINT('/usr/lpp/aws/m2-agent')
```

> **Note**
>
> 此步骤需要系统级权限。

步骤 5：挂载文件系统

要挂载文件系统，请使用 MOUNT 命令。您可以在 ISPF 的命令行中或批量挂载文件系统。

例如：

```
MOUNT FILESYSTEM('yourhlq.M2AGENT.ZFS') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/usr/lpp/aws/m2-agent')
```

步骤 6：验证挂载

使用 D OMVS,A 命令或在 Unix 系统服务（USS）中进行检查，来验证文件系统是否已正确挂载。

步骤 7：输入 OMV

使用以下命令输入 OMV：

```
TSO OMVS
```

步骤 8：设置代理安装目录环境变量

使用以下命令设置代理安装目录环境：

```
export AGENT_DIR=/usr/lpp/aws/m2-agent
```

步骤 9：设置工作目录环境变量

使用以下命令设置工作目录环境变量：

```
export WORK_DIR=$AGENT_DIR/tmp
```

步骤 10：创建工作目录

使用以下命令设置工作目录环境：

```
mkdir -p $WORK_DIR
```

第 11 步：将 AWS 大型机现代化 tar 包复制到 z/OS 上的工作目录中

使用 FTP 或任何传输方法时，请确保以二进制模式传输该 tar 文件。

步骤 12：取得根用户身份

使用以下命令取得根用户身份：

```
su
```

按照以下步骤完成代理安装：

> **Note**
>
> 要继续执行这些步骤，您必须先取得根用户身份。

1. 使用以下命令将 m2-agent 版本环境变量设置为当前正在安装的版本：

```
export M2_AGENT_VERSION=1.0.0
```

2. 使用以下命令解压缩代理 tar 软件包：

```
tar -xpf m2-agent-package-$M2_AGENT_VERSION.tar -C $AGENT_DIR
```

3. 使用以下命令创建指向当前代理安装目录的 `current-version` 符号链接：

```
ln -s $AGENT_DIR/m2-agent-v$M2_AGENT_VERSION $AGENT_DIR/current-version
```

4. 更新并提交 CPY#PDS 以创建文件传输代理数据集。

> **Note**
>
> JCL 使用 SYS2.AWS.M2 HLQ。

要创建文件传输代理，请设置参数行 000006-000012。另外，更新三个符号变量 HLQ、VOLSER 和 AGNTPATH，以便稍后在 JCL 中使用：

```
oedit $AGENT_DIR/current-version/installation/CPY#PDS
submit $AGENT_DIR/current-version/installation/CPY#PDS
```

> **Note**
>
> 此 JCL 专为在大型机上设置代理安装的某些方面而量身定制。它分配必要的数据集，然后将特定文件从 Unix 文件系统复制到这些数据集。

配置权限和 STC

1. 按照说明，更新并提交其中一个 SYS2.AWS.M2.SAMPLIB(SEC#RACF)（用于设置 RACF 权限）或 SYS2.AWS.M2.SAMPLIB(SEC#TSS)（用于设置 TSS 权限）。这些成员由前面的 CPY#PDS 步骤创建。
2. 如果默认文件传输代理目录路径 (/usr/lpp/aws/m2-agent) 已更改，请更新 SYS2.AWS.M2.SAMPLIB(M2AGENT) STC JCL 中的 PWD 导出。

3. 更新 SYS2.AWS.M2.SAMPLIB(M2AGENT) JCL 并将其复制到 SYS1.PROCLIB。
4. 使用以下命令将 SYS2.AWS.M2.LOADLIB 添加到 APF 列表：

```
SETPROG APF ADD DSNAME(SYS2.AWS.M2.LOADLIB) SMS
```

5. 将代理的 logs 和 diag 文件夹的组和所有者设置为代理用户/组 （M2USER/M2GROUP）。使用以下命令：

```
chown -R M2USER:M2GROUP $AGENT_DIR/current-version/logs
chown -R M2USER:M2GROUP $AGENT_DIR/current-version/diag
```

创建具有长期访问凭证的 IAM 用户

您需要创建 IAM 用户使用响应和请求队列以及将数据集保存到 Amazon S3 存储桶所需的大型机代理。

要创建此用户时，请执行以下操作：

1. 直接在权限选项中选择附加策略。
2. 创建用户后，打开安全凭证选项卡，并创建一个访问密钥。有关创建 IAM 访问密钥的更多信息，请参阅管理 IAM 用户的访问密钥。
3. 在访问密钥部分，当出现提示时，为用例选择其他。

> **Note**
>
> 选择完成之前，保存在访问密钥创建向导的最后一页上显示的访问密钥和秘密访问密钥。这些密钥用于配置大型机代理。

> **Note**
>
> 保存用于与 IAM 角色建立信任关系的 IAM 用户 ARN。

为代理创建 IAM 角色

您可以创建一个新的 IAM 角色，该角色的可信实体类型为自定义信任策略。该策略将使用以下模板：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DataTransferEndpointAgentSqsReceive",
            "Effect": "Allow",
            "Action": [
                "sqs:DeleteMessage",
                "sqs:ReceiveMessage"
            ],
            "Resource": "<data-transfer-endpoint-request-queue-arn>"
        },
        {
            "Sid": "DataTransferEndpointS3",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "<data-transfer-endpoint-intermediate-bucket-arn>/*"
        },
        {
            "Sid": "DataTransferEndpointAgentSqsSend",
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource": "<data-transfer-endpoint-response-queue-arn>"
        },
        {
            "Sid": "DataTransferEndpointAgentKmsDecrypt",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "<kms-key-id>"
        }
    ]
}
```

其中：

- `request-queue-arn` 和 `response-queue-arn` 是在数据传输端点初始化期间创建的请求 Amazon SQS 队列的 ARN。
- `transfer-bucket-arn` 是之前创建的传输存储桶的 ARN。

> **Note**
>
> 您可以使用 AWS 控制台查找所有这些值。

> **Note**
>
> 保存角色名称，稍后您将使用该名称来配置大型机代理。

代理配置

配置文件传输代理：

1. 导航到 `$AGENT_DIR/current-version/config`。
2. 使用以下命令编辑代理的配置文件 `appication.properties`，以添加环境配置：

```
oedit $AGENT_DIR/current-version/config/application.properties
```

例如：

```
agent.environments[0].account-id=<AWS_ACCOUNT_ID>
agent.environments[0].agent-role-name=<AWS_IAM_ROLE_NAME>
agent.environments[0].access-key-id=<AWS_IAM_ROLE_ACCESS_KEY>
agent.environments[0].secret-access-id=<AWS_IAM_ROLE_SECRET_KEY>
agent.environments[0].bucket-name=<AWS_S3_BUCKET_NAME>
agent.environments[0].environment-name=<AWS_REGION>
agent.environments[0].region=<AWS_REGION>
```

其中：

- `AWS_ACCOUNT_ID` 是客户账户的 ID。
- `AWS_IAM_ROLE_NAME` 是在the section called “为代理创建 IAM 角色”中创建的 IAM 角色的名称。
- `AWS_IAM_ROLE_ACCESS_KEY` 是在the section called “创建具有长期访问凭证的 IAM 用户”中创建的 IAM 用户的访问密钥
- `AWS_IAM_ROLE_SECRET_KEY` 是在the section called “创建具有长期访问凭证的 IAM 用户”中创建的 IAM 用户的访问密钥。

- AWS_S3_BUCKET_NAME 是使用数据传输端点创建的传输存储桶的名称。
- AWS_REGION 是您在其中配置文件传输代理的区域。

> **Note**
>
> 可以有几个这样的部分，只要括号中的索引 ([0]) 为每个部分递增。

必须重新启动代理才能使更改生效。

要求

1. 添加或删除参数时，必须停止并启动代理。在 CLI 中使用以下命令启动文件传输代理：

```
/S M2AGENT
```

要停止 M2 代理，请在 CLI 中使用以下命令：

```
/P M2AGENT
```

2. 通过定义多个环境，您可以将文件传输代理传输到多个区域和账户。 AWS

```
#Region 1
agent.environments[0].account-id=AWS_ACCOUNT_ID
agent.environments[0].agent-role-name=AWS_IAM_ROLE_NAME
agent.environments[0].access-key-id=AWS_IAM_ROLE_ACCESS_KEY
agent.environments[0].secret-access-id=AWS_IAM_ROLE_SECRET_KEY
agent.environments[0].bucket-name=AWS_S3_BUCKET_NAME
agent.environments[0].environment-name=AWS_REGION
agent.environments[0].region=AWS_REGION

#Region 2
agent.environments[1].account-id=AWS_ACCOUNT_ID
agent.environments[1].agent-role-name=AWS_IAM_ROLE_NAME
agent.environments[1].access-key-id=AWS_IAM_ROLE_ACCESS_KEY
agent.environments[1].secret-access-id=AWS_IAM_ROLE_SECRET_KEY
agent.environments[1].bucket-name=AWS_S3_BUCKET_NAME
agent.environments[1].environment-name=AWS_REGION
agent.environments[1].region=AWS_REGION
```

数据传输端点

借助数据传输端点，可实现高可用性、可扩展性，并简化源大型机上的代理管理。单个代理安装在大型机 LPAR 上，可以组合成一个数据传输端点。当请求传输数据集时，数据传输端点中的一个代理将处理该传输。要启动数据传输，数据传输端点上必须至少有一个代理处于联机状态。

此过程假设您已完成设置 AWS 大型机现代化 和在源大型机上配置文件传输代理中的步骤。

创建数据传输端点

要为文件传输创建数据传输端点，您必须在 AWS 大型机现代化控制台中执行以下步骤。

创建数据传输端点

1. 打开 AWS 大型机现代化控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域 选择器中，选择要将文件从大型机传输到 Amazon S3 存储桶的区域。
3. 在数据传输端点页面的文件传输下，选择创建数据传输端点。
4. 在数据传输端点先决条件页面上，阅读所有说明，确保您已完成这些步骤。确认后，选择下一步。
5. 在配置数据传输端点页面上，为数据传输端点添加基本信息。

    1. 在基本信息部分，输入数据传输端点名称、描述和 KMS 密钥。有关 KMS 密钥的更多信息，请参阅创建密钥。

        > **Note**
        >
        > 数据传输端点名称必须与在源大型机上配置文件传输代理时定义的名称一致。

        > **Note**
        >
        > 您必须为 KMS 添加以下基于资源的策略，以便 AWS 大型机现代化服务可以读取和使用这些密钥进行加密/解密：

        ```
        {
          "Sid" : "Enable AWS M2 Permissions",
          "Effect" : "Allow",
          "Principal" : {
              "Service" : "m2.amazonaws.com"
                    },
        ```

```
    "Action" : [
        "kms:Encrypt",
        "kms:Decrypt"
               ],
    "Resource" : "*"
}
```

2. 指定中间数据的 S3 位置，即存储从大型机传输的数据集的中间 S3 位置。

> **Note**
>
> 建议您为传输任务创建新的 Amazon S3 存储桶。有关更多信息，请参阅创建存储桶。您也可以通过选择浏览 S3 选项来浏览现有 Amazon S3 存储桶。

3. 输入必填字段后，选择下一步。

6. 在审核并创建数据传输端点页面上，检查您是否已完成先决条件，并审核基本信息。确认后，选择创建和连接。
7. 在检查连接页面上，一旦建立代理连接，您将看到显示其 ID 和检测信号的所有代理。建立连接后，您将收到一条消息“数据传输端点已成功创建”。

> **Note**
>
> 如果您看到“无法建立代理连接”消息，请重新执行步骤 4，确保您已满足所有必需的先决条件。

8. 选择完成。

您将被重定向到数据传输端点概述页面，可以在其中查看所有数据传输端点的列表。您还可以看到可用或发生故障的数据传输端点。

您还可以按名称搜索数据传输端点，并访问每个可用代理的其他信息。

传输任务

传输任务用于定义数据集的源编码和目标编码。源编码是源数据集的格式，目标编码是这些数据集将存储在目标 Amazon S3 存储桶中的格式。这些目标存储桶由传输任务定义。

此过程假设您已完成设置 AWS 大型机现代化 中的步骤并设置了the section called “数据传输端点”。

主题

创建传输任务

要为文件传输创建传输任务，您必须在 AWS 大型机现代化控制台中执行以下步骤。

创建传输任务

> **Note**
>
> 您必须至少有一个数据传输端点才能创建新的传输任务。

1. 打开 AWS 大型机现代化控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域 选择器中，选择您要将文件从大型机传输到 Amazon S3 存储桶的区域。
3. 在传输任务页面的文件传输下，选择数据传输端点以创建传输任务。
4. 如果您的数据传输端点没有任何传输任务，则可以通过选择创建传输任务来创建新任务。
5. 在“创建转移任务”页面上，设置转移任务的属性。

    - 在此页面上，输入传输任务的基本信息，包括转移任务名称、描述、密钥和数据集搜索条件。

> **Note**
>
> - 使用通过数据传输端点定义的 KMS 密钥对密钥进行加密。该密钥还应包含使用 `userId` 和 `password` 密钥访问大型机上的数据集所需的大型机凭证。有关更多信息，请参阅 AWS Secrets Manager 密钥。
> - 您必须使用以下基于资源的策略配置密钥，以便 AWS 大型机现代化服务可以访问它来执行数据传输任务：

```
{
  "Version" : "2012-10-17",
  "Statement" : [ {
    "Effect" : "Allow",
    "Principal" : {
        "Service" : "m2.amazonaws.com"
```

```
    },
    "Action" : [ "secretsmanager:GetSecretValue",
                 "secretsmanager:DescribeSecret" ],
    "Resource" : "*"
  } ]
}
```

> **Note**
>
> 当前支持的最大数据集大小为 90 GiB。

- 输入或浏览文件的目标 Amazon S3 存储桶位置。
- 选择默认的数据传输端点。您也可以选择从可用端点更改该端点。

6. 选择下一步。
7. 在“选择数据集”页面上，必须手动更新所选每个数据集的源编码和目标编码。源编码是源数据集格式，目标编码是目标数据集格式，用于转换数据集。
8. 确认源编码和目标编码后，选择下一步。
9. 在审核并创建页面上，您可以审核或编辑传输任务的相关信息。
10. 选择创建传输任务。

您会看到一条消息“传输任务已成功创建”。

查看传输任务

要查看文件传输的传输任务，您必须在 AWS 大型机现代化控制台中执行以下步骤。

查看传输任务

1. 打开 AWS 大型机现代化控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在 AWS 区域 选择器中，选择您要将文件从大型机传输到 Amazon S3 存储桶的区域。
3. 在传输任务页面的文件传输下，选择数据传输端点以查看您的传输任务。
4. 对于已存在传输任务的端点，这些任务将填充到传输任务部分下。您可以选择从此列表中查看任何传输任务的详细信息。

教程：开始使用 AWS Mainframe Modernization 文件传输

AWS 大型机现代化文件传输允许您为大型机现代化、迁移和增强用例传输和转换大型机数据集。

执行本教程中的步骤，了解 AWS Mainframe Modernization 文件传输的工作原理。

概述

文件传输功能包括以下部分：

1. 安装在源大型机上的代理。
2. 直接从 AWS 大型机现代化管理服务控制台访问数据集发现、传输和转换功能。

作为用户，您可以将数据集从大型机传输到 Amazon S3 存储桶。

主题

步骤 1：将代理二进制文件 tar 包从传输 AWS 到大型机逻辑分区

从 M2-agent tar 链接下载 tar 文件。

步骤 2：在源大型机上配置文件传输代理

在此步骤中，您将在源大型机上配置并启动 AWS Mainframe Modernization 文件传输代理。需要代理来促进文件传输服务功能和源大型机之间的通信。每个大型机至少需要一个代理。可以启动多个代理，以实现高可用性和增强的可扩展性。

按照the section called “安装文件传输代理”指南中的说明，在大型机上完成文件传输代理的安装。

步骤 3：创建数据传输端点

按照the section called “数据传输端点”页面上的步骤创建新的数据传输端点。

步骤 4：创建传输任务

按照the section called “传输任务”页面上的步骤创建和管理您的传输任务。

步骤 5：查看传输任务进度

您可以在 AWS 大型机现代化控制台中查看传输任务的进度。有关详细信息，请参阅the section called “查看传输任务”部分。

AWS Mainframe Modernization 中的安全性

AWS 十分重视云安全性。作为 AWS 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。

安全性是 AWS 和您的共同责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS 负责保护在 AWS 云中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 AWS Compliance Programs 的一部分。要了解适用于 AWS Mainframe Modernization 的合规性计划，请参阅合规性计划范围内的 AWS 服务。
- 云中的安全性 - 您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档将帮助您了解如何在使用 AWS Mainframe Modernization 时应用责任共担模式，并介绍了如何配置 AWS Mainframe Modernization 以实现您的安全性和合规性目标。您还可以了解如何使用其他 AWS 服务以帮助您监控和保护 AWS Mainframe Modernization 资源。

AWS Mainframe Modernization 提供受 IAM 保护的资源（应用程序、环境、部署等），这些资源是 AWS Mainframe Modernization 管理资源，对这些资源进行任何操作均需 IAM 策略允许。

AWS Mainframe Modernization 用于更换平台时，也受到 IAM 的保护。IAM 还通过标准 IAM 策略授予或拒绝主体对源自原始大型机环境的已定义资源执行特定操作的权限。当应用程序尝试对受保护的资源执行此类操作时，AWS Mainframe Modernization 更换平台运行时会调用 IAM 授权服务。IAM 将根据标准 IAM 策略评估机制返回“允许”或“拒绝”。

目录

AWS 大型机现代化中的数据保护

分 AWS 担责任模型适用于 AWS 大型机现代化中的数据保护。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设施上的内容的控制。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。有关欧洲数据保护的信息，请参阅AWS 安全性博客上的 AWS 责任共担模式和 GDPR 博客文章。

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。我们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)。
- 使用 SSL/TLS 与资源通信。 AWS 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 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 140-2 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 AWS 服务 使用控制台、API 或 AWS SDK 进行 AWS 大型机现代化或其他操作时。 AWS CLI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网址，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。

AWS 大型机现代化收集的数据

AWS 大型机现代化会从您那里收集几种类型的数据：

- `Application configuration`：这是您为配置应用程序而创建的 JSON 文件。它包含您对 AWS 大型机现代化提供的不同选项的选择。该文件还包含相关 AWS 资源的信息，例如存储应用程序项目的亚马逊简单存储服务路径或存储数据库凭证的 AWS Secrets Manager 亚马逊资源名称 (ARN)。
- `Application executable (binary)`：这是您编译的二进制文件，打算在 AWS 大型机现代化上部署。

- `Application JCL or scripts`：此源代码代表您的应用程序管理批处理作业或其他处理。
- `User application data`：导入数据集时， AWS 大型机现代化将它们存储在关系数据库中，以便您的应用程序可以访问它们。
- `Application source code`: 通过 Amazon AppStream 2.0， AWS 大型机现代化为您提供了编写和编译代码的开发环境。

AWS 大型机现代化以原生方式存储这些数据。 AWS我们收集的您的数据存储在 AWS Mainframe Modernization 托管的 Amazon S3 存储桶中。部署应用程序时， AWS 大型机现代化会将数据下载到由亚马逊弹性区块存储支持的亚马逊弹性计算云实例上。触发清理后，数据将从 Amazon EBS 卷和 Amazon S3 中删除。Amazon EBS 卷是单租户的，这表示一个实例用于一个客户。实例不共享。当您删除运行时环境时，Amazon EBS 卷也会被删除。当您删除应用程序时，构件和配置将从 Amazon S3 中删除。

应用程序日志存储在 Amazon 中 CloudWatch。客户应用程序日志消息也会导 CloudWatch 出到。CloudWatch 日志可能包含客户敏感数据，例如业务数据或调试消息中的安全信息）。有关更多信息，请参阅使用 A AWS mazon 监控大型机现代化 CloudWatch。

此外，如果您选择将一个或多个 Amazon Elastic File System 或 Amazon FSx 文件系统附加到运行时环境，则这些系统中的数据将存储在 AWS中。如果您决定停止使用这些文件系统，则需要清理相应数据。

当您将数据放入 AWS 大型机现代化用于应用程序部署和数据集导入的 Amazon S3 存储桶时，您可以使用所有可用的 Amazon S3 加密选项来保护数据。此外，如果您将一个或多个 Amazon EFS 和 Amazon FSx 文件系统附加到运行时环境，则可以使用 Amazon EFS 和 Amazon FSx 加密选项。

AWS 大型机现代化服务的静态数据加密

AWS 大型机现代化与集成， AWS Key Management Service 为所有永久存储数据的依赖资源（即亚马逊简单存储服务、Amazon DynamoDB 和亚马逊弹性区块存储）提供透明的服务器端加密 (SSE)。AWS 大型机现代化在中为您创建和管理对称加密 AWS KMS 密钥。 AWS KMS

默认情况下，静态数据加密有助于降低保护敏感数据的操作开销和复杂性。同时，它还支持迁移需要满足严格加密合规性和监管要求的应用程序。

在创建运行时环境和应用程序时，您无法禁用此加密层或选择其他加密类型。

您可以为 AWS 大型机现代化应用程序和运行时环境使用自己的客户托管密钥来加密 Amazon S3 和 Amazon EBS 资源。

对于您的 AWS 大型机现代化应用程序，您可以使用此密钥来加密您的应用程序定义以及其他应用程序资源，例如 JCL 文件，这些资源保存在服务账户中创建的 Amazon S3 存储桶中。有关更多信息，请参阅创建应用程序 。

对于您的 AWS 大型机现代化运行时环境， AWS 大型机现代化使用您的客户托管密钥加密其创建并附加到您的 AWS 大型机现代化 Amazon EC2 实例（也位于该服务的账户中）的 Amazon EBS 卷。有关更多信息，请参阅创建运行时环境。

> **Note**
>
> DynamoDB 资源始终使用大型机现代化服务 AWS 托管式密钥 账户进行 AWS 加密。您无法使用客户托管密钥加密 DynamoDB 资源。

AWS 大型机现代化使用您的客户托管密钥完成以下任务：

- 重新部署应用程序。
- 更换 AWS 大型机现代化 Amazon EC2 实例。

AWS 大型机现代化不会使用您的客户托管密钥来加密亚马逊关系数据库服务或 Amazon Aurora 数据库、亚马逊简单队列服务队列以及为支持 AWS 大型机现代化应用程序而创建的亚马逊 ElastiCache 缓存，因为它们都不包含客户数据。

有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的客户托管密钥。

下表汇总了 AWS 大型机现代化如何加密您的敏感数据。

数据类型	AWS 托管式密钥 加密	客户托管密钥加密
`Definition` 包含特定应用程序的定义。	已启用	已启用
`EnvironmentSummary` 包含有关运行时环境的信息。	已启用	已启用
`ApplicationSummary`	已启用	已启用

数据类型	AWS 托管式密钥 加密	客户托管密钥加密
包含有关 AWS 大型机现代化应用程序的信息。		
`DeploymentSummary` 包含有关部署 AWS 大型机现代化应用程序的信息。	已启用	已启用

> **Note**
>
> AWS 大型机现代化可自动启用静态加密 AWS 托管式密钥 ，从而免费保护您的敏感数据。但是，使用客户管理的密钥需要 AWS KMS 付费。有关定价的更多信息，请参阅 AWS Key Management Service 定价。

有关的更多信息 AWS KMS，请参阅 AWS Key Management Service。

AWS 大型机现代化如何使用补助金 AWS KMS

AWS 大型机现代化需要获得授权，才能使用您的客户托管密钥。

当您创建应用程序或运行时环境，或者在使用客户托管密钥加密 AWS 的大型机现代化中部署应用程序时， AWS 大型机现代化将通过向发送CreateGrant请求来代表您创建补助金。 AWS KMS中的赠款 AWS KMS 用于授予 AWS 大型机现代化访问客户账户中 KMS 密钥的权限。

AWS 大型机现代化需要获得授权，才能使用您的客户托管密钥进行以下内部操作：

- 向发送DescribeKey请求， AWS KMS 以验证在创建应用程序、运行时环境或应用程序部署时输入的对称客户托管密钥 ID 是否有效。
- 向发送GenerateDataKey请求 AWS KMS 以加密托管 AWS 大型机现代化运行时环境的 Amazon EC2 实例上的 Amazon EBS 卷。
- 向发送解密请求 AWS KMS 以解密 Amazon EBS 上的加密内容。

AWS 在创建运行时环境、创建或重新部署应用程序以及创建部署时，大型机现代化使用 AWS KMS 授权来解密存储在 Secrets Manager 中的密钥。 AWS 大型机现代化所产生的补助金支持以下操作：

- 创建或更新运行时环境授权：
  - Decrypt
  - Encrypt
  - ReEncryptFrom
  - ReEncryptTo
  - GenerateDataKey
  - DescribeKey
  - CreateGrant
- 创建或重新部署应用程序授权：
  - GenerateDataKey
- 创建部署授权：
  - Decrypt

您可以随时撤销授予访问权限，或删除服务对客户托管密钥的访问权限。如果这样做， AWS 大型机现代化将无法访问由客户托管密钥加密的任何数据，这会影响依赖这些数据的操作。例如，如果 AWS 大型机现代化尝试访问由客户托管密钥加密的应用程序定义，但未授予该密钥，则应用程序创建操作将失败。

AWS 大型机现代化收集用户应用程序配置（JSON 文件）和工件（二进制文件和可执行文件）。AWS Mainframe Modernization 还会创建元数据，用于跟踪用于其操作的各种实体，并创建日志和指标。客户可见的日志和指标包括：

- CloudWatch 反映应用程序和运行时引擎（ AWS Blu Age 或 Micro Focus）的日志。
- CloudWatch 操作仪表板的指标。

此外， AWS 大型机现代化还会收集有关服务的使用数据和指标，用于计量、活动报告等。客户无法看到这些数据。

AWS 大型机现代化根据数据类型将这些数据存储在不同的位置。您上传的客户数据存储在 Amazon S3 桶中。服务数据存储在 Amazon S3 和 DynamoDB 中。部署应用程序时，您的数据和服务数据都会下载到 Amazon EBS 卷上。如果您选择将 Amazon EFS 或 Amazon FSx 存储附加到运行时环境，则存储在这些文件系统中的数据也会下载到 Amazon EBS 卷上。

默认情况下会配置静态加密。您无法将其禁用或更改。目前，您也无法更改其配置。

创建客户托管密钥

您可以使用 AWS Management Console 或 AWS KMS API 创建对称的客户托管密钥。

创建对称的客户托管密钥

按照《AWS Key Management Service 开发人员指南》中创建对称的客户托管密钥的步骤进行操作。

密钥策略

密钥策略控制对客户托管密钥的访问。每个客户托管密钥必须只有一个密钥策略，其中包含确定谁可以使用该密钥以及如何使用该密钥的声明。创建客户托管密钥时，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。

要将客户托管密钥用于 AWS 大型机现代化资源，密钥策略中必须允许以下 API 操作：

- kms:CreateGrant – 添加客户托管密钥授权。授予对指定 KMS 密钥的控制访问权限，该密钥允许访问 AWS 大型机现代化所需的操作。有关使用授权的更多信息，请参阅《AWS Key Management Service 开发人员指南》。

  这样， AWS 大型机现代化就可以做到以下几点：

  - 调用 GenerateDataKey 生成加密的数据密钥并将其存储，因为数据密钥不会立即用于加密。
  - 调用 Decrypt 来使用存储的加密数据密钥访问加密数据。
  - 设置停用主体，以允许服务 RetireGrant。
- kms:DescribeKey— 提供客户托管的密钥详细信息，以便 AWS 大型机现代化能够验证密钥。

AWS 大型机现代化需要客户密钥策略中的kms:DescribeKey权限kms:CreateGrant和权限。 AWS 大型机现代化使用此政策为自己创建补助金。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "Enable IAM User Permissions",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::AccountId:role/ExampleRole"
        },
        "Action": [
            "kms:CreateGrant",
            "kms:DescribeKey"
        ],
```

```
        "Resource": "*"
    }]
}
```

> **Note**
>
> 前面示例`Principal`中显示的角色是您用于 AWS 大型机现代化操作的角色，例如`CreateApplication`和`CreateEnvironment`。

有关在策略中指定权限的更多信息，请参阅《AWS Key Management Service 开发人员指南》。

有关密钥访问故障排除的信更多息，请参阅《AWS Key Management Service 开发人员指南》。

为 AWS Mainframe Modernization 指定客户托管密钥

您可以为以下资源指定客户托管密钥：

- 应用程序
- 环境

创建资源时，您可以通过输入 KMS ID 来指定密钥， AWS 大型机现代化使用该ID来加密资源存储的敏感数据。

- KMS ID – 客户托管密钥的密钥标识符。输入密钥 ID、密钥 ARN、别名名称或别名 ARN。

您可以使用 AWS Management Console 或指定客户托管密钥 AWS CLI。

要在中创建运行时环境时指定您的客户托管密钥 AWS Management Console，请参阅创建 AWS Mainframe Modernization 运行时环境。要在中创建应用程序时指定您的客户托管密钥 AWS Management Console，请参阅创建 AWS Mainframe Modernization 应用程序。

要在使用创建运行时环境时添加客户托管密钥 AWS CLI，请按如下方式指定`kms-key-id`参数：

```
aws m2 create-environment –engine-type microfocus –instance-type M2.m5.large
--publicly-accessible –engine-version 7.0.3 –name test
--high-availability-config desiredCapacity=2
--kms-key-id myEnvironmentKey
```

要在使用创建应用程序时添加客户托管密钥 AWS CLI，请按如下方式指定`kms-key-id`参数：

```
aws m2 create-application –name test-application –description my description
--engine-type microfocus
--definition content="$(jq -c . raw-template.json | jq -R)"
--kms-key-id myApplicationKey
```

AWS 大型机现代化加密环境

加密上下文是一组可选的键值对，包含有关数据的其他上下文信息。

AWS KMS 使用加密上下文作为其他经过身份验证的数据来支持经过身份验证的加密。当您在加密数据的请求中包含加密上下文时，会将加密上下文 AWS KMS 绑定到加密数据。要解密数据，您必须在请求中包含相同的加密上下文。

AWS 大型机现代化加密环境

AWS 大型机现代化在与应用程序相关的所有 AWS KMS 加密操作（创建应用程序和创建部署）中使用相同的加密上下文，其中密钥是aws:m2:app，值是应用程序的唯一标识符。

Example

```
"encryptionContextSubset": {
        "aws:m2:app": "a1bc2defabc3defabc4defabcd"
}
```

使用加密上下文进行监控

使用对称客户托管密钥加密应用程序或运行时环境时，您还可以使用审计记录和日志中的加密上下文来识别客户托管密钥的使用情况。

使用加密上下文控制对客户托管式密钥的访问

您可以使用密钥策略和 IAM 策略中的加密上下文作为 conditions 来控制对您的对称客户托管密钥的访问。您还可以在授权中使用加密上下文约束。

AWS 大型机现代化在授权中使用加密上下文限制来控制对您账户或区域中客户托管密钥的访问权限。授权约束要求授权允许的操作使用指定的加密上下文。以下示例是 AWS 大型机现代化在创建应用程序时用来加密应用程序工件的赠款。

```
//This grant is retired immediately after create application finish
{
   "grantee-principal": m2.us-west-2.amazonaws.com,
   "retiring-principal": m2.us-west-2.amazonaws.com,
```

```
    "operations": [
        "GenerateDataKey"
    ]
    "condition": {
         "encryptionContextSubset": {
             “aws:m2:app”: “a1bc2defabc3defabc4defabcd”
    }
}
```

监控用于 AWS Mainframe Modernization 的加密密钥

当您在 AWS 大型机现代化资源中使用 AWS KMS 客户托管密钥时，您可以使用AWS CloudTrail或 Amazon CloudWatch Logs 来跟踪 AWS 大型机现代化向其发送的请求。 AWS KMS

针对运行时环境的示例

以下示例是DescribeKey、CreateGrant、和Decrypt监控 KMS 操作 AWS CloudTrail 的事件GenerateDataKey，这些操作由 AWS 大型机现代化调用，以访问由您的客户托管密钥加密的数据：

DescribeKey

AWS 大型机现代化使用该DescribeKey操作来验证 AWS KMS 客户和区域中是否存在与您的运行时环境相关的客户托管密钥。

以下示例事件记录了 DescribeKey 操作：

```
{
    "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:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
```

```
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T19:40:26Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-12-06T20:23:43Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.251.233.182",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "keyId": "00dd0db0-0000-0000-ac00-b0c000SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_256_GCM_SHA384",
        "clientProvidedHostHeader": "kms.us-west-2.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

CreateGrant

当您使用 AWS KMS 客户托管密钥加密运行时环境时，AWS 大型机现代化会代表您发送多个`CreateGrant`请求以执行必要的 KMS 操作。 AWS 大型机现代化创建的某些补助金在使用后会立即停用。其他授权会在您删除运行时环境时停用。

以下示例事件记录了与创建环境工作流程关联的 Lambda 执行角色的 `CreateGrant` 操作。

```
{
    "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:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T20:11:45Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T20:23:09Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "operations": [
            "Encrypt",
```

```
            "Decrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "GenerateDataKey",
            "GenerateDataKey",
            "DescribeKey",
            "CreateGrant"
        ],
        "granteePrincipal": "m2.us-west-2.amazonaws.com",
        "retiringPrincipal": "m2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

以下示例事件记录了自动扩缩组服务相关角色的 CreateGrant 操作。与创建环境工作流程关联的 Lambda 执行角色调用 CreateGrant 操作。它允许执行角色针对自动扩缩组的服务相关角色创建子授权。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
```

```
        "principalId": "AROA3YPCLM65MZFUPM4JO:EnvironmentWorkflow-alpha-
CreateEnvironmentLambda7-HfxDj5zz86tr",
        "arn": "arn:aws:sts::111122223333:assumed-role/EnvironmentWorkflow-
alpha-CreateEnvironmentLambdaS-1AU4A8VNQEEKN/EnvironmentWorkflow-alpha-
CreateEnvironmentLambda7-HfxDj5zz86tr",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/EnvironmentWorkflow-alpha-
CreateEnvironmentLambdaS-1AU4A8VNQEEKN",
                "accountId": "111122223333",
                "userName": "EnvironmentWorkflow-alpha-
CreateEnvironmentLambdaS-1AU4A8VNQEEKN"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T20:22:28Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-12-06T20:23:09Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "54.148.236.160",
    "userAgent": "aws-sdk-java/2.18.21 Linux/4.14.255-276-224.499.amzn2.x86_64
 OpenJDK_64-Bit_Server_VM/11.0.14.1+10-LTS Java/11.0.14.1 vendor/Amazon.com_Inc. md/
internal exec-env/AWS_Lambda_java11 io/sync http/Apache cfg/retry-mode/legacy",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "operations": [
            "Encrypt",
            "Decrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "GenerateDataKey",
            "GenerateDataKey",
            "DescribeKey",
            "CreateGrant"
```

```
        ],
        "granteePrincipal": "m2.us-west-2.amazonaws.com",
        "retiringPrincipal": "m2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_256_GCM_SHA384",
        "clientProvidedHostHeader": "kms.us-west-2.amazonaws.com"
    }
}
}
```

GenerateDataKey

当您为运行时环境资源启用 AWS KMS 客户托管密钥时，Auto Scaling 会创建一个用于加密与运行时环境关联的 Amazon EBS 卷的唯一密钥。它向发送`GenerateDataKey`请求 AWS KMS ，指定资源的 AWS KMS 客户托管密钥。

以下示例事件记录了 `GenerateDataKey` 操作：

```
{
    "eventVersion": "1.08",
```

```
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65EEXVIEH7D:AutoScaling",
        "arn": "arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForAutoScaling/
AutoScaling",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling",
                "accountId": "111122223333",
                "userName": "AWSServiceRoleForAutoScaling"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T20:23:16Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "autoscaling.amazonaws.com"
    },
    "eventTime": "2022-12-06T20:23:18Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "autoscaling.amazonaws.com",
    "userAgent": "autoscaling.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:ebs:id": "vol-080f7a32d290807f3"
        },
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "numberOfBytes": 64
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
```

```
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

当您访问加密的运行时环境时，Amazon EBS 会调用 `Decrypt` 操作，以便使用存储的加密数据密钥访问加密数据。

以下示例事件记录了 `Decrypt` 操作：

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ebs.amazonaws.com"
    },
    "eventTime": "2022-12-06T20:23:22Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ebs.amazonaws.com",
    "userAgent": "ebs.amazonaws.com",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "encryptionContext": {
            "aws:ebs:id": "vol-080f7a32d290807f3"
        }
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
```

```
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventCategory": "Management"
}
```

针对应用程序的示例

以下示例是 AWS 大型机现代化为访问由客户托管密钥加密的数据而调用的 KMS 操作`CreateGrant`和`GenerateDataKey`监控 KMS 操作 AWS CloudTrail 的事件：

CreateGrant

当您使用 AWS KMS 客户托管密钥加密应用程序资源时，Lambda 执行角色会代表您发送访问您账户中的 KMS 密钥的`CreateGrant`请求。 AWS 该授权允许 Lambda 执行角色使用您的客户托管密钥将客户应用程序资源上传到 Amazon S3。该授权将在应用程序创建后立即停用。

以下示例事件记录了 `CreateGrant` 操作：

```
{
    "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:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
```

```
            "attributes": {
                "creationDate": "2022-12-06T21:51:45Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T22:47:04Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "constraints": {
            "encryptionContextSubset": {
                "aws:m2:app": "a1bc2defabc3defabc4defabcd"
            }
        },
        "retiringPrincipal": "m2.us-west-2.amazonaws.com",
        "operations": [
            "GenerateDataKey"
        ],
        "granteePrincipal": "m2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
```

```
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

GenerateDataKey

当您为应用程序资源启用 AWS KMS 客户托管密钥时，Lambda 执行角色会创建一个用于加密客户数据并将其上传到亚马逊简单存储服务的密钥。Lambda 执行角色向发送GenerateDataKey请求 AWS KMS ，指定资源的 AWS KMS 客户托管密钥。

以下示例事件记录了 GenerateDataKey 操作：

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65CLCEKKC7Z:ApplicationWorkflow-alpha-
CreateApplicationVersion-CstWZUn5R4u6",
        "arn": "arn:aws:sts::111122223333:assumed-role/ApplicationWorkflow-
alpha-CreateApplicationVersion-1IZRBZYDG20B/ApplicationWorkflow-alpha-
CreateApplicationVersion-CstWZUn5R4u6",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/ApplicationWorkflow-alpha-
CreateApplicationVersion-1IZRBZYDG20B",
                "accountId": "111122223333",
                "userName": "ApplicationWorkflow-alpha-
CreateApplicationVersion-1IZRBZYDG20B"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T23:28:32Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
```

```
    "eventTime": "2022-12-06T23:29:08Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:m2:app": "a1bc2defabc3defabc4defabcd",
            "aws:s3:arn": "arn:aws:s3:::supernova-processedtemplate-111122223333-us-
west-2/111122223333/a1bc2defabc3defabc4defabcd/1/cics-transaction/ZBNKE35.so"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


针对部署的示例

以下示例是 AWS 大型机现代化为访问由客户托管密钥加密的数据而调用的 KMS 操作`CreateGrant`和`Decrypt`监控 KMS 操作 AWS CloudTrail 的事件：

CreateGrant

当您使用 AWS KMS 客户托管密钥加密部署资源时，AWS 大型机现代化将代表您发送两个`CreateGrant`请求。第一笔授权针对当前的 Lambda 执行角色进行调用 ListBatchJobScriptFiles，部署完成后立即停用。第二项授权针对的是 Amazon EC2 范围缩小实例角色，以便 Amazon EC2 可以从 Amazon S3 下载客户应用程序资源。该授权在应用程序从运行时环境中删除时即会停用。

以下示例事件记录了 `CreateGrant` 操作：

```
{
    "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:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T21:51:45Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T23:40:07Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "operations": [
            "Decrypt"
```

```
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:m2:app": "a1bc2defabc3defabc4defabcd"
            }
        },
        "granteePrincipal": "m2.us-west-2.amazonaws.com",
        "retiringPrincipal": "m2.us-west-2.amazonaws.com",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

当您访问部署时，Amazon EC2 会调用 `Decrypt` 操作，以便使用存储的加密数据密钥来解密并从 Amazon S3 下载加密的客户数据。

以下示例事件记录了 `Decrypt` 操作：

```
{
    "eventVersion": "1.08",
    "userIdentity": {
```

```
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65BSPZ37E6G:m2-hm-bqe367dxtfcpdbzmnhfzranisu",
        "arn": "arn:aws:sts::111122223333:assumed-role/
SupernovaEnvironmentInstanceScopeDownRole/m2-hm-bqe367dxtfcpdbzmnhfzranisu",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/
SupernovaEnvironmentInstanceScopeDownRole",
                "accountId": "111122223333",
                "userName": "SupernovaEnvironmentInstanceScopeDownRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T23:19:29Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T23:40:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:m2:app": "a1bc2defabc3defabc4defabcdm",
            "aws:s3:arn": "arn:aws:s3:::supernova-processedtemplate-111122223333-us-
west-2/111122223333/a1bc2defabc3defabc4defabcdm/1/cics-transaction/BBANK40P.so"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
```

```
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

了解更多信息

以下资源提供有关静态数据加密的更多信息。

- 有关 AWS Key Management Service 基本概念的更多信息，请参阅 《AWS Key Management Service 开发人员指南》。
- 有关 AWS Key Management Service的安全最佳实操的更多信息，请参阅 《AWS Key Management Service 开发人员指南》。

传输中加密

对于属于交易工作负载的交互式应用程序，终端仿真器与 TN3270 协议 AWS 的大型机现代化服务端点之间的数据交换在传输过程中未加密。如果应用程序要求在传输过程中加密，则您可能需要实现一些额外的隧道机制。

AWS 大型机现代化使用 HTTPS 来加密服务 API。 AWS 大型机现代化中的所有其他通信均受服务 VPC 或安全组以及 HTTPS 保护。 AWS 大型机现代化可传输应用程序工件、配置和应用程序数据。应用程序构件是从您的 Amazon S3 存储桶中复制的，应用程序数据也是如此。您可以使用指向 Amazon S3 的链接或通过在本地上传文件来提供应用程序配置。

传输中的基本加密是默认配置的，但不适用于 TN3270 协议。 AWS 大型机现代化对 API 端点使用 HTTPS，这些端点也是默认配置的。

AWS 大型机现代化的 Identity and Access 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 AWS 大型机现代化资源。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) 会有所不同，具体取决于您在 AWS 大型机现代化中所做的工作。

服务用户-如果您使用 AWS 大型机现代化服务完成工作，则您的管理员会为您提供所需的凭据和权限。当你使用更多 AWS 的大型机现代化功能来完成工作时，你可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 AWS Mainframe Modernization 中的特征，请参阅AWS 大型机现代化身份和访问权限疑难解答。

服务管理员 — 如果您负责公司 AWS 的大型机现代化资源，则可能拥有对 AWS 大型机现代化的完全访问权限。您的工作是确定您的服务用户应访问哪些 AWS 大型机现代化功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解贵公司如何使用 IAM 进行 AWS 大型机现代化，请参阅AWS 大型机现代化如何与 IAM 配合使用。

IAM 管理员 — 如果您是 IAM 管理员，则可能需要详细了解如何编写策略来管理对 AWS 大型机现代化的访问权限。要查看您可以在 IAM 中使用的基于身份 AWS 的大型机现代化策略示例，请参阅。大型机现代化的基于身份的 AWS 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 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （ IAM Identity Center ）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。

当您以联合身份登录时，管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证 和《IAM 用户指南》中的在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，我们建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户 和应用程序中使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果有一些特定的使

用场景需要长期凭证以及 IAM 用户，我们建议轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个用于指定一组 IAM 用户的身份。您不能使用群组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人担任。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问——要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限——IAM 用户或角色可代入 IAM 角色，以暂时获得针对特定任务的不同权限。
- 跨账户访问——您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问账户中的资源。角色是授予跨账户存取权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以担任代表您执行操作的角色。服务相关角色出现在您的中 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相关角色的权限。
- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 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。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

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。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。然后，管理员可以向角色添加 IAM policy，并且用户可以代入角色。

IAM policy 定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户群组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的 创建 IAM policy。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、群组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表(ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表 (ACL) 概述。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型授予的最大权限。

- 权限边界——权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可以为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果您在组织内启用了特征，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关组织和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略——会话策略是当以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策

略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解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。

AWS 大型机现代化如何与 IAM 配合使用

在使用 IAM 管理 AWS 大型机现代化的访问权限之前，请先了解有哪些 IAM 功能可用于 AWS 大型机现代化。

可在 AWS 大型机现代化中使用的 IAM 功能

IAM 功能	AWS 大型机现代化支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键	支持
ACL	否
ABAC（策略中的标签）	支持
临时凭证	支持
转发访问会话（FAS）	支持
服务角色	支持
服务相关角色	支持

要全面了解 AWS 大型机现代化和其他 AWS 服务如何与大多数 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。

大型机现代化的基于身份的 AWS 策略

支持基于身份的策略	是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

大型机现代化的基于身份的 AWS 策略示例

要查看 AWS 大型机现代化基于身份的策略的示例，请参阅。大型机现代化的基于身份的 AWS 策略示例

AWS 大型机现代化中的基于资源的策略

支持基于资源的策略	否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时 AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅IAM 用户指南中的 IAM 角色与基于资源的策略有何不同。

AWS 大型机现代化的政策行动

支持策略操作	支持

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。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 AWS 大型机现代化操作列表，请参阅《服务授权参考》中的“ AWS 大型机现代化定义的操作”。

AWS 大型机现代化中的策略操作在操作前使用以下前缀：

```
m2
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "m2:StartApplication",
      "m2:StopApplication"
         ]
```

也可以使用通配符 (*) 指定多个操作。例如，要指定以单词 `List` 开头的所有操作，包括以下操作：

```
"Action": "m2:List*"
```

要查看 AWS 大型机现代化基于身份的策略的示例，请参阅。大型机现代化的基于身份的 AWS 策略示例

AWS 大型机现代化的政策资源

支持策略资源	支持

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。语句必须包含 Resource 或 NotResource 元素。作为最佳实操，请使用其 Amazon 资源名称 (ARN) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

您可以使用特定 AWS 大型机现代化资源的 ARN 来识别 IAM 策略适用的资源，从而限制对这些资源的访问。有关 ARN 格式的更多信息，请参阅《AWS 一般参考》中的 Amazon 资源名称 (ARN)。

例如， AWS 大型机现代化环境具有以下 ARN。

```
"Resource": "arn:aws:m2:regionId:accountId:env/service-generated-unique-identifier"
```

AWS 大型机现代化应用程序具有以下 ARN。

```
"Resource": "arn:aws:m2:regionId:accountId:app/service-generated-unique-identifier"
```

并非所有 AWS 大型机现代化操作都支持资源级权限。对于不支持资源级权限的操作，必须使用通配符 (*)。

以下 AWS 大型机现代化操作不支持资源级权限。

```
ListApplications
            ListApplicationVersions
            ListBatchJobDefinitions
            ListBatchJobExecutions
            ListDataSetImportHistory
            ListDataSets
            ListDeployments
            ListEngineVersions
            ListEnvironments
            ListTagsForResource
```

要查看 AWS 大型机现代化资源类型及其 ARN 的列表，请参阅《服务授权参考》中的 “ AWS 大型机现代化定义的资源”。要了解您可以使用哪些操作来指定每种资源的 ARN，请参阅AWS 大型机现代化定义的操作。

要查看 AWS 大型机现代化基于身份的策略的示例，请参阅。大型机现代化的基于身份的 AWS 策略示例

AWS 大型机现代化 API 权限：操作、资源和条件参考

在编写您可附加到 IAM 身份的权限策略（基于身份的策略）时，可以使用下表作为参考。此表中包括以下内容：

- 每个 AWS 大型机现代化 API 操作
- 您可授予执行该操作的权限的对应操作
- 您可以为其授予权限的 AWS 资源

您在策略的 `Action` 字段中指定操作，并在策略的 `Resource` 字段中指定资源值。

您可以在 AWS 大型机现代化策略中使用 AWS 全局条件键来表达条件。有关 AWS 密钥的完整列表，请参阅 IAM 用户指南中的可用全局条件密钥。

> **Note**
>
> 要指定操作，请在 API 操作名称之前使用 `m2:` 前缀（例如，`m2:CreateApplication`）。

AWS 大型机现代化 API 和操作所需的权限

AWS 大型机现代化 API 操作	所需权限（API 操作）	资源
CancelBatchJobExecution		应用程序
CreateApplication	`s3:GetObject` `s3:ListBucket`	应用程序
CreateDataSetImportTask	`m2:CreateDataSetImportTask` `s3:GetObject`	应用程序

AWS 大型机现代化 API 操作	所需权限（API 操作）	资源
CreateDeployment	elasticloadbalancing:AddTags elasticloadbalancing:CreateListener elasticloadbalancing:CreateTargetGroup elasticloadbalancing:RegisterTargets	应用程序

AWS 大型机现代化 API 操作	所需权限（API 操作）	资源
CreateEnvironment	ec2:CreateNetworkInterface ec2:CreateNetworkInterfacePermission ec2:DescribeNetworkInterfaces ec2:DescribeSecurityGroups ec2:DescribeSubnets ec2:DescribeVpcAttribute ec2:DescribeVpcs ec2:ModifyNetworkInterfaceAttribute elasticfilesystem:DescribeMountTargets elasticloadbalancing:AddTags elasticloadbalancing:CreateLoadBalancer fsx:DescribeFileSystems iam:CreateServiceLinkedRole	环境

AWS 大型机现代化 API 操作	所需权限（API 操作）	资源
DeleteApplication	elasticloadbalancing:DeleteListener elasticloadbalancing:DeleteTargetGroup logs:DeleteLogDelivery	应用程序
DeleteApplicationFromEnvironment	elasticloadbalancing:DeleteListener elasticloadbalancing:DeleteTargetGroup	应用程序 环境
DeleteEnvironment	elasticloadbalancing:DeleteLoadBalancer	环境
GetApplication		应用程序
GetApplicationVersion		应用程序
GetBatchJobExecution		应用程序
GetDataSetDetails		应用程序
GetDataSetImportTask		应用程序
GetDeployment		应用程序
GetEnvironment		环境
ListApplications		*
ListApplicationVersions		*

AWS 大型机现代化 API 操作	所需权限（API 操作）	资源
ListBatchJobDefinitions		*
ListBatchJobExecutions		*
ListDataSetImportHistory		*
ListDataSets		*
ListDeployments		*
ListEngineVersions		*
ListEnvironments		*
ListTagsForResource		*
StartApplication		应用程序
StartBatchJob		应用程序
StopApplication		应用程序
TagResource		*
UntagResource		*
UpdateApplication	`s3:GetObject` `s3:ListBucket`	应用程序
UpdateEnvironment		环境

AWS 大型机现代化的策略条件密钥

支持特定于服务的策略条件键	支持

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。

在 Condition 元素（或 Condition 块）中，您可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个密钥，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

您也可以在指定条件时使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。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。

以下条件键特定于 AWS 大型机现代化

```
m2:EngineType
            m2:InstanceType
```

要查看 AWS 大型机现代化条件密钥列表，请参阅《服务授权参考》中的 “AWS 大型机现代化条件密钥”。要了解您可以使用哪些操作和资源使用条件键，请参阅AWS 大型机现代化定义的操作。

要查看 AWS 大型机现代化基于身份的策略的示例，请参阅。大型机现代化的基于身份的 AWS 策略示例

AWS Mainframe Modernization 中的访问控制列表 (ACL)

支持 ACL	否

访问控制列表(ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

基于属性的访问控制 (ABAC) 和大型机现代化 AWS

支持 ABAC（策略中的标签）	支持

基于属性的访问权限控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以向 IAM 实体（用户或角色）和许多 AWS 资源附加标签。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件密钥在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件密钥，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密钥，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

在 AWS 大型机现代化中使用临时证书

支持临时凭证	支持

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的AWS 服务 与 IAM 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的切换到角色（控制台）。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。 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 中的临时安全凭证。

AWS 大型机现代化的转发访问会话

支持转发访问会话 (FAS)	支持

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

> **Important**
>
> 这些令牌允许 AWS 大型机现代化在未经您明确同意的情况下访问客户数据；例如， AWS 大型机现代化在未获得客户明确许可的情况下部署应用程序工件以及来自 Amazon S3 存储桶的关联业务数据。您可能需要相应地更新所有合规性文档。

AWS Mainframe Modernization 的服务角色

支持服务角色	支持

服务角色是由一项服务代入、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。

AWS 大型机现代化支持活动挂钩的服务角色（交易/作业暂停或完成等）。

> **Warning**
>
> 更改服务角色的权限可能会破坏 AWS 大型机现代化功能。只有当 AWS 大型机现代化提供相关指导时，才能编辑服务角色。

在 AWS 大型机现代化中选择 IAM 角色

如果您之前创建了在 Amazon EC2 上运行的应用程序可以代入的 IAM 角色，则可以在创建启动模板或启动配置时选择此角色。 AWS 大型机现代化为您提供了可供选择的角色列表。创建这些角色时，请关

联最低权限 IAM 策略以限制对应用程序所需的特定 API 调用的访问权限，这一点非常重要。有关更多信息，请参阅《Amazon EC2 Auto Scaling 用户指南》中的在 Amazon EC2 实例上运行的应用程序的 IAM 角色。

AWS 大型机现代化的服务相关角色

支持服务相关角色	支持

服务相关角色是一种链接到的服务角色。 AWS 服务服务可以担任代表您执行操作的角色。服务相关角色出现在您的中 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相关角色的权限。

有关创建或管理 AWS 大型机现代化服务相关角色的详细信息，请参阅。使用 Mainframe Modernization 服务相关角色

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。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

大型机现代化的基于身份的 AWS 策略示例

默认情况下，用户和角色无权创建或修改 AWS 大型机现代化资源。他们也无法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的 创建 IAM 策略。

有关 AWS 大型机现代化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 “AWS 大型机现代化的操作、资源和条件密钥”。

主题

- 策略最佳实践
- 使用 AWS 大型机现代化控制台
- 允许用户查看他们自己的权限

策略最佳实践

基于身份的策略决定是否有人可以在您的账户中创建、访问或删除 AWS 大型机现代化资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- 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 托管式策略或工作职能的AWS 托管式策略。
- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定 AWS 服务的（例如）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 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 IAM policy，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，确保策略符合 IAM policy 语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，有助于制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Access Analyzer 策略验证。
- 需要多重身份验证 (MFA)-如果 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。要在调用 API 操作时需要 MFA，请将 MFA 条件添加到策略中。有关更多信息，请参阅《IAM 用户指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_configure-api-require.html中的配置受 MFA 保护的 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 中的安全最佳实操。

使用 AWS 大型机现代化控制台

要访问 AWS 大型机现代化控制台，您必须拥有一组最低权限。这些权限必须允许您在中列出和查看有关 AWS 大型机现代化资源的详细信息。 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 AWS 大型机现代化控制台，还要将 AWS 大型机现代化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。

允许用户查看他们自己的权限

该示例说明了如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 大型机现代化身份和访问权限疑难解答

使用以下信息来帮助您诊断和修复在使用 AWS 大型机现代化和 IAM 时可能遇到的常见问题。

主题

我无权执行 iam：PassRole

如果您收到一条错误消息，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移交给 AWS 大型机现代化。

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 `marymajor` 的 IAM 用户尝试使用控制台在 AWS Mainframe Modernization 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 AWS 账户 访问我的 AWS 大型机现代化资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 AWS 大型机现代化是否支持这些功能，请参阅AWS 大型机现代化如何与 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 角色与基于资源的策略有何不同。

使用 Mainframe Modernization 服务相关角色

AWS Mainframe Modernization 使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，与 Mainframe Modernization 直接关联。服务相关角色由 Mainframe Modernization 预定义，具有服务代表您调用其它 AWS 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Mainframe Modernization，因为您不必手动添加必要的权限。Mainframe Modernization 定义其服务相关角色的权限，除非另外定义，否则只有 Mainframe Modernization 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这将保护您的 Mainframe Modernization 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的 AWS 服务，并查找 Service-linked roles（服务相关角色）列中显示为 Yes（是）的服务。选择是，可转到查看该服务的服务相关角色文档的链接。

Mainframe Modernization 的服务相关角色权限

Mainframe Modernization 使用名为 AWSServiceRoleForAWSM2 的服务相关角色，将网络配置为连接到您的 VPC 并访问文件系统等资源。

AWSServiceRoleForAWSM2 服务相关角色信任以下服务代入该角色：

- `m2.amazonaws.com`

名为 AWSM2ServicePolicy 的角色权限策略，允许 Mainframe Modernization 对指定资源完成以下操作：

- 创建、删除、描述和附加权限到 Mainframe Modernization 环境的 Amazon EC2 网络接口，以便建立与客户 VPC 的连接。

- 从 Elastic Load Balancing 中注册或注销条目，这是客户连接到 Mainframe Modernization 环境的方式。
- 描述 Amazon EFS 或 Amazon FSx 文件系统（如果使用）。
- 从运行时环境向客户的 CloudWatch 发送指标。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "ec2:DescribeSubnets",
    "ec2:CreateNetworkInterface",
    "ec2:DeleteNetworkInterface",
    "ec2:DescribeNetworkInterfaces",
    "ec2:CreateNetworkInterfacePermission",
    "ec2:ModifyNetworkInterfaceAttribute"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "elasticfilesystem:DescribeMountTargets"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "elasticloadbalancing:RegisterTargets",
    "elasticloadbalancing:DeregisterTargets"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "fsx:DescribeFileSystems"
   ],
   "Resource": "*"
```

```
  },
  {
   "Effect": "Allow",
   "Action": [
    "cloudwatch:PutMetricData"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "cloudwatch:namespace": [
      "AWS/M2"
     ]
    }
   }
  }
 ]
}
```

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

创建 Mainframe Modernization 服务相关角色

您无需手动创建服务相关角色。当您使用AWS Management Console、AWS CLI 或 AWS API 创建运行时环境时，Mainframe Modernization 会为您创建服务相关角色。

如果您删除此服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建运行时环境时，Mainframe Modernization 会再次为您创建服务相关角色。

编辑 Mainframe Modernization 服务相关角色

Mainframe Modernization 不允许您编辑 AWSServiceRoleForAWSM2 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除 Mainframe Modernization 服务相关角色

如果不再需要使用某个需要服务相关角色的特征或服务，建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，必须先清除服务相关角色的资源，然后才能手动删除它。

> **Note**
>
> 如果在您试图删除资源时，Mainframe Modernization 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

删除 AWSServiceRoleForAWSM2 使用的 Mainframe Modernization 资源

- 删除 Mainframe Modernization 中的运行时环境 在删除环境本身之前，请务必从环境中删除应用程序。

要使用 IAM 手动删除服务相关角色

使用 IAM 控制台、AWS CLI 或 AWS API 删除 AWSServiceRoleForAWSM2 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Mainframe Modernization 服务相关角色的受支持区域

Mainframe Modernization 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 AWS 区域和终端节点。

AWS Mainframe Modernization 合规性验证

作为多个 AWS 合规性计划的一部分，第三方审计员将评测 AWS Mainframe Modernization 的安全性和合规性。其中包括 SOC、PCI、FedRAMP、HIPAA 及其它。

有关特定合规性计划范围内的 AWS 服务列表，请参阅合规性计划范围内的 AWS 服务。有关常规信息，请参阅AWS合规性计划。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅下载 AWS Artifact 中的报告。

您使用 AWS Mainframe Modernization 的合规性责任取决于您数据的敏感度、贵公司的合规性目标以及适用的法律法规。AWS 提供以下资源来帮助满足合规性：

- 安全性与合规性 Quick Start 指南 - 这些部署指南讨论了架构注意事项，并提供了在 AWS 上部署基于安全性和合规性的基准环境的步骤。
- 《设计符合 HIPAA 安全性和合规性要求的架构》白皮书 – 此白皮书介绍公司如何使用AWS创建符合 HIPAA 标准的应用程序。

- AWS合规性资源 – 此业务手册和指南集合可能适用于您的行业和位置。
- 《AWS Config 开发人员指南》中的使用规则评估资源 – AWS Config；评测您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub – 此AWS服务提供了AWS中安全状态的全面视图，可帮助您检查是否符合安全行业标准和最佳实操。

AWS Mainframe Modernization 弹性

AWS 全球基础设施围绕 AWS 区域和可用区构建。区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅 AWS全球基础设施。

AWS Mainframe Modernization 中的基础设施安全性

作为一项托管式服务，AWS Mainframe Modernization 受 AWS 全球网络安全保护。有关 AWS 安全服务以及 AWS 如何保护基础架构的信息，请参阅 AWS 云安全。要按照基础设施安全最佳实操设计您的 AWS 环境，请参阅《安全性支柱 AWS Well-Architected Framework》中的 基础设施保护。

您可以使用 AWS 发布的 API 调用通过网络访问 Mainframe Modernization。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（Ephemeral Diffie-Hellman）或 ECDHE（Elliptic Curve Ephemeral Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service (AWS STS) 生成临时安全凭证来对请求进行签名。

使用接口端点 (AWS PrivateLink) 访问 AWS Mainframe Modernization

您可以使用 AWS PrivateLink 在您的 VPC 和 AWS Mainframe Modernization 之间创建私有连接。您可以像在您的 VPC 中一样访问 Mainframe Modernization，而无需使用互联网网关、NAT 设

备、VPN 连接或 AWS Direct Connect 连接。VPC 中的实例不需要公有 IP 地址即可访问 Mainframe Modernization。

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。这些请求方托管的网络接口，用作发往 Mainframe Modernization 的流量的入口点。

有关更多信息，请参阅《AWS PrivateLink 指南》中的通过 AWS PrivateLink 访问 AWS 服务。

Mainframe Modernization 注意事项

在为 Mainframe Modernization 设置接口端点之前，请先查看《AWS PrivateLink 指南》中的注意事项。

Mainframe Modernization 支持通过接口端点调用其所有 API 操作。

为 Mainframe Modernization 创建接口端点

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 Mainframe Modernization 创建接口端点。有关更多信息，请参阅《AWS PrivateLink 指南》中的创建接口端点。

使用以下服务名称为 Mainframe Modernization 创建接口端点：

```
com.amazonaws.region.m2
```

如果为接口端点启用私有 DNS，则可使用其默认区域 DNS 名称向 Mainframe Modernization 发出 API 请求。例如，`m2.us-east-1.amazonaws.com`。

为接口端点创建端点策略

端点策略是一种 IAM 资源，您可以将其附加到接口端点。默认端点策略提供通过接口端点访问 Mainframe Modernization 的完全访问权限。要控制允许从 VPC 访问 Mainframe Modernization 的权限，请将自定义端点策略附加到接口端点。

端点策略指定以下信息：

- 可执行操作的主体（AWS 账户、用户和 IAM 角色）。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《AWS PrivateLink 指南》中的使用端点策略控制对服务的访问权限。

示例：Mainframe Modernization 操作的 VPC 端点策略

以下是自定义端点策略的示例。附加到接口端点时，此策略其会向所有主体授予针对所有资源列出的 Mainframe Modernization 操作的访问权限。

```
//Example of an endpoint policy where access is granted to the
//listed AWS Mainframe Modernization actions for all principals on all resources
{"Statement": [
      {"Principal": "*",
         "Effect": "Allow",
         "Action": [
            "m2:ListApplications",
            "m2:ListEnvironments",
            "m2:ListDeployments"
         ],
         "Resource":"*"
      }
   ]
}

//Example of an endpoint policy where access is denied to all the
//AWS Mainframe Modernization CREATE actions for all principals on all resources
{"Statement": [
      {"Principal": "*",
         "Effect": "Deny",
         "Action": [
            "m2:Create*"
         ],
         "Resource":"*"
      }
   ]
}
```

监控 AWS 大型机现代化

监控是维护 AWS 大型机现代化和您的其他 AWS 解决方案的可靠性、可用性和性能的重要组成部分。AWS 提供以下监控工具，用于监控 AWS 大型机现代化情况、报告问题并在适当时自动采取行动：

- 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，创建自定义的控制平面，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。
- Amazon CloudWatch Logs 允许您监控、存储和访问来自 Amazon 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。
- 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。

使用 A AWS mazon 监控大型机现代化 CloudWatch

您可以使用监控 AWS 大型机现代化 CloudWatch，它会收集原始数据并将其处理为可读的近乎实时的指标。这些统计数据会保存 15 个月，从而使您能够访问历史信息，并能够更好地了解您的 Web 应用程序或服务的执行情况。此外，可以设置用于监测特定阈值的警报，并在达到相应阈值时发送通知或执行操作。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

下表列出了 AWS 大型机现代化的指标和维度。这些指标的命名空间为 `AWS/M2`。

运行时环境指标

指标	描述
CPUUtilization	环境中实例的 CPU 利用率。 维度：environmentId 单位：百分比

指标	描述
	有效统计数据：Average、Minimum、Maximum
InboundNetworkThroughput	环境中实例的入站网络吞吐量。 维度：environmentId 单位：字节/秒 有效统计数据：Average、Minimum、Maximum
MemoryUtilization	环境中实例的内存利用率。 维度：environmentId 单位：百分比 有效统计数据：Average、Minimum、Maximum
OutboundNetworkThroughput	环境中实例的出站网络吞吐量。 维度：environmentId 单位：字节/秒 有效统计数据：Average、Minimum、Maximum

应用程序指标

指标	描述
BatchJobCompletedCount	在时间间隔内完成的作业数。 该指标适用于 Micro Focus 和 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：计数

指标	描述
	有效统计数据：Sum
BatchJobFailedCount	在时间间隔内失败的作业数。 该指标适用于 Micro Focus 和 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：计数 有效统计数据：Sum
JvmMemoryFree	Java 虚拟机当前未使用的可用内存。 此指标仅适用于 AWS Blu Age 运行时引擎。适用于 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：字节 有效统计数据：Average、Minimum、Maximum
JvmMemoryMax	Java 虚拟机允许的最大内存量。 此指标仅适用于 AWS Blu Age 运行时引擎。适用于 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：字节 有效统计数据：Average、Minimum、Maximum

指标	描述
JvmMemoryUsed	Java 虚拟机已经使用的内存量。 此指标仅适用于 AWS Blu Age 运行时引擎。适用于 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：字节 有效统计数据：Average、Minimum、Maximum
ProcessesActiveCount	正在处理请求的活跃并发服务执行进程的数量。 此指标仅可用于 Micro Focus 运行时引擎。 维度：applicationId 单位：计数 有效统计数据：Sum
SessionCount	应用程序的 HTTP 会话数。 此指标仅适用于 AWS Blu Age 运行时引擎。适用于 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：计数 有效统计数据：Average、Minimum、Maximum

指标	描述
SharedMemoryFree	Enterprise Server 可用的内存，用于存储运行事务和作业所需的所有信息。 此指标仅可用于 Micro Focus 运行时引擎。 维度：applicationId 单位：计数 有效统计数据：Average、Minimum、Maximum
ThreadActiveCount	正在处理请求的引擎线程数。 此指标仅适用于 AWS Blu Age 运行时引擎。适用于 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：计数 有效统计数据：Average、Minimum、Maximum
TransactionCompletedCount	在时间间隔内提交的事务数。 该指标适用于 Micro Focus 和 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：计数 有效统计数据：Sum

指标	描述
TransactionFailedCount	在时间间隔内失败的事务数。 该指标适用于 Micro Focus 和 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：计数 有效统计数据：Sum
TransactionResponseTime	从用户发送请求时起，到应用程序指示请求已完成的时间。 该指标适用于 Micro Focus 和 AWS Blu Age 3.7.0 及更高版本。 维度：applicationId 单位：毫秒 有效统计数据：Average、Minimum、Maximum

维度

维度	描述
applicationId	此维度筛选指标，应用到按 ID 识别的应用程序。
environmentId	此维度筛选指标，应用到按 ID 识别的环境。

使用 AWS CloudTrail 对 AWS Mainframe Modernization API 调用进行日志记录

AWS Mainframe Modernization 与 AWS CloudTrail 进行了集成，该服务针对 AWS Mainframe Modernization 中由用户、角色或 AWS 服务采取的操作提供记录。CloudTrail 将 AWS Mainframe Modernization 的所有 API 调用作为事件捕获。捕获的调用包含来自 AWS Mainframe Modernization 控制台的调用和对 AWS Mainframe Modernization API 操作的代码调用。如果创建跟踪，则可以将 CloudTrail 事件持续传送到 Amazon S3 存储桶（包括 AWS Mainframe Modernization 的事件）。如果您不配置跟踪，则仍可在 CloudTrail 控制台中的事件历史记录中查看最新事件。使用 CloudTrail 收集的信息，您可以确定向 AWS Mainframe Modernization 发出了什么请求、发出请求的 IP 地址、请求方、请求时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息，请参阅 AWS CloudTrail 用户指南。

CloudTrail 中的 AWS Mainframe Modernization 信息

在您创建 AWS 账户时，将在该账户上启用 CloudTrail。当 AWS Mainframe Modernization 中发生活动时，该活动将记录在 CloudTrail 事件中，并与其它 AWS 服务事件一同保存在事件历史记录中。您可以在 AWS 账户中查看、搜索和下载最新事件。有关更多信息，请参阅使用 CloudTrail 事件历史记录查看事件。

要持续记录 AWS 账户中的事件（包括 AWS Mainframe Modernization 的事件），请创建跟踪。通过跟踪，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪时，此跟踪应用于所有 AWS 区域。此跟踪在 AWS 分区中记录所有区域中的事件，并将日志文件传送至您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关更多信息，请参阅下列内容：

- 创建跟踪概览
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件
- 从多个账户接收 CloudTrail 日志文件

所有 AWS Mainframe Modernization 操作均由 CloudTrail 进行日志记录，并包含在《AWS Mainframe Modernization API 参考》中。例如，对 `CreateApplication`、`CreateEnvironment` 和 `CreateDeployment` 操作的调用会在 CloudTrail 日志文件中生成条目。

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。
- 请求是否由其它 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

了解 AWS Mainframe Modernization 日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日记账条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

下面的示例显示了一个 CloudTrail 日志条目，该条目说明了 `CreateApplication` 操作。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAII6WZTHGYAEXAMPLE",
        "arn": "arn:aws:sts::444455556666:assumed-role/Admin/Mary_Major",
        "accountId": "444455556666",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAII6WZTHGYAEXAMPLE",
                "arn": "arn:aws:iam::444455556666:role/Admin",
                "accountId": "444455556666",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-06-01T20:38:22Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-06-01T20:40:39Z",
```

```
    "eventSource": "m2.amazonaws.com",
    "eventName": "CreateApplication",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.196.65",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:91.0) Gecko/20100101
 Firefox/91.0",
    "requestParameters": {
        "clientToken": "1abc23de-f45g-6789-h01i-jkl2m3456789",
        "name": "MyApp",
        "description": "",
        "engineType": "microfocus",
        "definition": {
            "content": "{}"
        },
        "tags": {}
    },
    "responseElements": {
        "applicationVersion": 1,
        "Access-Control-Expose-Headers": "x-amzn-RequestId,x-amzn-ErrorType,x-amzn-
ErrorMessage,Date",
        "applicationArn": "arn:aws:m2:us-east-1:444455556666:app/
lsfhmwhw7fffrosff2lncwqcua",
        "applicationId": "lsfhmwhw7fffrosff2lncwqcua"
    },
    "requestID": "36982d38-fcde-4bfe-a89a-7bd78d43c926",
    "eventID": "d7f0fc36-46ae-4157-9a79-c79f385fda98",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "444455556666",
    "eventCategory": "Management"
}
```

故障排除

使用本节中的信息来帮助您使用 AWS Blu Age 和 Micro Focus 引擎对 AWS Mainframe Modernization 应用程序和运行时环境中的常见错误进行故障排除。

主题

错误：等待解锁数据集名称时超时

- 引擎：AWS Blu Age
- 组件：Blusam

如果您在使用 AWS Blu Age 引擎并在具有高可用性模式的环境中运行AWS的大型机现代化应用程序的 Amazon CloudWatch 日志中看到此错误，则表示另一个应用程序锁定了共享数据集。通常，如果其他应用程序崩溃或以其他方式失败并且没有释放锁定，就会发生这种情况。

此错误是如何发生的

应用程序 `example-app-1` 尝试锁定写入操作的记录 `example-record-1`。此操作同时创建了对数据集 `example-dataset-1`（包含 `example-record-1`）的锁定和对 `example-record-1` 自身的锁定。现在，另一个应用程序 `example-app-2` 尝试锁定相同的记录 `example-record-1`。数据集和记录已被锁定，因此 `example-app-2` 等待锁定释放。如果 `example-app-1` 崩溃，则对数据集 `example-dataset-1` 的锁定仍然存在，这会导致 `example-app-2` 取消其写入尝试并引发超时异常。这种死锁情况使所有应用程序都无法访问 `example-dataset-1`。

您如何了解是否遇到了此种情况？

查找失败的应用程序，并检查其是否使用错误消息中提及的相同数据集。检查应用程序是否在具有高可用性模式的运行时环境中运行。引发超时异常的应用程序无法继续运行，并将显示 `Failed` 状态。

您能做什么？

要立即解决此问题，您可以强制解除锁定。为了防止将来发生类似的情况，您可以配置两个参数来控制 Blusam 自动修复机制。

强制释放锁定

Blusam 锁管理器使用 Amazon f ElastiCache or Redis 在应用程序之间提供共享锁。要解除锁定 ElastiCache，请使用 Redis CLI 实用工具。您无法删除单个记录锁定，而是必须从拥有的数据集中移除所有锁定。完成以下步骤：

1. ElastiCache 使用以下命令连接到您的：

   ```
   redis-cli -h hostname -p port
   ```

   你可以在 ElastiCache 主机 ElastiCache 中找到你的详细信息，网址为 https://console.aws.amazon.com/elasticache/。
2. 输入您的密码。
3. 输入要运行的命令，如下所示：

命令	用途
KEYS *	获取所有现有密钥。
KEYS *YOUR_DATASET_NAME*	获取数据集锁定密钥。
DEL *THE_RETURNED_KEY*	删除数据集锁
FLUSHDB	清理整个 Redis。 **Warning** Redis 缓存中的所有数据都将丢失。如果 Redis 用于其他目的，例如处理 http 会话，您可能不希望使用 FLUSHDB。

配置 Blusam 自动修复机制

Blusam 锁定管理器包括自动修复机制，可防止数据集或记录出现死锁。您可以调整应用程序定义 (`application-main.yml`) 中的以下参数来配置自动修复机制：

- `locksDeadTime`：指应用程序可以持有锁定的最长时间。经过此时间后，该锁定声明为过期并立即释放。`locksDeadTime` 值以毫秒为单位，默认值为 1000。
- `locksCheck`：定义了用于检查锁定的 Blusam 锁定管理器策略。所有 Blusam 锁定 ElastiCache 都有时间戳并有到期时间。`locksCheck` 参数值决定是否移除过期的锁。
  - `off`：任何时候均不会执行任何检查。可能发生死锁。（不推荐使用）
  - `reboot`：在 AWS Mainframe Modernization 运行时环境中运行的 AWS Mainframe Modernization 应用程序实例启动或重新启动时，执行检查。所有过期的锁定将立即释放。（默认值）
  - `timeout`：在 AWS Mainframe Modernization 运行时环境中运行的 AWS Mainframe Modernization 应用程序实例启动或重新启动时，或是在尝试锁定数据集期间发生了超时到期时，执行检查。过期的锁定将立即释放。

有关 AWS Blu Age 应用程序的应用程序定义的更多信息，请参阅 AWS Blu Age 应用程序定义示例。

Blusam 锁定管理器

在使用高可用性模式的 AWS Mainframe Modernization 运行时环境中，一个 AWS Blu Age 应用程序可能会部署多次。对于处理 Blusam 数据集的应用程序，可能会出现并发访问问题。Blusam 锁管理器可确保数据完整性，并通过使用在应用程序之间提供共享锁来管理对记录和数据集的读写权限。ElastiCache这种机制允许多个应用程序同时读取记录，并确保一次只有一个应用程序写入记录。

写入锁定

要更新或删除特定记录，应用程序必须先锁定拥有该记录的数据集，然后锁定记录本身。当记录被锁定时，数据集的锁定就会被释放，同一数据集中的其他记录可供使用。更新或删除操作完成后，保持的记录锁定即会被释放。如果定义的应用程序策略允许等待释放，则一次只有一个应用程序可以更新记录，这将阻止其他应用程序在锁定被释放之前读取或写入。

读取锁定

只要记录或数据集没有写入锁定，多个应用程序即可以同时读取相同的记录。要锁定写入操作的记录，必须释放所有读取锁定。

> **Note**
>
> Blusam 锁定管理器使用相同的锁定机制处理来自给定应用程序中多个线程的访问。

无法访问应用程序的 URL

- 引擎：AWS Blu Age 和 Micro Focus
- 组件：应用程序

如果您无法访问您创建并部署到 AWS Mainframe Modernization 运行时环境的正在运行的 AWS Mainframe Modernization 应用程序的 URL，则可能需要在与运行时环境关联的安全组上配置入站规则。

此错误是如何发生的

创建运行时环境时，您提供的安全组（包括默认安全组）必须具有配置为允许流量从 VPC 外部到已部署应用程序的入站规则（如果您想允许此类访问）。

您如何了解是否遇到了此种情况？

应用程序成功启动且运行正常，但您无法使用其 URL 连接到该应用程序。

您能做什么？

检查与运行时环境关联的 Amazon VPC 安全组是否允许流量通过相应的应用程序端口进入该环境。要查看安全组规则，请完成以下步骤：

1. 打开 AWS Mainframe Modernization 控制台，网址为 https://console.aws.amazon.com/m2/。
2. 在左侧导航中，选择环境。
3. 选择托管您要连接到的应用程序的运行时环境。
4. 选择配置。
5. 在安全与网络下面，选择安全组。单击链接可在 Amazon VPC 控制台中打开安全组的详细信息。
6. 如有必要，请选择编辑入站规则并添加以下规则（如果尚不存在）：

   Type

   自定义 TCP

端口

8196 或与应用程序定义中指定的侦听器属性相匹配的端口。有关更多信息，请参阅步骤 2：创建应用程序定义。

来源

您调用应用程序的 IP 地址。您可以从下拉列表中选择 myIP。如果仍然存在超时问题，请尝试选择任何位置 IPV4 或任何位置 IPV6。在安全组上添加入站规则后，请务必停止应用程序并将其重新启动。

有关更多信息，请参阅《Amazon VPC 用户指南》中的使用安全组规则。

AWS Blu Insights 无法从控制台打开

- 引擎：AWS Blu Age
- 组件：Blu Insights

您尝试从 AWS Mainframe Modernization 控制台访问 Blu Insights 时，Blu Insights 无法打开，新选项卡会立即关闭。

此错误是如何发生的

您用于访问 Blu Insights 的角色没有足够的权限。

您能做什么？

将一个 IAM 策略附加到该角色以允许其访问 Blu Insights。请确保该策略至少包含以下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "m2:GetSignedBluinsightsUrl"
            ],
            "Resource": "arn:aws:m2:region:account:*"
            }
```

```
    ]
}
```

请确保将 `region` 和 `account` 替换为正确的 AWS 区域和 AWS 账户。

《AWS Mainframe Modernization 用户指南》的文档历史记录

下表介绍了 AWS Mainframe Modernization 的文档版本。

变更	说明	日期
更新 Micro Focus 托管运行时教程	本教程介绍如何使用 Micro Focus 运行时引擎在AWS大型机现代化托管运行时环境中部署和运行 CardDemo 示例应用程序。	2024年2月5日
AWSBlu Age 运行时和现代化工具 3.9.0 版本的发布说明。	此版本的 AWS Blu Age Runtime 和现代化工具侧重于整个产品的多项横向增强，旨在提高高可用性架构的性能，以及将任务执行提升到更高水平的新功能。	2023 年 12 月 18 日
在大型机与 AWS 之间传输文件	发布了将文件从源大型机传输到 AWS 的新功能。	2023 年 11 月 27 日
管理应用程序事务	发布的新功能，用于显示和编辑 AWS Mainframe Modernization 应用程序事务。	2023 年 10 月 16 日
AWS Blu Age 运行时和现代化工具版本 3.6.0 的发布说明。	此版本的 AWS Blu Age 运行时和现代化工具为 zOS 和 AS400 传统迁移提供了新功能，主要面向扩展 CICS 支持机制、补充 JCL 功能、优化并发和高容量功能的性能以及添加功能。 multi-data-source	2023 年 8 月 4 日
现在，您可以在应用程序停止时部署其新版本。	以前，必须删除已部署的应用程序版本才能部署其新版本。	2023 年 7 月 26 日

	现在，您只需停止已部署的版本即可部署新版本。	
打包的 AWS Blu Age 运行时更便于 Amazon EC2 部署	AWS Mainframe Modernization 和 AWS Blu Age 运行时现在可更灵活地用于配置完整的堆栈并在您 AWS 账户 的 Amazon EC2 实例上进行部署。	2023 年 7 月 6 日
单点登录到 AWSAWS Blu Age Blu Insights。	AWSAWS Blu Age Blu Insights 可从 AWS Management Console 通过单点登录。	2023 年 3 月 31 日
GA 版本	《AWS Mainframe Modernization 用户指南》的 GA 版本。	2022 年 6 月 8 日
初始版本	《AWS Mainframe Modernization 用户指南》的初始版本（公开预览版）。	2021 年 11 月 30 日

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。