使用和管理 vRealize Automation Code Stream

2022 年 12 月 14 日

vRealize Automation 8.5

vmware®

您可以从 VMware 网站下载最新的技术文档：

https://docs.vmware.com/cn/。

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

威睿信息技术（中国）有限公司
北京办公室
北京市
朝阳区新源南路 8 号
启皓北京东塔 8 层 801
www.vmware.com/cn

上海办公室
上海市
淮海中路 333 号
瑞安大厦 804-809 室
www.vmware.com/cn

广州办公室
广州市
天河路 385 号
太古汇一座 3502 室
www.vmware.com/cn

版权所有 © 2022 VMware, Inc. 保留所有权利。 版权和商标信息

目录

1 什么是 vRealize Automation Code Stream 及其工作原理 5

2 设置以对发布流程进行建模 9

如何添加项目 12

如何管理用户访问和批准 13

用户操作和批准是什么 20

3 创建并使用管道 22

如何运行管道和查看结果 24

提供了哪些任务类型 29

如何在管道中使用变量绑定 32

如何在条件任务中使用变量绑定来运行或停止管道 41

绑定管道任务时可以使用哪些变量和表达式 43

如何发送有关我的管道的通知 59

管道任务失败时如何创建 Jira 票证 61

如何回滚部署 64

4 规划本地构建、集成和交付代码 69

配置管道工作区 69

在使用智能管道模板之前计划 CICD 本机构建 71

在使用智能管道模板之前计划 CI 本机构建 76

在使用智能管道模板之前计划 CD 本机构建 77

在手动添加任务之前规划 CICD 本地构建 79

计划回滚 84

5 教程 86

如何持续将来自 GitHub 或 GitLab 存储库的代码集成到管道中 87

如何自动执行从 YAML 云模板部署的应用程序的发布 91

如何自动执行应用程序到 Kubernetes 集群的发布 97

如何将应用程序部署到蓝绿部署 105

如何集成自己的生成工具、测试工具和部署工具 109

如何在下一个任务中使用云模板任务的资源属性 119

如何使用 REST API 与其他应用程序集成 123

如何利用代码形式的管道 127

6 连接到端点 132

什么是端点 132

如何与 Jenkins 集成 134
如何与 Git 集成 140
如何与 Gerrit 集成 142
如何与 vRealize Orchestrator 集成 145

7 触发管道 150
如何使用 Docker 触发器运行持续交付管道 150
如何使用 Git 触发器运行管道 158
如何使用 Gerrit 触发器运行管道 164

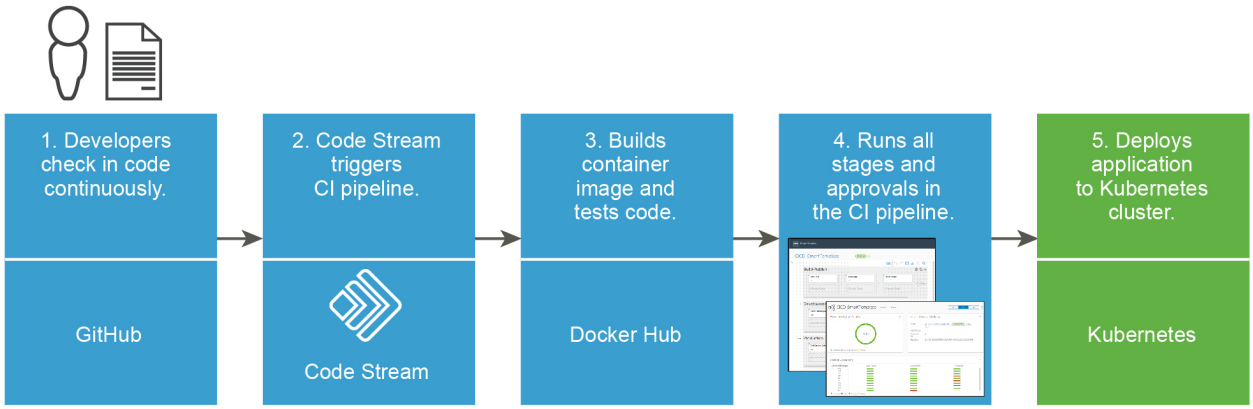
8 监控管道 171
管道仪表板显示的内容 171
如何使用自定义仪表板跟踪关键绩效指标 174

9 了解更多 177
搜索是什么 177
供管理员和开发人员使用的更多资源 181

什么是 vRealize Automation Code Stream 及其工作原理

1

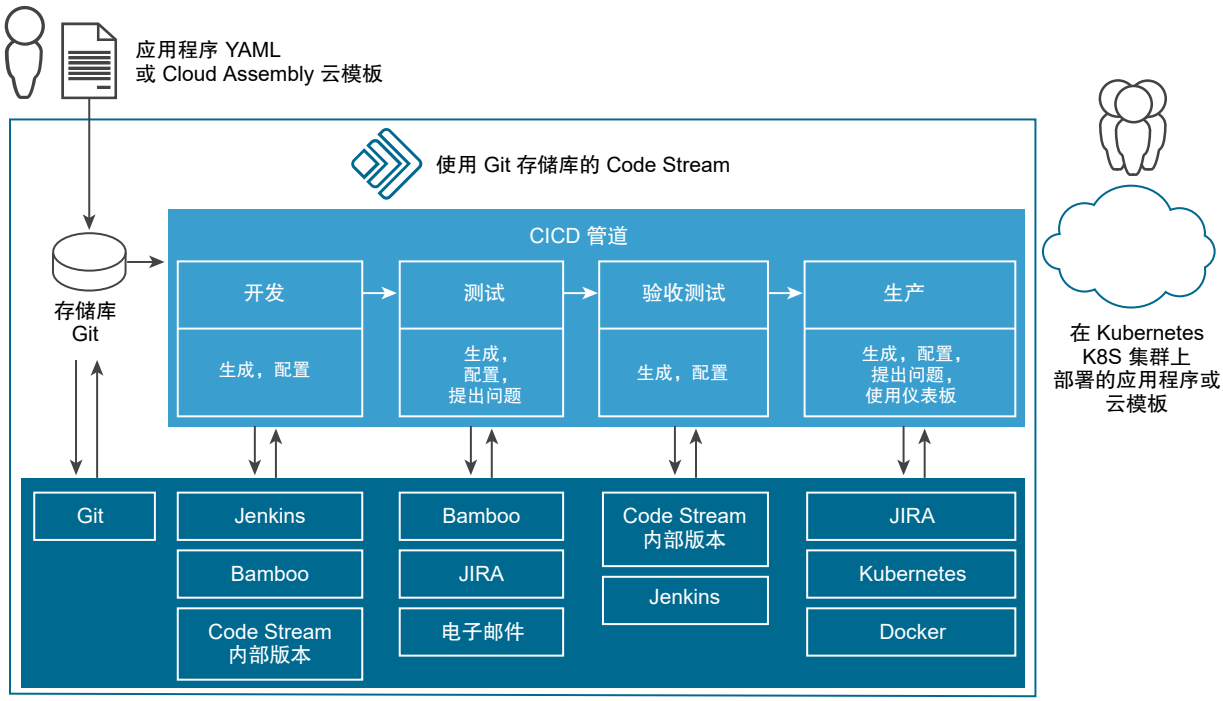
vRealize Automation Code Stream™ 是一个持续集成和持续交付 (CICD) 工具。通过创建管道以在 DevOps 生命周期中对软件发布流程进行建模，可以构建快速、持续交付软件的代码基础架构。



使用 vRealize Automation Code Stream 交付软件时，您可以集成 DevOps 生命周期的两个最重要部分：发布流程和开发人员工具。在初始设置（将 vRealize Automation Code Stream 与现有开发工具集成）之后，管道会自动执行整个 DevOps 生命周期。

从 vRealize Automation 8.2 开始，蓝图称为 VMware Cloud Templates。

可以创建一个用于生成、测试和发布软件的管道。 vRealize Automation Code Stream 使用该管道将软件从源代码存储库向前推进，执行测试，然后部署到生产环境中。



有关规划持续集成和持续交付管道的详细信息，请参见第 4 章 在 vRealize Automation Code Stream 中规划本地构建、集成和交付代码 。

vRealize Automation Code Stream 管理员如何使用 vRealize Automation Code Stream

作为管理员，您可以创建端点并确保工作实例对开发人员可用。您可以创建、触发和管理管道，还可以执行其他操作。您具有 `Administrator` 角色，如如何管理 vRealize Automation Code Stream 中的用户访问和批准中所述。

表 1-1. vRealize Automation Code Stream 管理员如何支持开发人员

要支持开发人员...	您可以执行的操作...
提供和管理环境。	为开发人员创建环境，以测试和部署其代码。 ■ 跟踪状态并发送电子邮件通知。 ■ 通过确保开发人员的环境不间断运行，使开发人员能够高效工作。 要了解详细信息，请参见供 vRealize Automation Code Stream 管理员和开发人员使用的更多资源。 另请参见 第 5 章 vRealize Automation Code Stream 使用教程。
提供端点。	确保开发人员具有可连接到其管道的端点的工作实例。
提供与其他服务的集成。	确保与其他服务的集成正常工作。 要了解详细信息，请参见 vRealize Automation 文档。
创建管道。	创建对发布流程进行建模的管道。 要了解详细信息，请参见第 3 章 在 vRealize Automation Code Stream 中创建并使用管道。

表 1-1. vRealize Automation Code Stream 管理员如何支持开发人员 （续）

要支持开发人员...	您可以执行的操作...
触发管道。	确保管道在事件发生时运行。 ■ 要在创建或更新生成工件时便触发独立的持续交付 (CD) 管线，请使用 Docker 触发器。 ■ 要在开发人员提交对其代码所做的更改时触发管道，请使用 Git 触发器。 ■ 要在开发人员审阅代码、执行合并等操作时触发管道，请使用 Gerrit 触发器。 ■ 要在创建或更新生成工件时运行独立的持续交付 (CD) 管道，请使用 Docker 触发器。 要了解详细信息，请参见第 7 章 在 vRealize Automation Code Stream 中触发管道。
管理管道和批准。	随时了解管道的最新信息。 ■ 查看管道状态，并了解运行管道的用户。 ■ 查看对管道执行的批准，并管理活动和非活动管道执行的批准。 要了解详细信息，请参见 vRealize Automation Code Stream 中的用户操作和批准是什么。 另请参见如何在 vRealize Automation Code Stream 中使用自定义仪表板跟踪管道的关键绩效指标。
监控开发人员环境。	创建用于监控管道状态、趋势、衡量指标和关键指标的自定义仪表板。使用自定义仪表板监控开发人员环境中通过或未通过的管道。还可以识别和报告未充分使用的资源，并释放资源。 您还可以查看： ■ 管道在成功之前的运行时间。 ■ 管道等待批准的时间，并通知必须批准该管道的用户。 ■ 失败最频率的阶段和任务。 ■ 运行时间最长的阶段和任务。 ■ 开发团队正在进行的发布。 ■ 已成功部署和发布的应用程序。 要了解详细信息，请参见第 8 章 监控 vRealize Automation Code Stream 中的管道。
对问题进行故障排除。	对开发人员环境中的管道失败状况进行故障排除并加以解决。 ■ 发现并解决持续集成和持续交付环境 (CICD) 中的问题。 ■ 使用管道仪表板并创建自定义仪表板以查看更多内容。请参见第 8 章 监控 vRealize Automation Code Stream 中的管道。 另请参见第 2 章 设置 vRealize Automation Code Stream 以对发布流程进行建模。

vRealize Automation Code Stream 是 vRealize Automation 的一部分。 vRealize Automation Code Stream 集成后，具有以下优势：

- 使用 vRealize Automation Cloud Assembly 部署云模板。
- 使用 vRealize Automation Service Broker 从目录中获取云模板。

要了解您可以执行的其他操作，请参见 VMware vRealize Automation 文档。

开发人员如何使用 vRealize Automation Code Stream

作为开发人员，您可以使用 vRealize Automation Code Stream 来生成和运行管道，以及在仪表板上监控管道活动。您具有 `User` 角色，如如何管理 vRealize Automation Code Stream 中的用户访问和批准中所述。

运行管道之后，您需要了解以下情况：

- 代码是否已成功通过管道的所有阶段。要查明这一点，请在管道执行中查看结果。
- 如果管道失败，该怎么办？导致失败的原因是什么？要查明这一点，请在管道仪表板中查看最常见的错误。

表 1-2. 使用 vRealize Automation Code Stream 的开发人员

要集成和发布代码	您要完成的工作
生成管道。	测试和部署代码。 管道失败时更新代码。
将管道连接到端点。	将管道中的任务连接到端点，例如 GitHub 存储库。
运行管道。	添加用户操作批准任务，以便其他用户可以在特定点批准您的管道。
查看仪表板。	在管道仪表板上查看结果。您可以查看趋势、历史记录、故障等。

有关入门的详细信息，请参见 VMware Code Stream 入门。

在“产品内置支持”面板中查找更多文档

如果您在此处找不到所需的信息，则可以在产品中获得更多帮助。

- 单击并阅读用户界面中的标志和工具提示，随时随地获得所需的上下文特定信息。
- 打开产品内置支持面板，并阅读针对活动用户界面页面显示的主题。您还可以在面板中搜索以获取问题的答案。

有关 Webhook 的更多信息

可以通过使用同一 Git 端点并在 Webhook 配置页面中为分支名称提供不同的值，为不同的分支创建多个 Webhook。要为同一 Git 存储库中的其他分支创建其他 Webhook，无需针对多个分支多次克隆 Git 端点。只需在 Webhook 中提供分支名称，即可重用 Git 端点。如果 Git Webhook 中的分支与端点中的分支相同，则无需在 Git Webhook 页面中提供分支名称。

设置 vRealize Automation Code Stream 以对发布流程进行建模

2

要对发布流程进行建模，请创建一个管道来表示通常用于发布软件的阶段、任务和批准。vRealize Automation Code Stream 随后会自动执行生成、测试、批准和部署代码的流程。

对软件发布流程进行建模所需的一切内容均已准备就绪，下文介绍如何在 vRealize Automation Code Stream 进行建模。

前提条件

- 确认是否有任何端点可供使用。在 vRealize Automation Code Stream 中，单击**端点**。
- 了解可以构建和部署代码的本地方法。请参见第 4 章 在 vRealize Automation Code Stream 中规划本地构建、集成和交付代码 。
- 确定将在管道中使用的部分资源是否必须标记为受限制。请参见如何管理 vRealize Automation Code Stream 中的用户访问和批准。
- 如果您具有用户角色或查看者角色而非管理员角色，请确定谁是 vRealize Automation Code Stream 实例的管理员。

步骤

1 检查 vRealize Automation Code Stream 中可用的项目，并选择适合您的项目。
   - 如果未显示任何项目，请让 vRealize Automation Code Stream 管理员创建一个项目，并使您成为该项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。
   - 如果您不是列出的任何项目的成员，请让 vRealize Automation Code Stream 管理员将您添加为项目的成员。

2 添加管道所需的任何新端点。

例如，您可能需要 Git、Jenkins、Code Stream Build、Kubernetes 和 Jira。

3 创建变量，以便可以在管道任务中重用值。

要限制管道中使用的资源（例如主机），请使用受限制变量。可以阻止管道在其他用户明确批准之前继续运行。

管理员可以创建机密变量和受限制变量。用户可以创建机密变量。

可以根据需要在多个管道中多次重用变量。例如，定义主机的变量可以为 `HostIPAddress`。要在管道任务中使用该变量，请输入 `${var.HostIPAddress}`。

4 如果您是管理员，则将对您的业务至关重要的任何端点和变量标记为受限制资源。

当非管理员用户尝试运行包含受限制资源的管道时，该管道将在使用受限制资源的任务处停止。然后，管理员必须恢复该管道。

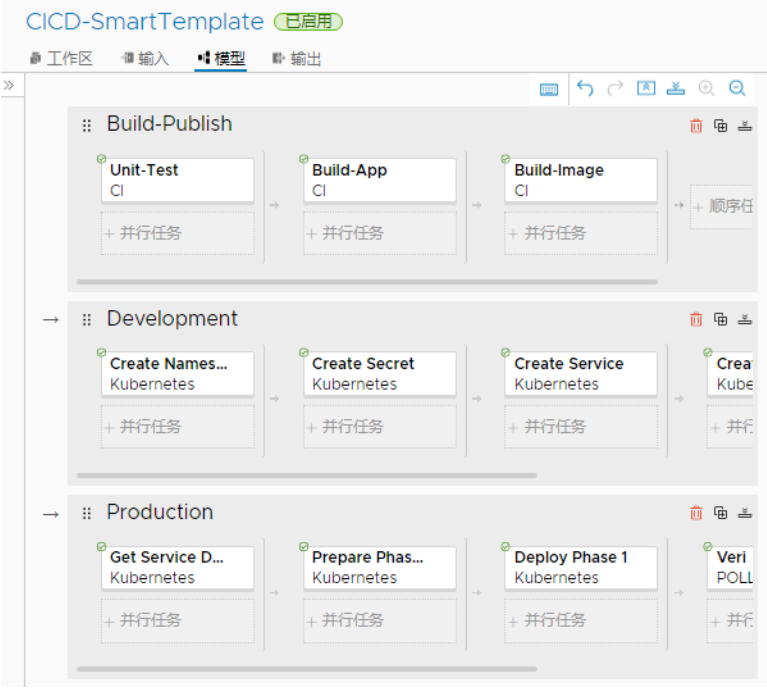
5 为本地 CICD 管道、CI 管道或 CD 管道计划构建策略。

创建持续集成 (CI) 和持续部署 (CD) 代码的管道之前，需要计划生成策略。生成计划可帮助您确定 vRealize Automation Code Stream 的需求，以便可以在本地生成、集成、测试和部署代码。

如何创建 vRealize Automation Code Stream 本地构建	此构建策略的结果
使用智能管道模板之一。	■ 为您生成所有阶段和任务。 ■ 克隆源存储库。 ■ 生成并测试代码。 ■ 将代码容器化以进行部署。 ■ 根据选择内容填充管道任务步骤。
手动添加各个阶段和任务。	您需要添加阶段、添加任务，并输入信息以加以填充。

6 通过使用智能管道模板或通过手动将阶段和任务添加到管道来创建管道。

然后，将任何资源标记为受限制。根据需要添加批准。应用任何常规变量、受限制变量或机密变量。在任务之间添加任何绑定。



7 验证、启用并运行管道。

8 查看管道执行。

9 要跟踪状态和关键绩效指标 (KPI)，请使用管道仪表板，并创建任何自定义仪表板。

结果

您已创建可在所选项目中使用的管道。

还可以导出管道 YAML，然后将其导入到其他项目并在其中重用。

后续步骤

了解您可能想要在环境中应用的用例。请参见第 5 章 vRealize Automation Code Stream 使用教程。

如何在 vRealize Automation Code Stream 中添加项目

可以创建项目，并向其中添加管理员和成员。项目成员可以使用创建管道和添加端点等功能。要为开发团队创建、删除或更新项目，您必须是 vRealize Automation Code Stream 管理员。

必须先存在项目，然后才能创建管道。创建管道时，需要选择项目以将所有管道信息分组在一起。端点和变量的定义也依赖于现有项目。

前提条件

- 确认您具有 vRealize Automation Code Stream 管理员角色。请参见 vRealize Automation Code Stream 中的角色是什么。

如果您没有 vRealize Automation Code Stream 管理员角色，但您是 vRealize Automation Cloud Assembly 中的管理员，则可以使用 vRealize Automation Cloud Assembly UI 创建、更新或删除项目。请参见如何为我的 vRealize Automation Cloud Assembly 开发团队添加项目

- 如果要将 Active Directory 组添加到项目，请确认您已为组织配置了 Active Directory 组。请参见如何在 vRealize Automation 中为项目启用 Active Directory 组。如果这些组未同步，则当您尝试将它们添加到项目时，这些组不可用。

步骤

1 选择**项目**，然后单击**新建项目**。

2 输入项目名称。

3 单击**创建**。

4 选择新创建的项目所对应的卡视图，然后单击**打开**。

5 单击**用户**选项卡，然后添加用户并分配角色。

   - 项目管理员可以添加成员。
   - 具有服务角色的项目成员可以使用服务。
   - 项目查看者可以查看项目，但不能创建、更新或删除项目。

   有关项目角色的详细信息，请参见如何管理 vRealize Automation Code Stream 中的用户访问和批准。

6 单击**保存**。

后续步骤

添加使用该项目的端点和管道。请参见第 6 章 将 vRealize Automation Code Stream 连接到端点和第 3 章 在 vRealize Automation Code Stream 中创建并使用管道。

创建管道后，将所有管道信息分组在一起的项目的名称将显示在管道卡视图和管道执行卡视图上。

如何管理 vRealize Automation Code Stream 中的用户访问和批准

vRealize Automation Code Stream 提供了多种方法用于确保用户具有使用管道以发布软件应用程序的相应授权和许可。

每个团队成员都分配有一个角色，该角色授予对管道、端点和仪表板的特定权限以及将资源标记为受限制的能力。

借助用户操作和批准，您可以控制管道何时运行，以及控制管道是否必须等待批准。您的角色决定您是否可以恢复管道，以及是否可以运行包含受限制端点或受限制变量的管道。

使用机密变量可隐藏和加密敏感信息。可以对必须隐藏和加密以及限制在执行中使用的字符串、密码和 URL 使用受限制变量。例如，对密码或 URL 使用机密变量。可以在管道的任何类型的任务中使用机密变量和受限制变量。

vRealize Automation Code Stream 中的角色是什么

根据您在 vRealize Automation Code Stream 中的角色，您可以执行特定操作以及访问特定区域。例如，您的角色可能允许您创建、更新和运行管道。或者，您可能仅具有查看管道的权限。

除受限制以外的所有操作表示此角色有权对实体执行创建、读取、更新和删除操作，但受限制的变量和端点除外。

表 2-1. vRealize Automation Code Stream 中的服务和项目级别访问权限

	vRealize Automation Code Stream 角色				
访问级别	Code Stream 管理员	Code Stream 开发人员	Code Stream 执行者	Code Stream 查看者	Code Stream 用户
vRealize Automation Code Stream 服务级别的访问	全部操作	除受限制以外的所有操作	执行操作	只读	无
项目级别的访问：项目管理员	全部操作	全部操作	全部操作	全部操作	全部操作
项目级别的访问：项目成员	全部操作	除受限制以外的所有操作	除受限制以外的所有操作	除受限制以外的所有操作	除受限制以外的所有操作
项目级别的访问：项目查看者	全部操作	除受限制以外的所有操作	执行操作	只读	只读

具有项目管理员角色的用户可以针对其作为项目管理员的项目执行所有操作。

项目管理员可以创建、读取、更新和删除管道、变量、端点、仪表板、触发器，以及启动包括受限制端点或变量的管道，前提是这些资源位于用户作为项目管理员的项目中。

具有“服务查看者”角色的用户可以查看管理员可使用的所有信息。除非管理员将他们设置为项目管理员或项目成员，否则他们无法执行任何操作。如果用户与项目关联，则他们具有与该角色相关的权限。项目查看者不会像管理员或成员角色那样扩展其权限。此角色在所有项目中均为只读。

如果您在项目中具有读取权限，则仍可查看受限制的资源。

- 要查看受限制的端点（在端点卡视图上显示锁定图标），请单击**配置 > 端点**。
- 要查看受限制的变量和机密变量（在**类型**列中显示“受限制”或“机密”），请单击**配置 > 变量**。

表 2-2. vRealize Automation Code Stream 服务角色功能

UI 上下文	功能	Code Stream 管理员角色	Code Stream 开发人员角色	Code Stream 执行者角色	Code Stream 查看者角色	Code Stream 用户角色
管道						
	查看管道	是	是	是	是	
	创建管道	是	是			
	运行管道	是	是	是		

表 2-2. vRealize Automation Code Stream 服务角色功能 （续）

UI 上下文	功能	Code Stream 管理员角色	Code Stream 开发人员角色	Code Stream 执行者角色	Code Stream 查看者角色	Code Stream 用户角色
	运行包含受限制端点或受限制变量的管道	是				
	更新管道	是	是			
	删除管道	是	是			
管道执行						
	查看管道执行	是	是	是	是	
	恢复、暂停和取消管道执行	是	是	是		
	恢复等待批准受限制资源的管道	是				
自定义集成						
	创建自定义集成	是	是			
	读取自定义集成	是	是	是	是	
	更新自定义集成	是	是			
端点						
	查看执行	是	是	是	是	
	创建执行	是	是			
	更新执行	是	是			
	删除执行	是	是			
将资源标记为受限制						
	将端点或变量标记为受限制	是				
仪表板						
	查看仪表板	是	是	是	是	
	创建仪表板	是	是			
	更新仪表板	是	是			
	删除仪表板	是	是			

vRealize Automation Code Stream 中的自定义角色和权限

可以在 vRealize Automation Cloud Assembly 中创建自定义角色，以将特权扩展到使用管道的用户。为 vRealize Automation Code Stream 管道创建自定义角色时，可以选择一个或多个**管道**权限。

选择将为其分配此自定义角色的用户所需的最小**管道**权限数量。

将用户分配给项目并在该项目中赋予角色时，如果为该用户分配包含一个或多个**管道**权限的自定义角色，则他们可以执行权限所允许的所有操作。例如，他们可以创建受限制的变量、管理受限制的管道、创建和管理自定义集成等。

表 2-3. 可分配给自定义角色的管道权限

管道权限	Code Stream 管理员	Code Stream 开发人员	Code Stream 执行者	Code Stream 查看者	Code Stream 用户	项目管理员	项目成员	项目查看者
管理管道	是	是				是	是	
管理受限制的管道	是					是		
管理自定义集成	是	是						
执行管道	是	是	是			是	是	
执行受限制的管道	是					是		
管理执行	是					是		
读取。此权限不可见。	是	是	是	是		是	是	是

表 2-4. 如何在自定义角色中使用管道权限

权限	可执行的操作
管理管道	■ 创建、更新、删除、克隆管道。 ■ 将管道发布和取消发布到 VMware Service Broker。 ■ 创建、更新和删除端点。 ■ 创建、更新和删除常规变量和机密变量。 ■ 创建、克隆、更新和删除 Gerrit 侦听器。 ■ 连接和断开连接 Gerrit 侦听器。 ■ 创建、克隆、更新、删除 Gerrit 触发器。 ■ 创建、更新和删除 Git webhook。 ■ 创建、更新和删除 Docker webhook。 ■ 使用智能管道模板创建管道。 ■ 从 YAML 导入管道和将其导出到 YAML。 ■ 创建、更新和删除自定义仪表板。 ■ 读取所有自定义集成。 ■ 读取所有受限制的端点和变量，但无法查看其值。
管理受限制的管道	■ 创建、更新和删除端点。 ■ 将端点标记为受限制，更新受限制的端点和删除这些端点。 ■ 创建、更新和删除常规变量和机密变量。 ■ 创建、更新和删除受限制的变量。 ■ 可以使用“管理管道”执行的所有权限。
管理自定义集成	■ 创建和更新自定义集成。 ■ 版本控制和发布自定义集成。 ■ 删除和弃用自定义集成版本。 ■ 删除自定义集成。
执行管道	■ 运行管道。 ■ 暂停、恢复和取消管道执行。 ■ 重新运行管道执行。 ■ 恢复、重新运行和手动触发 Gerrit 触发器事件。 ■ 批准用户操作，并可对用户操作执行批量批准。
执行受限制的管道	■ 运行管道。 ■ 暂停、恢复、取消和删除管道执行。 ■ 重新运行管道执行。 ■ 同步正在运行的管道执行。 ■ 强制删除正在运行的管道执行。 ■ 恢复、重新运行、删除和手动触发 Gerrit 触发器事件。 ■ 解决受限制的项目并继续管道执行。 ■ 切换用户上下文并在用户操作任务批准后继续管道执行。 ■ 可以使用“执行管道”执行的所有权限。
管理执行	■ 运行管道。 ■ 暂停、恢复、取消和删除管道执行。 ■ 重新运行管道执行。 ■ 恢复、重新运行、删除和手动触发 Gerrit 触发器事件。 ■ 可以使用“执行管道”执行的所有权限。

自定义角色可包括权限组合。这些权限组织为允许用户管理或运行管道的功能组（受限制和不受限制）。这些权限表示每个角色可在 vRealize Automation Code Stream 中执行的所有功能。

例如，如果创建自定义角色并包括称为**管理受限制的管道**的权限，则具有 vRealize Automation Code Stream 开发人员角色的用户可以执行以下操作：

- 创建、更新和删除端点。
- 将端点标记为受限制，更新受限制的端点和删除这些端点。
- 创建、更新和删除常规变量和机密变量。
- 创建、更新和删除受限制的变量。

表 2-5. 自定义角色中的管道权限组合示例

分配给自定义角色的权限数	组合权限示例	如何使用此组合
单个权限	**执行管道**	
两个权限	**管理管道**和**执行管道**	
三个权限	**管理管道**和**执行管道**和**执行受限制的管道**	
	管理管道和**管理自定义集成**和**执行受限制的管道**	此组合可应用于 vRealize Automation Code Stream 开发人员角色，但仅限于用户作为其成员的项目。
	管理管道和**管理自定义集成**和**管理执行**	此组合可应用于 vRealize Automation Code Stream 管理员，但仅限于用户属于其成员的项目。
	管理管道、**管理受限制的管道**和**管理自定义集成**	具有此组合后，用户将拥有完全权限，可以在 vRealize Automation Code Stream 中创建和删除任何内容。

如果您具有管理员角色

作为管理员，您可以创建自定义集成、端点、变量、触发器、管道和仪表板。

通过项目，管道可以访问基础架构资源。管理员创建项目，以便用户可以将管道、端点和仪表板组合在一起。然后，用户在其管道中选择项目。每个项目都包括一位管理员和分配有角色的用户。

如果您具有管理员角色，则可以将端点和变量标记为受限制资源，还可以运行使用受限制资源的管道。如果非管理用户运行包含受限制端点或变量的管道，则管道将在使用受限制变量的任务处停止，并且管理员必须恢复管道。

作为管理员，您还可以请求在 vRealize Automation Service Broker 中发布管道。

如果您具有开发人员角色

除了无法使用受限制端点或受限制变量，您可以像管理员一样使用管道。

如果运行使用受限端点或变量的管道，则该管道仅运行使用受限资源的任务。然后，管道停止， vRealize Automation Code Stream 管理员或项目管理员必须恢复管道。

如果您具有用户角色

您可以访问 vRealize Automation Code Stream，但不具有其他角色提供的任何特权。

如果您具有查看者角色

您可以查看与管理员所查看的相同资源，例如管道、端点、管道执行、仪表板、自定义集成和触发器，但无法执行创建、更新或删除操作。要执行操作，还必须为查看者角色赋予项目管理员或项目成员角色。

具有查看者角色的用户可以查看项目。此外，他们还可以查看受限制的端点和受限制的变量，但无法查看其相关详细信息。

如果您具有执行者角色

您可以运行管道并对用户操作任务执行操作。您还可以恢复、暂停和取消管道执行。但是，您无法修改管道。

如何分配和更新角色

您必须是管理员，才能为其他用户分配和更新角色。

1 要查看活动用户及其角色，请在 vRealize Automation 中单击右上角的九个点。

2 单击**身份与访问管理**。

3 要显示用户名和角色，请单击**活动用户**。

4 要为用户添加角色或更改其角色，请单击用户名旁边的复选框，然后单击**编辑角色**。

5 添加或更改用户角色时，您还可以添加对服务的访问权限。

6 要保存更改，请单击**保存**。

vRealize Automation Code Stream 中的用户操作和批准是什么

“用户操作”区域显示需要批准的管道运行。要求的审批者可以批准或拒绝管道运行。

创建管道时，在下列情况下可能需要向管道添加批准：

- 团队成员需要审阅您的代码。
- 其他用户需要确认生成工件。
- 您必须确保所有测试均已完成。
- 任务使用管理员标记为受限制的资源，并且该任务需要批准。
- 管道将软件发布到生产环境。

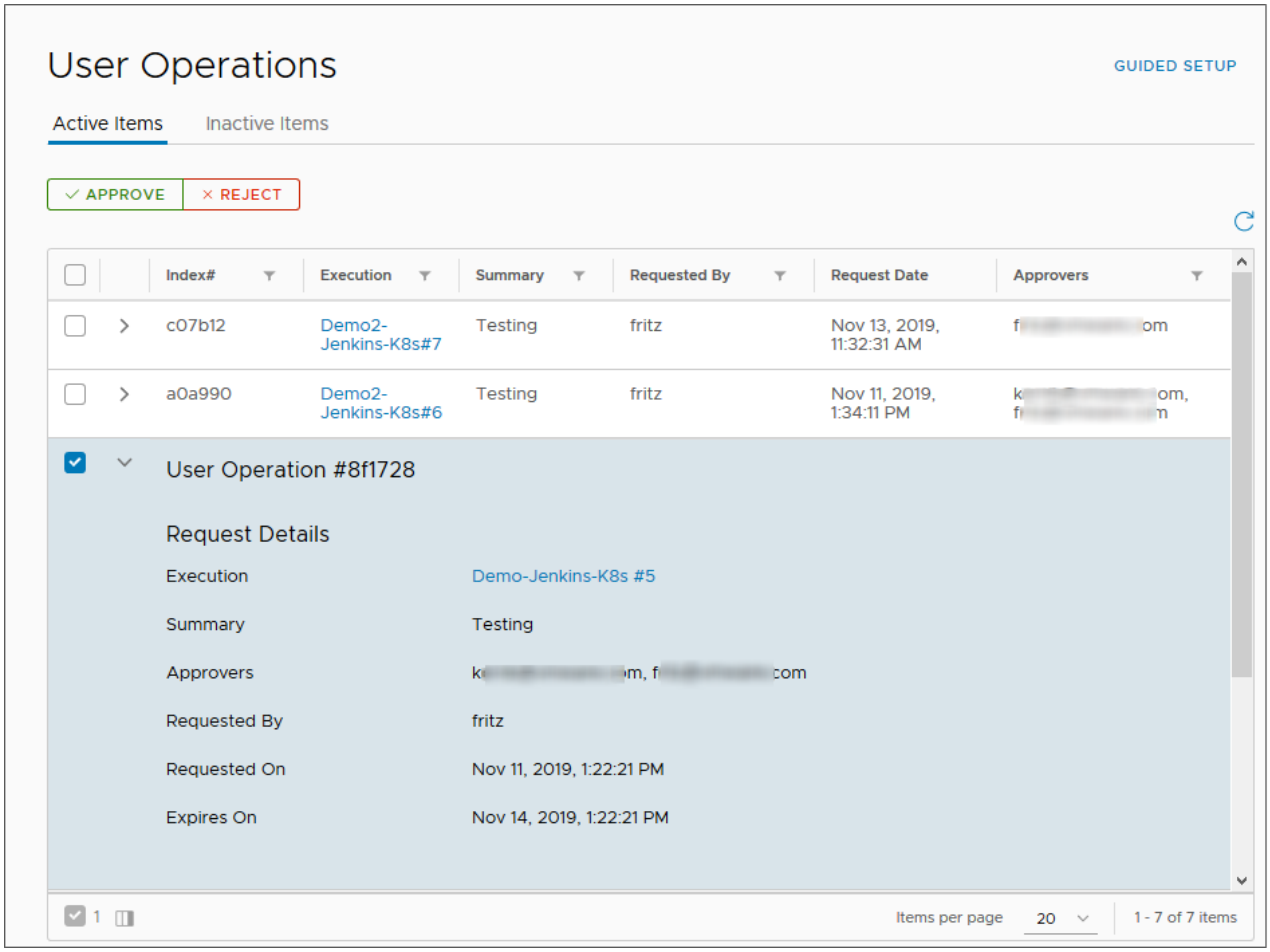
要确定是否批准管道任务，要求的审批者必须具有权限和专业知识。

添加用户操作任务时，可以设置过期超时（以天、小时或分钟为单位）。例如，您可能需要要求的用户在 30 分钟内批准管道。如果未在 30 分钟内批准管道，则管道将会按预期失败。

如果启用发送电子邮件通知，则“用户操作”任务只向具有完整电子邮件地址的审批者发送通知，而不会向非电子邮件格式的审批者名称发送通知。

在要求的用户批准任务后：

- 可以继续执行挂起的管道。
- 当管道继续运行时，将取消之前因为批准该相同用户操作任务而挂起的任何请求。



在“用户操作”区域中，要批准或拒绝的项显示为活动或非活动项。每一项都映射到管道中的用户操作任务。

- **活动项目**等待审批者必须审阅任务，然后批准或拒绝。如果您是审批者列表中的用户，则可以展开用户操作行，然后单击**接受**或**拒绝**。
- **非活动项**已被批准或拒绝。如果某个用户已拒绝该用户操作，或者该任务的批准已超时，则无法再对其进行批准。

索引编号是一个唯一的六个字符的字母数字字符串，可以将其用作筛选器搜索特定批准。

管道批准还会显示在**执行**区域中。

- 正在等待批准的管道显示为处于等待状态。
- 其他状态包括“已排队”、“已完成”和“已失败”。
- 如果管道处于等待状态，则要求的审批者必须批准您的管道任务。

在 vRealize Automation Code Stream 中创建并使用管道

3

可以使用 vRealize Automation Code Stream 对生成、测试和部署流程进行建模。使用 vRealize Automation Code Stream，可以设置支持发布周期的基础架构，并创建用于对软件发布活动进行建模的管道。vRealize Automation Code Stream 的软件交付流程涵盖了开发代码、测试以及将软件部署到生产实例。

每个管道都包含一些阶段和任务。阶段表示开发阶段，而任务执行在各个阶段交付软件应用程序所需的操作。

vRealize Automation Code Stream 中的管道是什么

管道是软件发布流程的持续集成和持续交付模型。其软件发布流程涵盖了源代码、测试到生产。它包含一系列阶段，而这些阶段包含表示软件发布周期中的活动的任务。软件应用程序在管道中从一个阶段流动到下一个阶段。

您可以添加端点，以便管道中的任务可以连接到数据源、存储库或通知系统。

创建管道

您可以从空白画布开始，使用智能管道模板或通过导入 YAML 代码来创建管道。

- 使用空白画布。有关示例，请参见在手动添加任务之前在 vRealize Automation Code Stream 中规划 CICD 本地构建。
- 使用智能管道模板。有关示例，请参见第 4 章 在 vRealize Automation Code Stream 中规划本地构建、集成和交付代码 。
- 导入 YAML 代码。单击**管道 > 导入**。在**导入**对话框中，选择 YAML 文件或输入 YAML 代码，然后单击**导入**。

当使用空白画布创建管道时，可以添加阶段、任务和批准。管道将自动执行生成、测试、部署和发布应用程序的流程。各个阶段中的任务将运行相应的操作，以在各个阶段生成、测试和发布代码。

表 3-1. 管道阶段和用途示例

阶段示例	可执行的操作示例
开发	在开发阶段，您可以置备计算机，检索工件，添加构建任务以创建 Docker 主机以便用于持续集成代码以及执行其他操作。 例如： ■ 要规划和创建持续集成 (CI) 生成，以通过使用 vRealize Automation Code Stream 中的本机生成功能交付代码，请参见在使用智能管道模板之前在 vRealize Automation Code Stream 中计划持续集成本机构建。
测试	在测试阶段，您可以添加 Jenkins 任务以测试软件应用程序，包含诸如 JUnit 和 JaCoCo 等后处理测试工具以及执行其他操作。 例如： ■ 将 vRealize Automation Code Stream 与 Jenkins 集成，并在管道中运行 Jenkins 作业，以生成和测试源代码。请参见如何将 vRealize Automation Code Stream 与 Jenkins 集成。 ■ 创建自定义脚本，以扩展 vRealize Automation Code Stream 的功能，从而与您自己的生成、测试和部署工具集成。请参见如何将自己的生成工具、测试工具和部署工具与 vRealize Automation Code Stream 集成。 ■ 跟踪持续集成 (CI) 管道的后处理趋势。请参见如何在 vRealize Automation Code Stream 中使用自定义仪表板跟踪管道的关键绩效指标。
生产	在生产阶段，可以与 vRealize Automation Cloud Assembly 中的云模板集成以置备基础架构，将软件部署到 Kubernetes 集群以及执行其他操作。 例如： ■ 要查看开发和生产阶段示例（这些阶段可以在您自己的蓝绿部署模型中部署软件应用程序），请参见如何将 vRealize Automation Code Stream 中的应用程序部署到蓝绿部署。 ■ 要将云模板集成到管道，请参见如何自动执行从 vRealize Automation Code Stream 中的 YAML 云模板部署的应用程序的发布。您还可以添加部署任务，以运行脚本来部署应用程序。 ■ 要将软件应用程序自动部署到 Kubernetes 集群，请参见如何自动执行 vRealize Automation Code Stream 中应用程序到 Kubernetes 集群的发布。 ■ 要将代码集成到管道并部署生成映像，请参见如何持续将来自 GitHub 或 GitLab 存储库的代码集成到 vRealize Automation Code Stream 中的管道。

您可以将管道导出为 YAML 文件。单击**管道**，单击管道卡视图，然后单击**操作 > 导出**。

批准管道

您可以在管道中的特定点获得其他团队成员的批准。

- 要通过在管道中包括用户操作任务来要求对管道进行批准，请参见如何运行管道和查看结果。此任务会向必须审阅该任务的用户发送电子邮件通知。审阅者必须批准或拒绝，然后管道才能继续运行。如果用户操作任务设置了过期超时（以天、小时或分钟为单位），则要求的用户必须在任务过期之前批准管道。否则，管道将会按预期失败。
- 在管道的任何阶段，如果任务或阶段失败，都可以让 vRealize Automation Code Stream 创建 Jira 票证。请参见管道任务失败时如何在 vRealize Automation Code Stream 中创建 Jira 票证。

触发管道

当开发人员将代码签入存储库、审阅代码或标识新的或已更新的构建工件时，会触发管道。

- 要将 vRealize Automation Code Stream 与 Git 生命周期集成，并在开发人员更新其代码时触发管道，请使用 Git 触发器。请参见如何使用 vRealize Automation Code Stream 中的 Git 触发器运行管道。
- 要将 vRealize Automation Code Stream 与 Gerrit 代码审阅生命周期集成，并在代码审阅时触发管道，请使用 Gerrit 触发器。请参见如何使用 vRealize Automation Code Stream 中的 Gerrit 触发器运行管道。
- 要在创建或更新 Docker 生成工件时触发管道，请使用 Docker 触发器。请参见如何使用 vRealize Automation Code Stream 中的 Docker 触发器运行持续交付管道。

有关 vRealize Automation Code Stream 支持的触发器的详细信息，请参见第 7 章 在 vRealize Automation Code Stream 中触发管道。

本章讨论了以下主题：

- 如何运行管道和查看结果
- vRealize Automation Code Stream 中提供哪些类型的任务
- 如何在 vRealize Automation Code Stream 管道中使用变量绑定
- 如何在条件任务中使用变量绑定来运行或停止 vRealize Automation Code Stream 中的管道
- 在 vRealize Automation Code Stream 中绑定管道任务时可以使用哪些变量和表达式
- 如何发送有关 vRealize Automation Code Stream 中的管道的通知
- 管道任务失败时如何在 vRealize Automation Code Stream 中创建 Jira 票证
- 如何在 vRealize Automation Code Stream 中回滚部署

如何运行管道和查看结果

可以从管道卡视图、在管道编辑模式下以及从管道执行运行管道。也可以使用可用触发器在发生特定事件时使 vRealize Automation Code Stream 运行管道。

当管道中的所有阶段和任务都有效时，就可以发布、运行或触发该管道。

如果要使用 vRealize Automation Code Stream 运行或触发管道，可以从管道卡视图启用并运行管道，也可以在位于管道中时启用和运行管道。然后，可以查看管道执行情况，以确认管道已构建、测试并部署您的代码。

如果您是管理员或非管理员用户，可在管道执行正在进行的过程中删除该执行。

- 管理员：要删除正在运行的管道执行，请单击**执行**。在要删除的执行上，单击**操作 > 删除**。
- 非管理员用户：要删除正在运行的管道执行，请单击**执行**，然后单击 **Alt Shift d** 。

当管道执行正在进行中且似乎已停滞时，管理员可以从“执行”页面或“执行详细信息”页面刷新执行。

- “执行”页面：单击**执行**。在要刷新的执行上，单击**操作 > 同步**。

- “执行详细信息”页面：单击**执行**，再单击执行详细信息的链接，然后单击**操作 > 同步**。

要在发生特定事件时运行管道，请使用触发器。

- Git 触发器可在开发人员更新代码时运行管道。
- Gerrit 触发器可在进行代码审阅时运行管道。
- Docker 触发器可在 Docker 注册表中创建工件时运行管道。
- `curl` 命令或 `wget` 命令可以指示 Jenkins 在 Jenkins 构建完成后运行管道。

有关使用触发器的更多信息，请参见第 7 章 在 vRealize Automation Code Stream 中触发管道。

以下过程介绍了如何从管道卡视图运行管道、查看执行、查看执行详细信息以及使用操作。此外，还介绍了如何释放管道，以便将其添加到 vRealize Automation Service Broker。

前提条件

- 验证是否已创建一个或多个管道。请参见第 5 章 vRealize Automation Code Stream 使用教程中的示例。

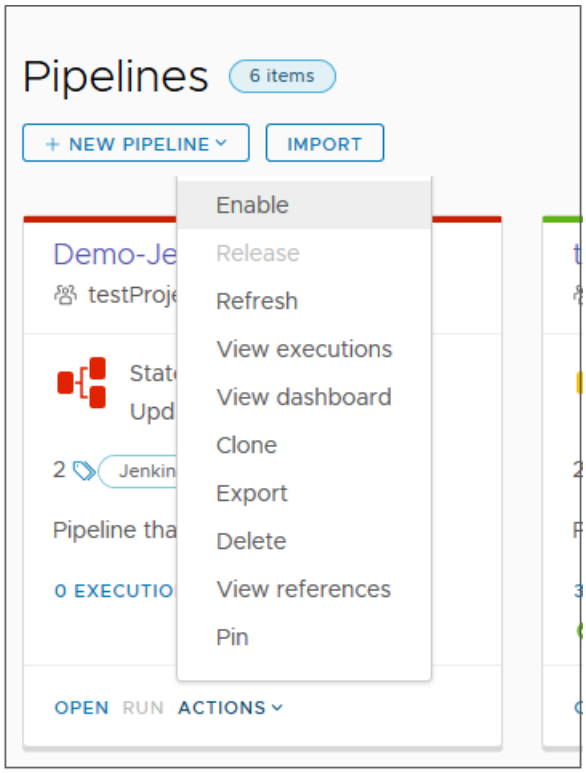
步骤

1 启用管道。

要运行或发布管道，必须先将其启用。

a 单击**管道**。

b 在管道卡中，单击**操作 > 启用**。



您还可以在管道中启用管道。如果您的管道已启用，则**运行**处于活动状态，**操作**菜单将显示**禁用**。

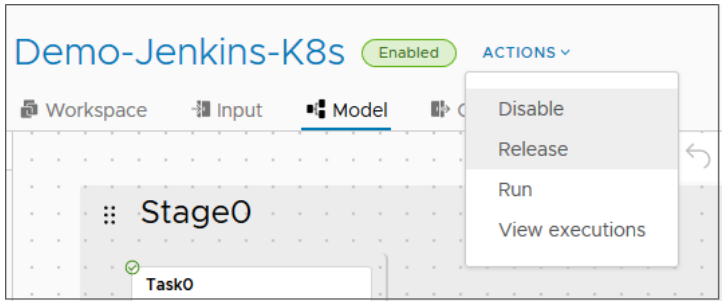
2 （可选） 发布管道。

如果要使管道可用作 vRealize Automation Service Broker 中的目录项，则必须在 vRealize Automation Code Stream 中发布它。

a 单击**管道**。

b 在管道卡中，单击**操作 > 发布**。

还可以在位于管道中时发布管道。



发布管道后，打开 vRealize Automation Service Broker 将管道添加为目录项并运行它。请参见将 vRealize Automation Code Stream 管道添加到 vRealize Automation Service Broker 目录。

注 如果管道需要 120 分钟以上的时间才能运行，请提供近似的执行时间作为请求超时值。要设置或查看项目的请求超时值，请以管理员身份打开 vRealize Automation Service Broker，然后选择**基础架构 > 项目**。单击项目名称，然后单击**置备**。

如果未设置请求超时值，则需要 120 分钟以上的时间才能运行的执行将显示为失败，并出现回调超时请求错误。但是，管道执行不受影响。

---

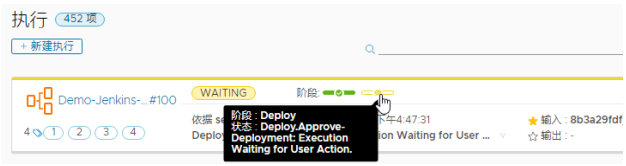
3 在管道卡中，单击**运行**。

4 要在运行过程中查看管道，请单击**执行**。

管道按顺序运行每个阶段，管道执行将针对每个阶段显示一个状态图标。如果管道包括用户操作任务，则用户必须批准该任务才能继续运行管道。使用用户操作任务时，管道会停止运行并等待所需用户批准该任务。

例如，可以使用用户操作任务批准将代码部署到生产环境。

如果用户操作任务设置了过期超时（以天、小时或分钟为单位），则要求的用户必须在任务过期之前批准管道。否则，管道将会按预期失败。



5 要查看正在等待用户批准的管道阶段，请单击该阶段的状态图标。

6 要查看任务的详细信息，请单击该任务。

所需的用户批准该任务之后，具有适当角色的用户必须恢复管道。有关所需角色，请参见如何管理 vRealize Automation Code Stream 中的用户访问和批准。

如果执行失败，则必须对失败的原因进行分类和修复。然后，转到执行，单击**操作 > 重新运行**。

您可以恢复主管道执行和嵌套执行。

7 在管道执行中，可以单击**操作**以查看管道，然后选择**暂停**、**取消**等操作。如果您是管理员，则可以在管道执行期间删除或同步管道执行。如果您是非管理员用户，可以删除正在运行的管道。

8 要轻松地在执行之间导航并查看任务的详细信息，请单击**执行**，并单击管道运行。然后单击顶部的选项卡，选择管道运行。

结果

恭喜！您运行了管道，检查了管道执行，并查看了需要批准才能继续运行管道的用户操作任务。您还使用管道执行中的**操作**菜单返回到管道模型，以便可以进行任何必需的更改。

后续步骤

要了解有关使用 vRealize Automation Code Stream 自动执行软件发布周期的更多信息，请参见第 5 章 vRealize Automation Code Stream 使用教程。

vRealize Automation Code Stream 中提供哪些类型的任务

配置管道时，可以添加特定类型的任务，以便管道针对所需操作运行这些任务。每种任务类型都与另一个应用程序集成，并在构建，测试和交付应用程序时启用管道。

要运行管道，无论是必须从存储库中提取工件以进行部署，运行远程脚本还是要求组成员批准用户操作，vRealize Automation Code Stream 都具有相应类型的任务！

在管道中使用任务之前，请确认相应的端点可用。

表 3-2. 获取批准或设置决策点

任务类型	作用	示例和详细信息
用户操作	“用户操作”任务启用所需的批准，以控制何时运行管道以及何时必须等待批准。	请参见如何运行管道和查看结果。和如何管理 vRealize Automation Code Stream 中的用户访问和批准。
条件	添加决策点，根据条件表达式确定继续运行或停止管道。条件为 true 时，管道将运行连续任务。条件为 false 时，管道将停止。	请参见如何在条件任务中使用变量绑定来运行或停止 vRealize Automation Code Stream 中的管道。

表 3-3. 自动执行持续集成和部署

任务类型	作用	示例和详细信息
云模板	从 GitHub 部署自动化云模板，置备应用程序，并为您的部署自动执行该云模板的持续集成和持续交付 (CICD)。	请参见如何自动执行从 vRealize Automation Code Stream 中的 YAML 云模板部署的应用程序的发布。 首次选择**创建**或**更新**，然后选择**云模板**和**版本**后，将显示云模板参数。可以将以下元素（可使用变量绑定）添加到云模板任务中的输入文本区域： ■ 整数 ■ 枚举字符串 ■ 布尔 ■ 数组变量 在输入中使用变量绑定时，请注意以下例外。对于枚举，必须从固定集中选择枚举值。对于布尔值，必须在输入文本区域中输入值。 当 vRealize Automation Cloud Assembly 中的云模板包含输入变量时，云模板参数将显示在云模板任务中。例如，如果云模板的输入类型为 `Integer`，则可以直接输入整数，也可以使用变量绑定作为变量输入。
CI	“CI”任务通过从注册表端点中提取 Docker 构建映像并将其部署到 Kubernetes 集群，将代码持续集成到管道中。	请参见在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建。
自定义	“自定义”任务将 vRealize Automation Code Stream 与您自己的构建工具、测试工具和部署工具集成。	请参见如何将自己的生成工具、测试工具和部署工具与 vRealize Automation Code Stream 集成。
Kubernetes	自动将软件应用程序部署到 AWS 上的 Kubernetes 集群。	请参见如何自动执行 vRealize Automation Code Stream 中应用程序到 Kubernetes 集群的发布。
管道	将管道嵌套在主管道中。嵌套管道时，该管道将作为主管道中的任务。 在主管道的“任务”选项卡上，可以通过单击嵌套管道的链接轻松导航到该管道。将在新的浏览器选项卡中打开嵌套管道。	要在**执行**中查找嵌套管道，请在搜索区域中输入 **`nested`**。

表 3-4. 集成开发、测试和部署应用程序

任务类型...	作用...	示例和详细信息...
Bamboo	与 Bamboo 持续集成 (CI) 服务器交互，该服务器持续生成、测试和集成软件，为部署做好准备，并在开发人员提交更改时触发代码生成。它会公开 Bamboo 生成的工件位置，以便任务可以输出用于生成和部署的其他任务的参数。	连接到 Bamboo 服务器端点并从管道启动 Bamboo 生成计划。
Jenkins	触发生成和测试源代码的 Jenkins 作业，运行测试用例，并且可以使用自定义脚本。	请参见如何将 vRealize Automation Code Stream 与 Jenkins 集成。

表 3-4. 集成开发、测试和部署应用程序 （续）

任务类型...	作用...	示例和详细信息...
TFS	支持将管道连接到 Team Foundation Server，以便管理和调用生成项目，包括可生成和测试代码的已配置作业。	vRealize Automation Code Stream 支持 Team Foundation Server 2013 和 2015。
vRO	通过在 vRealize Orchestrator 中运行预定义工作流或自定义工作流来扩展 vRealize Automation Code Stream 的功能。	请参见如何将 vRealize Automation Code Stream 与 vRealize Orchestrator 集成。

表 3-5. 通过 API 集成其他应用程序

任务类型...	作用...	示例和详细信息...
REST	将 vRealize Automation Code Stream 与其他使用 REST API 的应用程序集成，以便您能够持续开发和交付相互交互的软件应用程序。	请参见如何使用 REST API 将 vRealize Automation Code Stream 与其他应用程序集成。
Poll	调用 REST API，并对其进行轮询，直到管道任务满足退出条件并完成为止。 vRealize Automation Code Stream 管理员最多可以将轮询计数设置为 10000。轮询时间间隔必须大于或等于 60 秒。 选中**出现故障时继续**复选框时，如果计数或时间间隔超过这些值，轮询任务将继续运行。	请参见如何使用 REST API 将 vRealize Automation Code Stream 与其他应用程序集成。

表 3-6. 运行远程脚本和用户定义的脚本

任务类型	作用	示例和详细信息
PowerShell	通过 PowerShell 任务， vRealize Automation Code Stream 可以在远程主机上运行脚本命令。例如，脚本可以自动执行测试任务，并运行管理类型的命令。 脚本可以是远程脚本，也可以是用户定义的脚本。可以通过 HTTP 或 HTTPS 进行连接，并且可以使用 TLS。 Windows 主机必须配置 `winrm` 服务， `winrm` 必须配置 `MaxShellsPerUser` 和 `MaxMemoryPerShellMB`。 要运行 PowerShell 任务，您必须具有与远程 Windows 主机的活动会话。 **PowerShell 命令行长度** 如果输入 base64 PowerShell 命令，请注意必须计算整体命令长度。 vRealize Automation Code Stream 管道对 base64 PowerShell 命令进行编码并包装到另一个命令中，这会增加命令的总长度。 PowerShell `winrm` 命令允许的最大长度为 8192 字节。对 PowerShell 任务进行编码和包装时，命令长度限制较低。因此，必须在输入 PowerShell 命令之前先计算命令长度。 vRealize Automation Code Stream PowerShell 任务的命令长度限制取决于原始命令的 base64 编码长度。命令长度的计算方法如下。 `3 * (length of original command / 4)) - (numberOfPaddingCharacters) + 77 (Length of Write-output command)` vRealize Automation Code Stream 的命令长度必须小于最大限制 (8192)。	配置 `MaxShellsPerUser` 和 `MaxMemoryPerShellMB` 时： ■ `MaxShellsPerUser` 的可接受值为 `500`（50 个并发管道），每个管道 5 个 PowerShell 任务。要设置值，请运行：`winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'` ■ `MaxMemoryPerShellMB` 的可接受内存值为 `2048`。要设置值，请运行：`winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'` 脚本会将输出写入到其他管道可以使用的响应文件中。
SSH	通过 SSH 任务，Bash shell 脚本任务可以在远程主机上运行脚本命令。例如，脚本可以自动执行测试任务，并运行管理类型的命令。 脚本可以是远程脚本，也可以是用户定义的脚本。可以通过 HTTP 或 HTTPS 进行连接，并且需要私钥或密码。 必须在 Linux 主机上配置 SSH 服务，并且 `MaxSessions` 的 SSHD 配置必须设置为 `50`。	脚本可以是远程脚本，也可以是用户定义的脚本。例如，脚本可能会如下所示： `message="Hello World" echo $message` 脚本会将输出写入到其他管道可以使用的响应文件中。

如何在 vRealize Automation Code Stream 管道中使用变量绑定

绑定管道任务意味着将在管道运行时为任务创建依赖关系。您可以通过多种方式为管道任务创建绑定。您可以将任务绑定到其他任务，绑定到变量和表达式，或绑定到条件。

如何在云模板任务中将美元绑定应用于云模板变量

可以在 vRealize Automation Code Stream 管道云模板任务中将美元绑定应用于云模板变量。在 vRealize Automation Code Stream 中修改变量的方式取决于变量属性在云模板中的编码。

如果必须在云模板任务中使用美元绑定，但在云模板任务中使用的当前云模板版本不允许使用美元绑定，请在 vRealize Automation Cloud Assembly 中修改云模板并部署新版本。然后，在云模板任务中使用新的云模板版本，并根据需要添加美元绑定。

要对 vRealize Automation Cloud Assembly 云模板提供的属性类型应用美元绑定，您必须具有正确的权限。

- 您必须与在 vRealize Automation Cloud Assembly 中创建云模板部署的人员具有相同的角色。
- 对管道进行建模的人员和运行管道的人员可能是两个不同的用户，并且可能具有不同的角色。
- 如果开发人员具有 vRealize Automation Code Stream 执行者角色并对管道进行建模，则该开发人员还必须具有部署云模板的人员的相同 vRealize Automation Cloud Assembly 角色。例如，所需角色可能是 vRealize Automation Cloud Assembly 管理员。
- 只有对管道进行建模的人员才可以创建管道并创建部署，因为他们具有权限。

要在云模板任务中使用 API 令牌，请执行以下操作：

- 对管道进行建模的人员可以为具有 vRealize Automation Code Stream 执行者角色的其他用户提供 API 令牌。之后，当执行者运行管道时，将使用 API 令牌和 API 令牌创建的凭据。
- 当用户在云模板任务中输入 API 令牌时，会创建管道所需的凭据。
- 要对 API 令牌值进行加密，请单击**创建变量**。
- 如果没有为 API 令牌创建变量，并在云模板任务中使用该令牌，则 API 令牌值将以纯文本形式显示。

要在云模板任务中将美元绑定应用于云模板变量，请执行以下步骤。

首先，在云模板中定义输入变量属性，例如 `integerVar`、`stringVar`、`flavorVar`、`BooleanVar`、`objectVar` 和 `arrayVar`。可以在 `resources` 部分中找到定义的映像属性。云模板代码中的属性可能如下所示：

```
formatVersion: 1
inputs:
  integerVar:
    type: integer
    encrypted: false
    default: 1
  stringVar:
    type: string
    encrypted: false
    default: bkix
  flavorVar:
    type: string
    encrypted: false
    default: medium
  BooleanVar:
    type: boolean
    encrypted: false
    default: true
  objectVar:
    type: object
    encrypted: false
```

```
    default:
      bkix2: bkix2
  arrayVar:
    type: array
    encrypted: false
    default:
      - '1'
      - '2'
resources:
  Cloud_Machine_1:
    type: Cloud.Machine
    properties:
      image: ubuntu
      flavor: micro
      count: '${input.integerVar}'
```

可以对 `image` 和 `flavor` 使用美元符号变量 (**$**)。例如：

```
resources:
  Cloud_Machine_1:
    type: Cloud.Machine
    properties:
      input: '${input.image}'
      flavor: '${input.flavor}'
```

要在 vRealize Automation Code Stream 管道中使用云模板并将美元绑定添加到其中，请执行以下步骤。

1 在 vRealize Automation Code Stream 中，单击**管道 > 空白画布**。

2 在管道中添加**云模板**任务。

3 在云模板任务中，对于**云模板源**，选择 **Cloud Assembly 云模板**，输入云模板名称，然后选择云模板版本。

4 请注意，可以输入为管道提供凭据的 API 令牌。要在云模板任务中创建加密 API 令牌的变量，请单击**创建变量**。

5 在显示的**参数和值**表中，请注意参数值。`flavor` 的默认值为 `small`，`image` 的默认值为 `ubuntu`。

6 假设您必须在 vRealize Automation Cloud Assembly 中更改云模板。例如：

   a 将 `flavor` 设置为使用 `array` 类型的属性。当类型为 **`array`** 时，vRealize Automation Cloud Assembly 允许对 `Flavor` 使用逗号分隔值。

   b 单击**部署**。

   c 在“部署类型”页面上，输入部署名称，然后选择云模板的版本。

   d 在“部署输入”页面上，可以为 `Flavor` 定义一个或多个值。

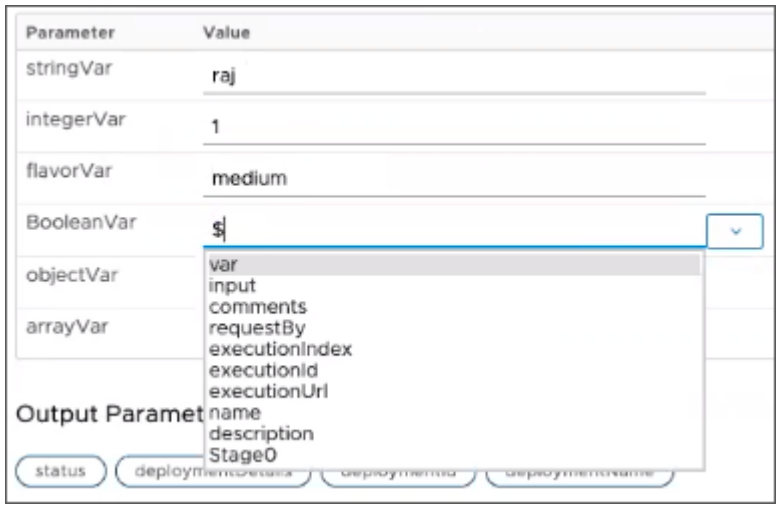
   e 请注意，“部署输入”包括在云模板代码中定义的所有变量，并在云模板代码中显示为已定义。例如：`Integer Var`、`String Var`、`Flavor Var`、`Boolean Var`、`Object Var` 和 `Array Var`。`String Var` 和 `Flavor Var` 是字符串值，`Boolean Var` 是一个复选框。

   f 单击**部署**。

7 在 vRealize Automation Code Stream 中，选择新版本的云模板，然后在**参数和值**表中输入值。云模板支持以下类型的参数，这些参数支持使用美元符号变量实现 vRealize Automation Code Stream 绑定。 vRealize Automation Code Stream 云模板任务的用户界面与 vRealize Automation Cloud Assembly 云模板的用户界面之间存在细微差异。根据 vRealize Automation Cloud Assembly 中云模板的编码，可能不允许在 vRealize Automation Code Stream 的云模板任务中输入值。

a 对于 **flavorVar**，如果云模板将类型定义为字符串或数组，请输入字符串或逗号分隔值数组。例如，数组类似于 **`test, test`**。

b 对于 **BooleanVar**，在下拉菜单中选择 **true** 或 **false**。或者，要使用变量绑定，请输入 **`$`**，然后从列表中选择变量绑定。



c 对于 **objectVar**，按照以下格式输入用大括号和引号括起来的值：**`{"bkix":"bkix":}`**。

d **objectVar** 将传递到云模板，并且可以通过不同的方式使用，具体取决于云模板。允许 JSON 对象使用字符串格式，您可以在键-值表中以逗号分隔值的形式添加键-值对。可以为 JSON 对象输入纯文本，也可以输入键-值对作为 JSON 的常规 stringified 格式。

e 对于 **arrayVar**，按照以下格式将逗号分隔输入值作为数组输入：**`["1","2"]`**。

8 在管道中，可以将输入参数绑定到数组。

a 单击**输入**选项卡。

b 输入输入的名称。例如，**`arrayInput`**。

c 在**参数和值**表中，单击 **arrayVar**，然后输入 **`${input.arrayInput}`**。

d 保存并启用管道后，当管道运行时，必须提供一个数组输入值。例如，输入 **`["1","2"]`**，然后单击**运行**。

现在，您已了解如何在 vRealize Automation Code Stream 管道云模板任务的云模板中使用美元符号 ($) 变量绑定。

如何在管道运行时将参数传递到管道

您可以将输入参数添加到管道，使 vRealize Automation Code Stream 将其传递到管道。当管道运行时，用户必须键入输入参数的值。将输出参数添加到管道中，管道任务可以使用任务中的输出值。 vRealize Automation Code Stream 支持通过多种方式使用参数，以满足您的管道需求。

例如，要在管道运行 REST 任务时提示用户输入其 Git 服务器的 URL，您可以将 REST 任务绑定到 Git 服务器 URL。

要创建变量绑定，请将 URL 绑定变量添加到 REST 任务。当管道运行并到达 REST 任务时，用户必须输入 Git 服务器的 URL。下面介绍了如何创建绑定：

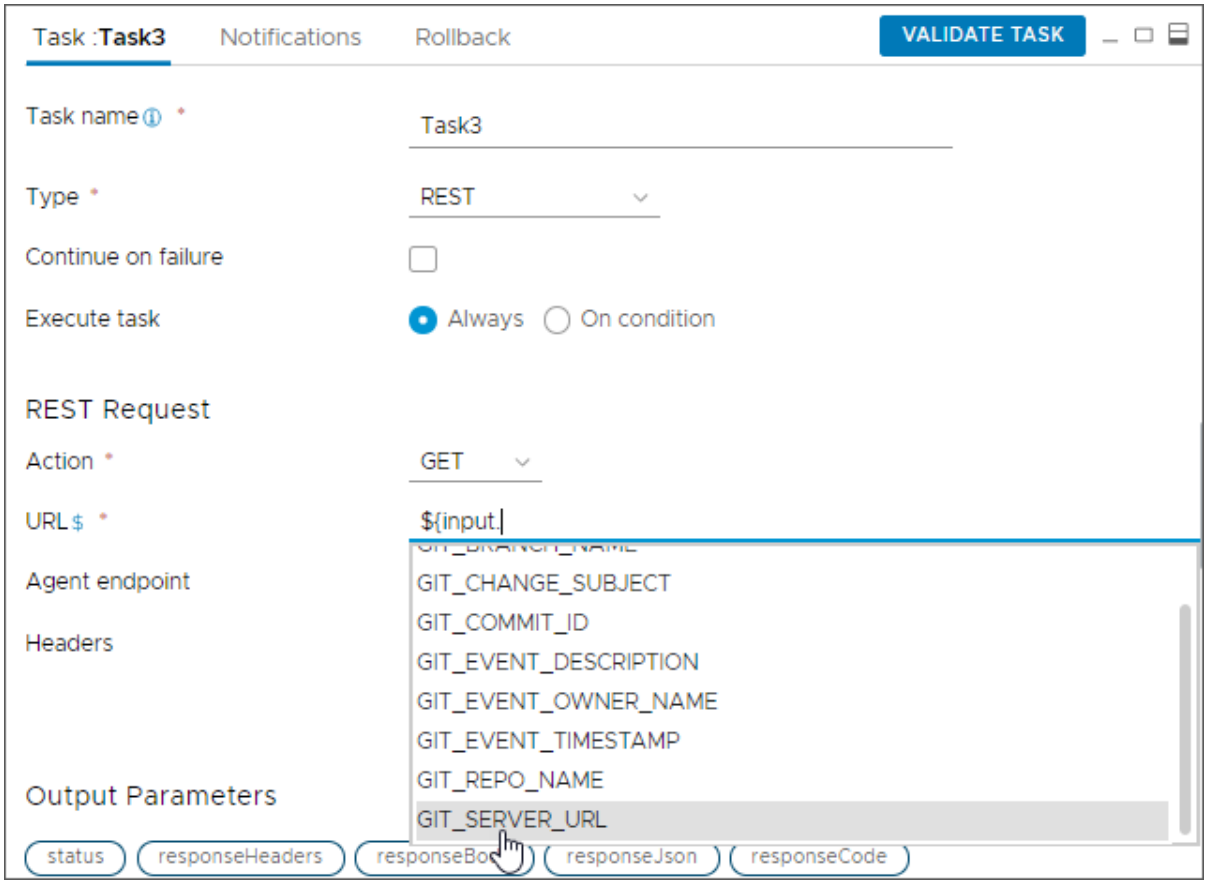
1 在管道中，单击**输入**选项卡。

2 要设置参数，请对**自动插入参数**单击 **Git**。

   此时将显示 Git 参数列表，其中包括 **GIT_SERVER_URL**。如果必须使用 Git 服务器 URL 的默认值，请编辑此参数。

3 单击**型号**，然后单击 REST 任务。

4 在**任务**选项卡的 **URL** 区域中输入 **$**，然后选择**输入**和 **GIT_SERVER_URL**。



   条目类似于：**${input.GIT_SERVER_URL}**

5 要验证任务的变量绑定完整性，请单击**验证任务**。

   vRealize Automation Code Stream 指示任务验证成功。

6 当管道运行 REST 任务时，用户必须输入 Git 服务器的 URL。否则，任务将无法结束运行。

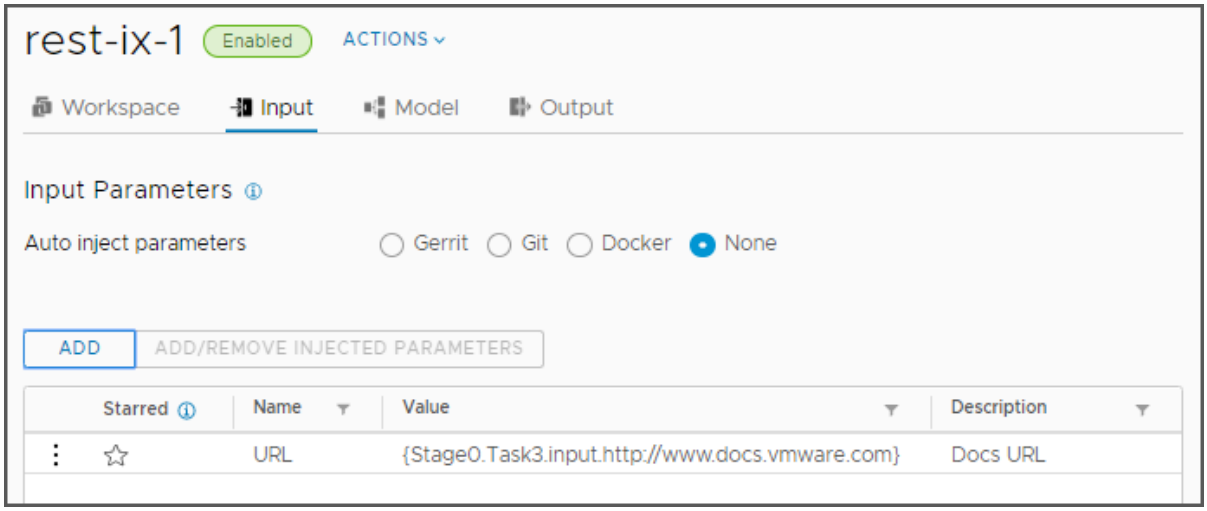
如何通过创建输入参数和输出参数来绑定两个管道任务

将两个任务绑定在一起时，必须将绑定变量添加到接收任务的输入配置中。然后，当管道运行时，用户将绑定变量替换为所需的输入。

要将管道任务绑定在一起，请在输入参数和输出参数中使用美元符号变量 ($)。以下示例展示了如何执行此操作。

假设您需要管道在 REST 任务中调用 URL 并输出响应。要调用 URL 并输出响应，需要在 REST 任务中同时包括输入参数和输出参数。此外，还需要可以批准该任务的用户，并包括用户操作任务，以便其他用户可以在管道运行时进行批准。此示例显示了如何在输入和输出参数中使用表达式，并使管道等待任务的批准。

1 在管道中，单击**输入**选项卡。



2 将**自动插入参数**保留为**无**。

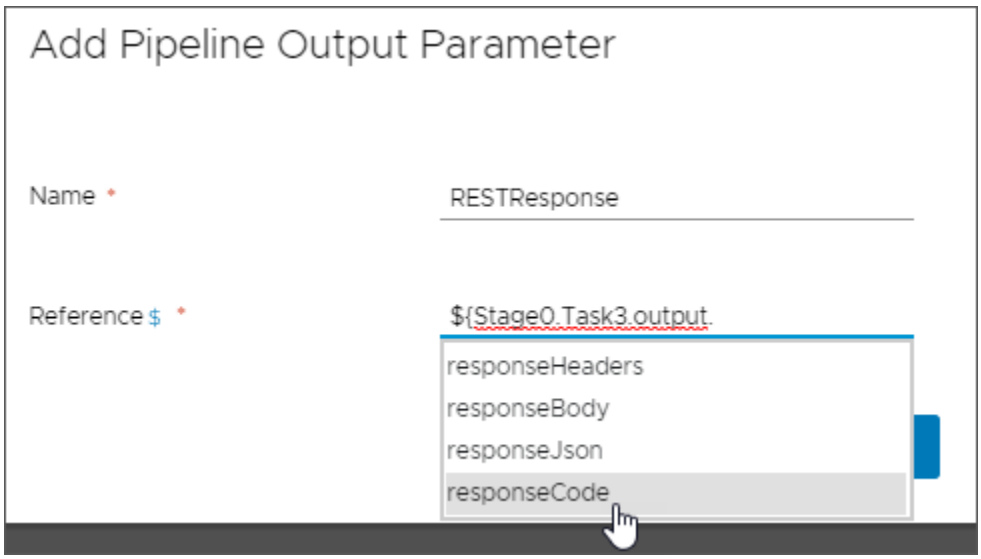
3 单击**添加**，然后输入参数名称、值和说明，然后单击**确定**。例如：

   a 输入 URL 名称。

   b 输入值：`{Stage0.Task3.input.http://www.docs.vmware.com}`。

   c 输入说明。

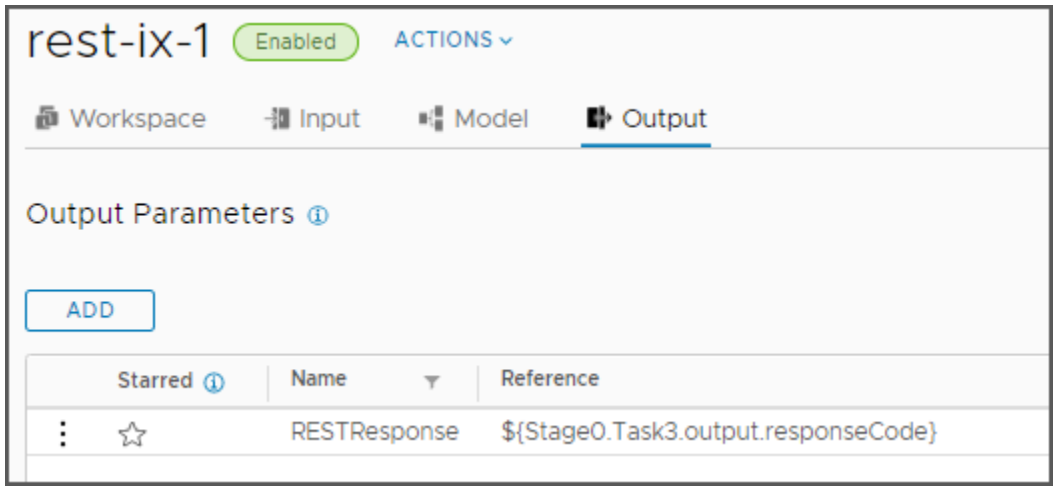
4 单击**输出**选项卡和**添加**，然后输入输出参数的名称和映射。



a 输入唯一的输出参数名称。

b 单击**引用**区域，然后输入 $。

c 在选项弹出时选择相应的选项，以输入任务输出映射。依次选择 **Stage0**、**Task3** 和**输出**，再选择 **responseCode**。然后单击**确定**。

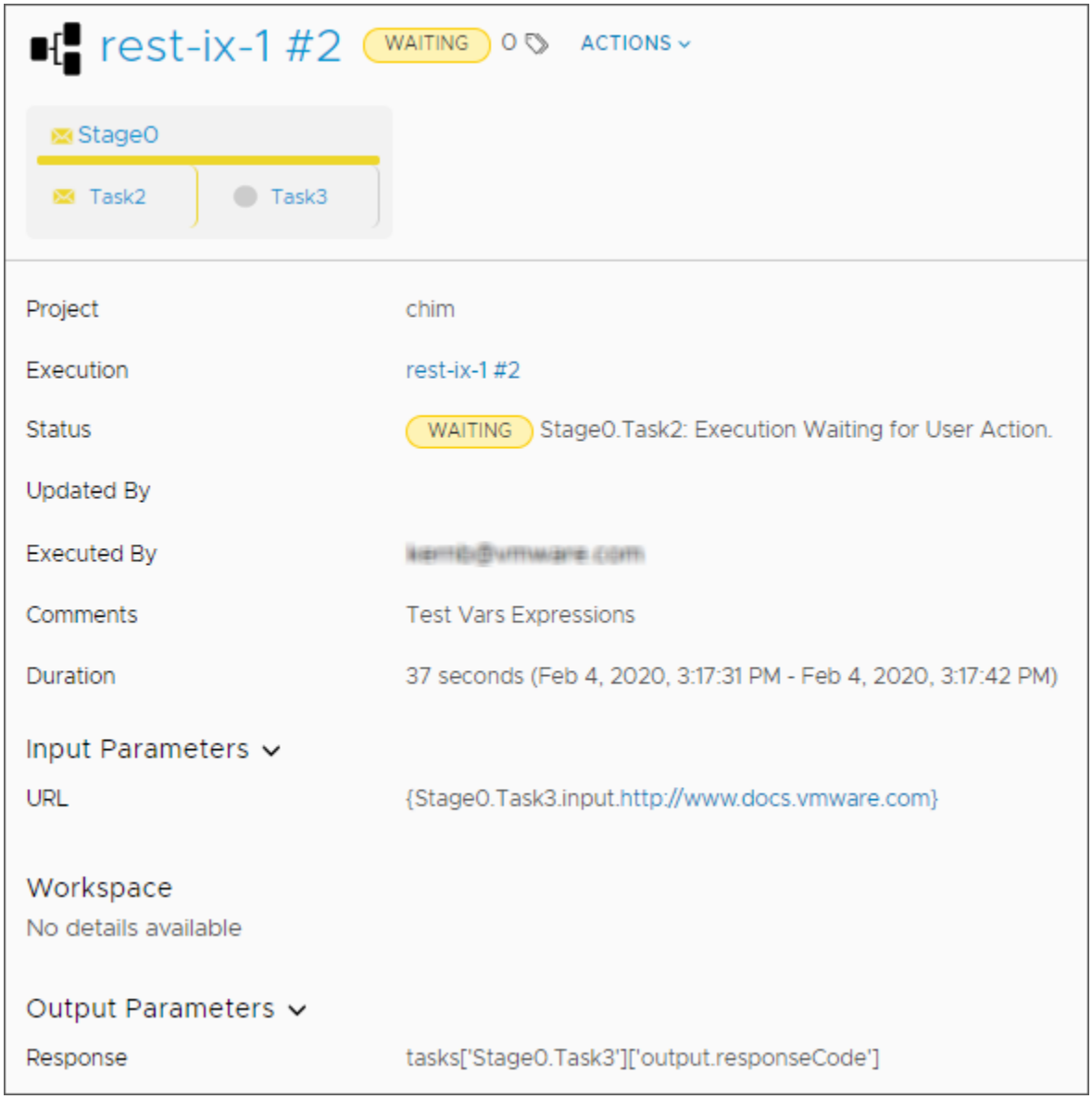


5 保存管道。

6 从**操作**菜单中，单击**运行**。

7 单击**操作 > 查看执行**。

8 单击管道执行，然后查看您定义的输入参数和输出参数。

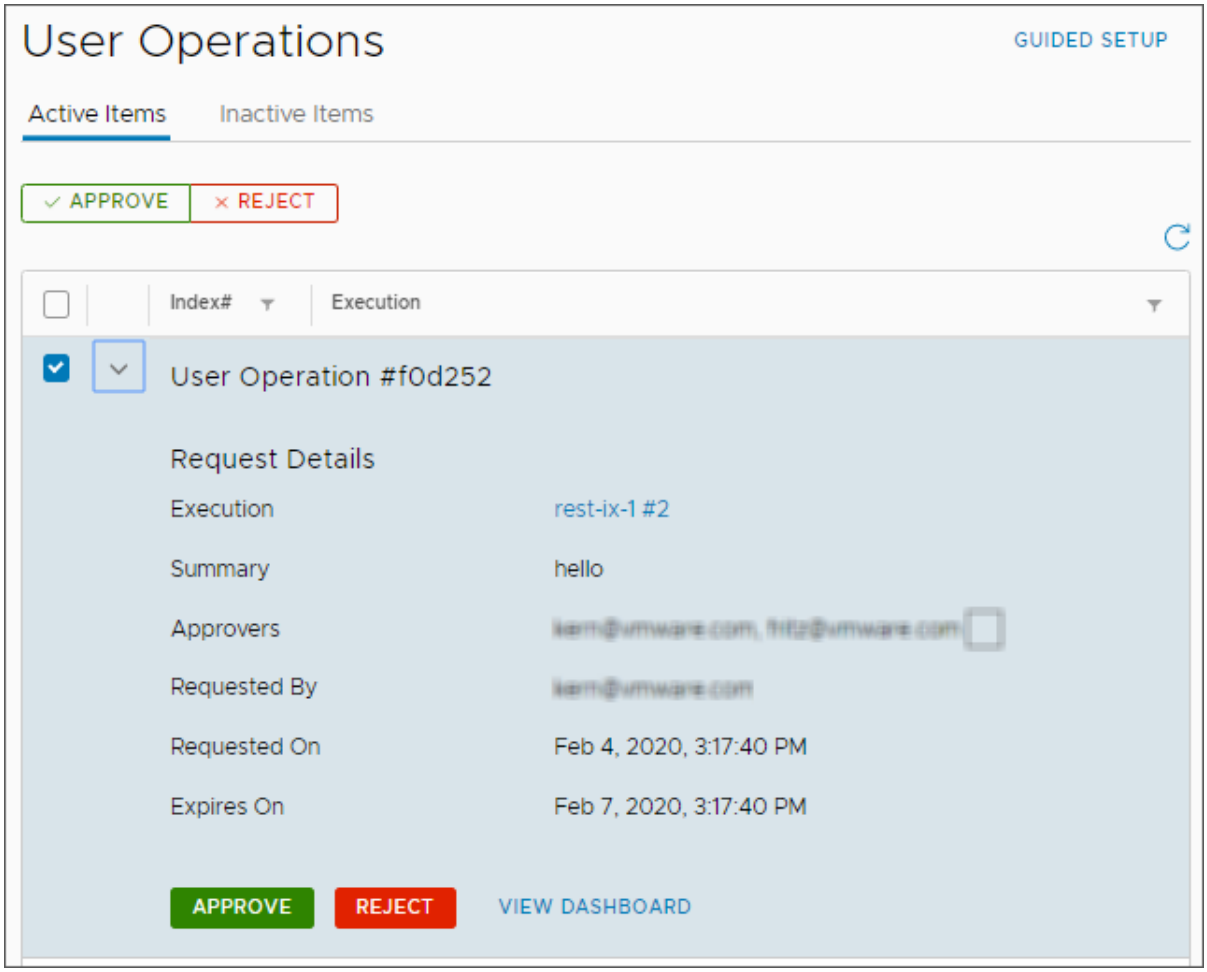


9 要批准管道，请单击**用户操作**，然后在**活动项**选项卡上查看批准列表。或者，留在“执行”中，单击任务，然后单击**批准**。

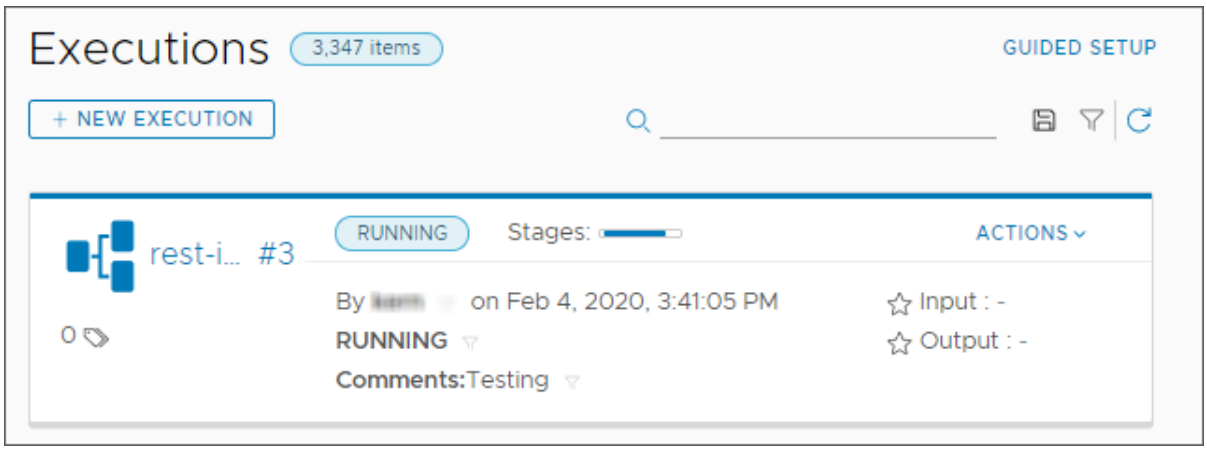
10 要启用**批准**和**拒绝**按钮，请单击执行旁边的复选框。

11 要查看详细信息，请展开下拉箭头。

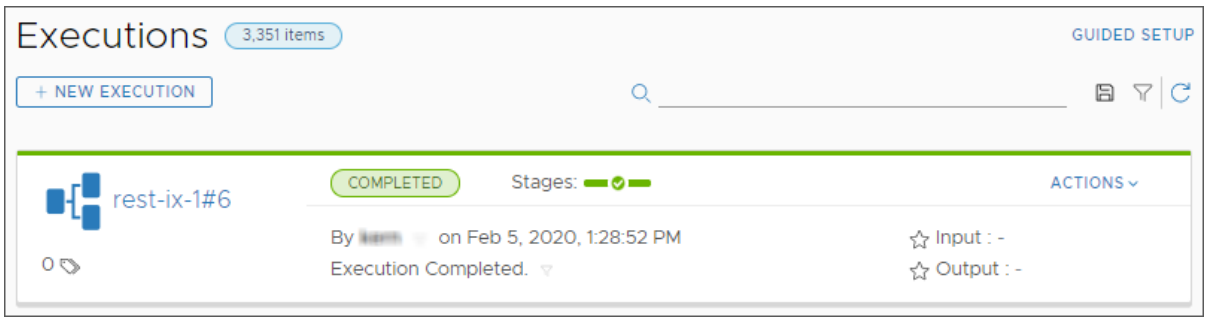
12 要批准任务，请单击**批准**，输入原因，然后单击**确定**。



13 单击**执行**并观察管道是否继续运行。



14 如果管道失败，请更正所有错误，然后保存管道并再次运行。



如何了解有关变量和表达式的更多信息

要查看有关在绑定管道任务时使用变量和表达式的详细信息，请参见在 vRealize Automation Code Stream 中绑定管道任务时可以使用哪些变量和表达式。

要了解如何将管道任务输出与条件变量绑定结合使用，请参见如何在条件任务中使用变量绑定来运行或停止 vRealize Automation Code Stream 中的管道。

如何在条件任务中使用变量绑定来运行或停止 vRealize Automation Code Stream 中的管道

可以让管道中的任务输出根据所提供的条件确定运行或停止管道。要根据任务输出让管道通过或不通过，请使用“条件”任务。

可以使用**条件**任务作为管道中的决策点。通过将“条件”任务与提供的条件表达式结合使用，可以评估管道、阶段和任务中的任何属性。

“条件”任务的结果决定了管道中的下一个任务是否运行。

- 条件为 true 表示允许管道继续运行。
- 条件为 false 表示停止管道。

有关如何通过将任务与“条件”任务绑定在一起，将一个任务的输出值用作下一个任务的输入的示例，请参见如何在 vRealize Automation Code Stream 管道中使用变量绑定。

表 3-7. “条件”任务及其条件表达式与管道的关系

条件任务	影响对象	作用
条件任务	管道	**条件**任务根据任务输出为 true 还是 false 来确定管道在该点是运行还是停止。
条件表达式	条件任务输出	运行管道时，您在**条件**任务中包含的条件表达式会生成 true 或 false 输出状态。例如，条件表达式可能需要“条件”任务输出状态为已完成，或者使用构建号 74。 条件表达式将显示在“任务”选项卡上的“条件”任务中。

条件任务在功能和行为上与其他任务类型中的**基于条件**设置有所不同。

在其他类型的任务中，**基于条件**根据当前任务前提条件表达式的评估结果为 true 还是 false，确定当前任务是否运行，而不是连续执行任务。当管道运行时，**基于条件**设置的条件表达式将针对当前任务生成 true 或 false 输出状态。**基于条件**设置与自己的条件表达式一起显示在“任务”选项卡上。

此示例使用“条件”任务。

前提条件

- 确认管道存在，并包含阶段和任务。

步骤

1 在您的管道中，确定“条件”任务必须出现的决策点。

2 在依赖于其通过或未通过状态的任务之前添加“条件”任务。

3 向“条件”任务添加条件表达式。

例如：

```
"${Stage1.task1.output.status}" == "COMPLETED" || ${input.buildNumber} ==
74
```

4 验证任务。

5 保存管道，然后启用并运行管道

结果

观察管道执行情况，并注意管道是继续运行，还是在“条件”任务处停止。

后续步骤

如果回滚管道部署，也可以使用“条件”任务。例如，在回滚管道中，“条件”任务可帮助 vRealize Automation Code Stream 根据条件表达式标记管道失败，并可以针对各种失败类型触发单个回滚流。

要回滚部署，请参见如何在 vRealize Automation Code Stream 中回滚部署。

在 vRealize Automation Code Stream 中绑定管道任务时可以使用哪些变量和表达式

借助变量和表达式，您可以在管道任务中使用输入参数和输出参数。您输入的参数会将管道任务绑定到一个或多个变量、表达式或条件，并会确定管道的运行行为。

管道可以运行简单或复杂的软件交付解决方案

将管道任务绑定在一起时，可以包括默认表达式和复杂表达式。因此，管道可以运行简单或复杂的软件交付解决方案。

要在管道中创建参数，请单击**输入**或**输出**选项卡，然后通过输入美元符号 **$** 和表达式来添加变量。例如，此参数用作调用 URL 的任务输入：`${Stage0.Task3.input.URL}`。

变量绑定的格式使用名为 SCOPE 和 KEY 的语法组件。`SCOPE` 将上下文定义为输入或输出，`KEY` 则定义详细信息。在参数示例 `${Stage0.Task3.input.URL}` 中，`input` 是 `SCOPE`，URL 是 `KEY`。

任何任务的输出属性都可以解析为任意数量的变量绑定嵌套级别。

要了解有关如何在管道中使用变量绑定的更多信息，请参见如何在 vRealize Automation Code Stream 管道中使用变量绑定。

将美元表达式与 SCOPE 和 KEY 结合使用以绑定管道任务

您可以通过在美元符号变量中使用表达式来将管道任务绑定在一起。将表达式输入为 `${SCOPE.KEY.<PATH>}`。

为确定管道任务的行为，在每个表达式中，`SCOPE` 都是 vRealize Automation Code Stream 使用的上下文。SCOPE 将查找 `KEY`，它定义了任务执行的操作的详细信息。当 `KEY` 的值是嵌套对象时，您可以提供一个可选的 `PATH`。

这些示例介绍了 `SCOPE` 和 `KEY`，并向您展示了如何在管道中使用它们。

表 3-8. 使用 SCOPE 和 KEY

SCOPE	表达式的用途和示例	KEY	如何在管道中使用 Scope 和 Key

input	管道的输入属性：
	`${input.input1}`
		输入属性的名称	要在任务中引用管道的输入属性，请使用以下格式：
			`tasks:`
			`  mytask:`
			`  type: REST`
			`  input:`
			`  url: ${input.url}`
			`  action: get`
			
			`input:`
			`  url: https://www.vmware.com`

output	管道的输出属性：
	`${output.output1}`
		输出属性的名称	要引用用于发送通知的输出属性，请使用以下格式：
			`notifications:`
			`  email:`
			`  - endpoint: MyEmailEndpoint`
			`  subject: "Deployment Successful"`
			`  event: COMPLETED`
			`  to:`
			`  - user@example.org`
			`  body: |`
			`  Pipeline deployed the service successfully. Refer ${output.serviceURL}`

表 3-8. 使用 SCOPE 和 KEY （续）

SCOPE	表达式的用途和示例	KEY	如何在管道中使用 Scope 和 Key

task input	任务的输入：
	`${MY_STAGE.MY_TASK.input.SOMETHING}`
		在通知中指示任务输入	当 Jenkins 作业启动时，它可以引用从任务输入触发的作业的名称。在这种情况下，请使用以下格式发送通知：
			`notifications:`
			`  email:`
			`  - endpoint: MyEmailEndpoint`
			`  stage: MY_STAGE`
			`  task: MY_TASK`
			`  subject: "Build Started"`
			`  event: STARTED`
			`  to:`
			`  - user@example.org`
			`  body: |`
			`  Jenkins job ${MY_STAGE.MY_TASK.input.job} started for commit id ${input.COMMITID}.`

task output	任务的输出：
	`${MY_STAGE.MY_TASK.output.SOMETHING}`
		在后续任务中指示任务的输出	要在任务 2 中引用管道任务 1 的输出，请使用以下格式：
			`taskOrder:`
			`  - task1`
			`  - task2`
			`tasks:`
			`  task1:`
			`  type: REST`
			`  input:`
			`  action: get`
			`  url: https://www.example.org/api/status`
			`  task2:`
			`  type: REST`
			`  input:`
			`  action: post`
			`  url: https://status.internal.example.org/api/activity`
			`  payload: ${MY_STAGE.task1.output.responseBody}`

表 3-8. 使用 SCOPE 和 KEY （续）

SCOPE	表达式的用途和示例	KEY	如何在管道中使用 Scope 和 Key

var	变量：
	`${var.myVariable}`
		引用端点中的变量	要对密码引用端点中的机密变量，请使用以下格式：
			`---`
			`project: MyProject`
			`kind: ENDPOINT`
			`name: MyJenkinsServer`
			`type: jenkins`
			`properties:`
			`  url: https://jenkins.example.com`
			`  username: jenkinsUser`
			`  password: ${var.jenkinsPassword}`

var	变量：
	`${var.myVariable}`
		在管道中引用变量	要在管道 URL 中引用变量，请使用以下格式：
			`tasks:`
			`  task1:`
			`  type: REST`
			`  input:`
			`  action: get`
			`  url: ${var.MY_SERVER_URL}`

task status	任务的状态：
	`${MY_STAGE.MY_TASK.status}`
	`${MY_STAGE.MY_TASK.statusMessage}`		

stage status	阶段的状态：
	`${MY_STAGE.status}`
	`${MY_STAGE.statusMessage}`		

默认表达式

可在管道中使用变量和表达式。此摘要包括您可以使用的默认表达式。

表达式	说明
`${comments}`	管道执行请求时提供的注释。
`${duration}`	管道执行的持续时间。

表达式	说明
${endTime}	执行管道的结束时间（如果结束）(UTC)。
${executedOn}	管道执行的开始时间 (UTC)，与开始时间相同。
${executionId}	管道执行的 ID。
${executionUrl}	在用户界面中导航到管道执行的 URL。
${name}	管道的名称。
${requestBy}	请求执行的用户的名称。
${stageName}	当前阶段在阶段的范围内使用时的名称。
${startTime}	管道执行的开始时间 (UTC)。
${status}	执行的状态。
${statusMessage}	管道执行的状态消息。
${taskName}	当前任务在任务输入或通知中使用时的名称。

在管道任务中使用 SCOPE 和 KEY

您可以将表达式与任何支持的管道任务结合使用。以下示例展示了如何定义 SCOPE 和 KEY 并确认语法。这些代码示例使用 MY_STAGE 和 MY_TASK 作为管道阶段和任务名称。

要了解有关可用任务的更多信息，请参见 vRealize Automation Code Stream 中提供哪些类型的任务。

表 3-9. 控制任务

任务	Scope	Key	如何在任务中使用 Scope 和 Key
用户操作			
	Input	summary：用户操作请求的摘要 description：用户操作请求的描述 approvers：审批者电子邮件地址列表，其中每个条目可以是包含逗号的变量，也可以使用分号分隔电子邮件 approverGroups：平台和身份的审批者组地址列表 sendemail：（可选）如果设置为 true，则可以根据请求或响应发送电子邮件通知 expirationInDays：表示请求的到期时间的天数	${MY_STAGE.MY_TASK.input.summary} ${MY_STAGE.MY_TASK.input.description} ${MY_STAGE.MY_TASK.input.approvers} ${MY_STAGE.MY_TASK.input.approverGroups} ${MY_STAGE.MY_TASK.input.sendemail} ${MY_STAGE.MY_TASK.input.expirationInDays}

表 3-9. 控制任务 （续）

任务	Scope	Key	如何在任务中使用 Scope 和 Key

	Output	`index`：表示请求的 6 位十六进制字符串
		`respondedBy`：批准/拒绝用户操作的人员的帐户名称
		`respondedByEmail`：响应人员的电子邮件地址
		`comments`：响应期间提供的注释
			`${MY_STAGE.MY_TASK.output.index}`
			`${MY_STAGE.MY_TASK.output.respondedBy}`
			`${MY_STAGE.MY_TASK.output.respondedByEmail}`
			`${MY_STAGE.MY_TASK.output.comments}`

条件			
	Input	`condition`：要评估的条件。当条件的评估结果为 true 时，会将该任务标记为完成，而其他响应将导致任务失败	`${MY_STAGE.MY_TASK.input.condition}`
	Output	`result`：评估结果	`${MY_STAGE.MY_TASK.output.response}`

表 3-10. 管道任务

任务	Scope	Key	如何在任务中使用 Scope 和 Key
管道			
	Input	`name`：要运行的管道的名称 `inputProperties`：要传递到嵌套管道执行的输入属性	`${MY_STAGE.MY_TASK.input.name}` `${MY_STAGE.MY_TASK.input.inputProperties}` # 引用所有属性 `${MY_STAGE.MY_TASK.input.inputProperties.input1}` # 引用 input1 的值

	Output	`executionStatus`：管道执行的状态
		`executionIndex`：管道执行的索引
		`outputProperties`：管道执行的输出属性
			`${MY_STAGE.MY_TASK.output.executionStatus}`
			`${MY_STAGE.MY_TASK.output.executionIndex}`
			`${MY_STAGE.MY_TASK.output.outputProperties}` # 引用所有属性
			`${MY_STAGE.MY_TASK.output.outputProperties.output1}` # 引用 output1 的值

表 3-11. 自动执行持续集成任务

任务	Scope	Key	如何在任务中使用 SCOPE 和 KEY
CI			

	Input	`steps`：一组字符串，表示要运行的命令
		`export`：运行步骤后要保留的环境变量
		`artifacts`：要保留在共享路径中的工件的路径
		`process`：用于 JUnit、JaCoCo、Checkstyle、FindBugs 处理的配置元素集
			`${MY_STAGE.MY_TASK.input.steps}`
			`${MY_STAGE.MY_TASK.input.export}`
			`${MY_STAGE.MY_TASK.input.artifacts}`
			`${MY_STAGE.MY_TASK.input.process}`
			`${MY_STAGE.MY_TASK.input.process[0].path}` # 引用第一个配置的路径

	Output	`export`：键值对，表示从输入 `exports` 中导出的环境变量
		`artifacts`：已成功保留的工件的路径
		`process`：输入 `processResponse` 的已处理结果集
			`${MY_STAGE.MY_TASK.output.exports}` # 引用所有导出
			`${MY_STAGE.MY_TASK.output.exports.myvar}` # 引用 **`myvar`** 的值
			`${MY_STAGE.MY_TASK.output.artifacts}`
			`${MY_STAGE.MY_TASK.output.processResponse}`
			`${MY_STAGE.MY_TASK.output.processResponse[0].result}` # 第一个进程配置的结果

自定义			

	Input	`name`：自定义集成的名称
		`version`：自定义集成版本（已发布或已弃用）
		`properties`：要发送到自定义集成的属性
			`${MY_STAGE.MY_TASK.input.name}`
			`${MY_STAGE.MY_TASK.input.version}`
			`${MY_STAGE.MY_TASK.input.properties}` # 引用所有属性
			`${MY_STAGE.MY_TASK.input.properties.property1}` # 引用 property1 的值

	Output	`properties`：自定义集成响应中的输出属性	`${MY_STAGE.MY_TASK.output.properties}` # 引用所有属性 `${MY_STAGE.MY_TASK.output.properties.property1}` # 引用 property1 的值

表 3-12. 自动执行持续部署任务：云模板

任务	Scope	Key	如何在任务中使用 Scope 和 Key
云模板			
	Input	`action`：**createDeployment**、**updateDeployment**、**deleteDeployment**、**rollbackDeployment** 之 `blueprintInputParams`：用于**创建部署**和**更新部署**操作 `allowDestroy`：可以在更新部署过程中销毁计算机。 **CREATE_DEPLOYMENT** ■ `blueprintName`：云模板的名称 ■ `blueprintVersion`：云模板的版本 或 ■ `fileUrl`：选择 GIT 服务器后远程云模板 YAML 的 URL。 **UPDATE_DEPLOYMENT** 以下任意组合： ■ `blueprintName`：云模板的名称 ■ `blueprintVersion`：云模板的版本 或 ■ `fileUrl`：选择 GIT 服务器后远程云模板 YAML 的 URL。 `------` ■ `deploymentId`：部署的 ID 或 ■ `deploymentName`：部署的名称 `------` **DELETE_DEPLOYMENT** ■ `deploymentId`：部署的 ID 或 ■ `deploymentName`：部署的名称 **ROLLBACK_DEPLOYMENT** 以下任意组合： ■ `deploymentId`：部署的 ID 或 ■ `deploymentName`：部署的名称	

表 3-12. 自动执行持续部署任务：云模板 （续）

任务	Scope	Key	如何在任务中使用 Scope 和 Key
		------ ■ `blueprintName`：云模板的名称 ■ `rollbackVersion`：要回滚到的版本	
	Output		可以绑定到其他任务或管道输出的参数： ■ 部署名称可以作为 **`${Stage0.Task0.output.deploymentName}`** 进行访问 ■ 部署 ID 可以作为 **`${Stage0.Task0.output.deploymentId}`** 进行访问 ■ 部署详细信息是一个复合对象，可以使用 JSON 结果访问内部详细信息。 要访问任何属性，请使用点运算符遵循 JSON 层次结构。例如，要访问资源 Cloud_Machine_1[0] 的地址，则 **$** 绑定为： **`${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}`** 同样，对于特定实例，**$** 绑定为： **`${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}`** 在 vRealize Automation Code Stream 用户界面中，可以获取任何属性的 **$** 绑定。 1 在任务输出属性区域中，单击**查看输出 JSON**。 2 要查找 **$** 绑定，请输入任何属性。 3 单击搜索图标，之后，将显示相应的 **$** 绑定。

JSON 输出示例：

部署详细信息对象示例：


```
{
    "id": "6a031f92-d0fa-42c8-bc9e-3b260ee2f65b",
    "name": "deployment_6a031f92-d0fa-42c8-bc9e-3b260ee2f65b",
    "description": "Pipeline Service triggered operation",
    "orgId": "434f6917-4e34-4537-b6c0-3bf3638a71bc",
    "blueprintId": "8d1dd801-3a32-4f3b-adde-27f8163dfe6f",
    "blueprintVersion": "1",
    "createdAt": "2020-08-27T13:50:24.546215Z",
    "createdBy": "user@vmware.com",
    "lastUpdatedAt": "2020-08-27T13:52:50.674957Z",
    "lastUpdatedBy": "user@vmware.com",
    "inputs": {},
    "simulated": false,
    "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
    "resources": {
        "Cloud_Machine_1[0]": {
            "id": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
            "name": "Cloud_Machine_1[0]",
            "powerState": "ON",
            "address": "10.108.79.33",
            "resourceLink": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
            "componentTypeId": "Cloud.vSphere.Machine",
            "endpointType": "vsphere",
            "resourceName": "Cloud_Machine_1-mcm110615-146929827053",
            "resourceId": "1606fbcd-40e5-4edc-ab85-7b559aa986ad",
            "resourceDescLink": "/resources/compute-descriptions/1952d1d3-15f0-4574-
ae42-4fbf8a87d4cc",
            "zone": "Automation / Vms",
            "countIndex": "0",
            "image": "ubuntu",
            "count": "1",
            "flavor": "small",
            "region": "MYBU",
            "_clusterAllocationSize": "1",
            "osType": "LINUX",
            "componentType": "Cloud.vSphere.Machine",
            "account": "bha"
        }
    },
    "status": "CREATE_SUCCESSFUL",
    "deploymentURI": "https://api.yourenv.com/automation-ui/#/deployment-ui;ash=/deployment/
6a031f92-d0fa-42c8-bc9e-3b260ee2f65b"
}
```

表 3-13. 自动执行持续部署任务：Kubernetes

任务	Scope	Key	如何在任务中使用 Scope 和 Key
Kubernetes			

	Input	`action`：**GET**、**CREATE**、**APPLY**、**DELETE**、**ROLLBACK** 之一
		■ `timeout`：任何操作的整体超时
		■ `filterByLabel`：要针对操作 **GET** 使用 K8S labelSelector 筛选的其他标签：
		GET、**CREATE**、**DELETE**、**APPLY**
		■ `yaml`：要处理并发送到 Kubernetes 的内嵌 YAML
		■ `parameters`：键值对 - 将 **$$KEY** 替换为内嵌 YAML 输入区域中的 **VALUE**：
		■ `filePath`：SCM Git 端点（如果已提供）中要从中获取 YAML 的相对路径
		■ `scmConstants`：键值对 - 将 **$${KEY}** 替换为通过 SCM 获取的 YAML 中的 **VALUE**。
		■ `continueOnConflict`：设置为 true 时，如果资源已存在，任务将继续执行。
		ROLLBACK
		■ `resourceType`：要回滚的资源类型
		■ `resourceName`：要回滚的资源名称
		■ `namespace`：必须执行回滚的命名空间
		■ `revision`：要回滚到的修订版本
			`${MY_STAGE.MY_TASK.input.action}` # 确定要执行的操作。
			`${MY_STAGE.MY_TASK.input.timeout}`
			`${MY_STAGE.MY_TASK.input.filterByLabel}`
			`${MY_STAGE.MY_TASK.input.yaml}`
			`${MY_STAGE.MY_TASK.input.parameters}`
			`${MY_STAGE.MY_TASK.input.filePath}`
			`${MY_STAGE.MY_TASK.input.scmConstants}`
			`${MY_STAGE.MY_TASK.input.continueOnConflict}`
			`${MY_STAGE.MY_TASK.input.resourceType}`
			`${MY_STAGE.MY_TASK.input.resourceName}`
			`${MY_STAGE.MY_TASK.input.namespace}`
			`${MY_STAGE.MY_TASK.input.revision}`

	Output	`response`：捕获整个响应 `response.<RESOURCE>`：资源对应于 configMaps、部署、端点、输入、作业、命名空间、容器、replicaSets、Replicationcontroller、密钥、服务、statefulSets、节点、loadBalancers。 `response.<RESOURCE>.<KEY>`：键对应于 apiVersion、种类、元数据、规范之一	`${MY_STAGE.MY_TASK.output.response}` `${MY_STAGE.MY_TASK.output.response.}`

表 3-14. 集成开发、测试和部署应用程序

任务	Scope	Key	如何在任务中使用 SCOPE 和 KEY
Bamboo			

	Input	`plan`：计划的名称
		`planKey`：计划键
		`variables`：要传递到计划的变量
		`parameters`：要传递到计划的参数
			`${MY_STAGE.MY_TASK.input.plan}`
			`${MY_STAGE.MY_TASK.input.planKey}`
			`${MY_STAGE.MY_TASK.input.variables}`
			`${MY_STAGE.MY_TASK.input.parameters}` # 引用所有参数
			`${MY_STAGE.MY_TASK.input.parameters.param1}` # 引用 param1 的值

表 3-14. 集成开发、测试和部署应用程序 （续）

任务	Scope	Key	如何在任务中使用 SCOPE 和 KEY

	Output	`resultUrl`：生成的构建的 URL
		`buildResultKey`：生成的构建的键
		`buildNumber`：内部版本号
		`buildTestSummary`：运行的测试摘要
		`successfulTestCount`：通过的测试结果数
		`failedTestCount`：失败的测试结果数
		`skippedTestCount`：跳过的测试结果数
		`artifacts`：内部版本中的工件
			`${MY_STAGE.MY_TASK.output.resultUrl}`
			`${MY_STAGE.MY_TASK.output.buildResultKey}`
			`${MY_STAGE.MY_TASK.output.buildNumber}`
			`${MY_STAGE.MY_TASK.output.buildTestSummary}` # 引用所有结果
			`${MY_STAGE.MY_TASK.output.successfulTestCount}` # 引用特定的测试计数
			`${MY_STAGE.MY_TASK.output.buildNumber}`

Jenkins			
	Input	`job`：Jenkins 作业的名称 `parameters`：要传递到作业的参数	`${MY_STAGE.MY_TASK.input.job}` `${MY_STAGE.MY_TASK.input.parameters}` # 引用所有参数 `${MY_STAGE.MY_TASK.input.parameters.param1}` # 引用参数的值

	Output	`job`：Jenkins 作业的名称
		`jobId`：生成的作业的 ID，例如 1234
		`jobStatus`：Jenkins 中的状态
		`jobResults`：测试/代码覆盖率结果的集合
		`jobUrl`：生成的作业运行的 URL
			`${MY_STAGE.MY_TASK.output.job}`
			`${MY_STAGE.MY_TASK.output.jobId}`
			`${MY_STAGE.MY_TASK.output.jobStatus}`
			`${MY_STAGE.MY_TASK.output.jobResults}` # 引用所有结果
			`${MY_STAGE.MY_TASK.output.jobResults.junitResponse}` # 引用 JUnit 结果
			`${MY_STAGE.MY_TASK.output.jobResults.jacocoRespose}` # 引用 JaCoCo 结果
			`${MY_STAGE.MY_TASK.output.jobUrl}`

TFS			

	Input	`projectCollection`：来自 TFS 的项目集合
		`teamProject`：从可用集合中选择的项目
		`buildDefinitionId`：要运行的构建定义 ID
			`${MY_STAGE.MY_TASK.input.projectCollection}`
			`${MY_STAGE.MY_TASK.input.teamProject}`
			`${MY_STAGE.MY_TASK.input.buildDefinitionId}`

	Output	`buildId`：生成的构建 ID
		`buildUrl`：用于访问构建摘要的 URL
		`logUrl`：用于访问日志的 URL
		`dropLocation`：工件（如果有）的放置位置
			`${MY_STAGE.MY_TASK.output.buildId}`
			`${MY_STAGE.MY_TASK.output.buildUrl}`
			`${MY_STAGE.MY_TASK.output.logUrl}`
			`${MY_STAGE.MY_TASK.output.dropLocation}`

vRO			

表 3-14. 集成开发、测试和部署应用程序 （续）

任务	Scope	Key	如何在任务中使用 SCOPE 和 KEY
	Input	workflowId：要运行的工作流的 ID parameters：要传递到工作流的参数	${MY_STAGE.MY_TASK.input.workflowId} ${MY_STAGE.MY_TASK.input.parameters}
	Output	workflowExecutionId：工作流执行的 ID properties：工作流执行的输出属性	${MY_STAGE.MY_TASK.output.workflowExecutionId} ${MY_STAGE.MY_TASK.output.properties}

表 3-15. 通过 API 集成其他应用程序

任务	Scope	Key	如何在任务中使用 SCOPE 和 KEY
REST			
	Input	url：要调用的 URL action：要使用的 TTP 方法 headers：要传递的 HTTP 标头 payload：请求负载 fingerprint：URL 为 https 时要匹配的指纹 allowAllCerts：设置为 true 时，可以是任何具有 https URL 的证书	${MY_STAGE.MY_TASK.input.url} ${MY_STAGE.MY_TASK.input.action} ${MY_STAGE.MY_TASK.input.headers} ${MY_STAGE.MY_TASK.input.payload} ${MY_STAGE.MY_TASK.input.fingerprint} ${MY_STAGE.MY_TASK.input.allowAllCerts}
	Output	responseCode：HTTP 响应代码 responseHeaders：HTTP 响应标头 responseBody：收到的响应的字符串格式 responseJson：Traversable 响应（如果 content-type 为 **application/json**）	${MY_STAGE.MY_TASK.output.responseCode} ${MY_STAGE.MY_TASK.output.responseHeaders} ${MY_STAGE.MY_TASK.output.responseHeaders.header1} # 引用响应标头“header1” ${MY_STAGE.MY_TASK.output.responseBody} ${MY_STAGE.MY_TASK.output.responseJson} # 引用 JSON 响应 ${MY_STAGE.MY_TASK.output.responseJson.a.b.c} # 在响应中的 a.b.c JSON 路径之后引用嵌套对象
Poll			

表 3-15. 通过 API 集成其他应用程序 （续）

任务	Scope	Key	如何在任务中使用 SCOPE 和 KEY

	Input	`url`：要调用的 URL
		`headers`：要传递的 HTTP 标头
		`exitCriteria`：任务成功或失败需要满足的条件。“成功”的键值对 → 表达式，“失败”→ 表达式
		`pollCount`：要执行的迭代次数。vRealize Automation Code Stream 管理员最多可以将轮询计数设置为 10000。
		`pollIntervalSeconds`：每次迭代之间等待的秒数。轮询时间间隔必须大于或等于 60 秒。
		`ignoreFailure`：设置为 true 时，将忽略中间响应失败
		`fingerprint`：URL 为 https 时要匹配的指纹
		`allowAllCerts`：设置为 true 时，可以是任何具有 https URL 的证书
			`${MY_STAGE.MY_TASK.input.url}`
			`${MY_STAGE.MY_TASK.input.headers}`
			`${MY_STAGE.MY_TASK.input.exitCriteria}`
			`${MY_STAGE.MY_TASK.input.pollCount}`
			`${MY_STAGE.MY_TASK.input.pollIntervalSeconds}`
			`${MY_STAGE.MY_TASK.input.ignoreFailure}`
			`${MY_STAGE.MY_TASK.input.fingerprint}`
			`${MY_STAGE.MY_TASK.input.allowAllCerts}`

	Output	`responseCode`：HTTP 响应代码
		`responseBody`：收到的响应的字符串格式
		`responseJson`：Traversable 响应（如果 content-type 为 **application/json**）
			`${MY_STAGE.MY_TASK.output.responseCode}`
			`${MY_STAGE.MY_TASK.output.responseBody}`
			`${MY_STAGE.MY_TASK.output.responseJson} # Refer to response as JSON`

表 3-16. 运行远程脚本和用户定义的脚本

任务	Scope	Key	如何在任务中使用 SCOPE 和 KEY
PowerShell 要运行 PowerShell 任务，您必须： ■ 具有与远程 Windows 主机的活动会话。 ■ 如果您想要输入 base64 PowerShell 命令，请先计算总命令长度。有关详细信息，请参见 vRealize Automation Code Stream 中提供哪些类型的任务。			

	Input	`host`：计算机的 IP 地址或主机名
		`username`：用于连接的用户名
		`password`：用于连接的密码
		`useTLS`：尝试 https 连接
		`trustCert`：设置为 true 时，信任自签名证书
		`script`：要运行的脚本
		`workingDirectory`：运行脚本之前要切换到的目录路径
		`environmentVariables`：要设置的环境变量的键-值对
		`arguments`：要传递到脚本的参数
			`${MY_STAGE.MY_TASK.input.host}`
			`${MY_STAGE.MY_TASK.input.username}`
			`${MY_STAGE.MY_TASK.input.password}`
			`${MY_STAGE.MY_TASK.input.useTLS}`
			`${MY_STAGE.MY_TASK.input.trustCert}`
			`${MY_STAGE.MY_TASK.input.script}`
			`${MY_STAGE.MY_TASK.input.workingDirectory}`
			`${MY_STAGE.MY_TASK.input.environmentVariables}`
			`${MY_STAGE.MY_TASK.input.arguments}`

	Output	`response`：文件 `$SCRIPT_RESPONSE_FILE` 的内容
		`responseFilePath`：`$SCRIPT_RESPONSE_FILE` 的值
		`exitCode`：进程退出代码
		`logFilePath`：包含 stdout 的文件的路径
		`errorFilePath`：包含 stderr 的文件的路径
			`${MY_STAGE.MY_TASK.output.response}`
			`${MY_STAGE.MY_TASK.output.responseFilePath}`
			`${MY_STAGE.MY_TASK.output.exitCode}`
			`${MY_STAGE.MY_TASK.output.logFilePath}`
			`${MY_STAGE.MY_TASK.output.errorFilePath}`

SSH			

表 3-16. 运行远程脚本和用户定义的脚本 （续）

任务	Scope	Key	如何在任务中使用 SCOPE 和 KEY
	Input	host：计算机的 IP 地址或主机名 username：用于连接的用户名 password：用于连接的密码（可以选择性地使用 privateKey） privateKey：用于连接的 privateKey passphrase：用于解除锁定 privateKey 的可选密码短语 script：要运行的脚本 workingDirectory：运行脚本之前要切换到的目录路径 environmentVariables：要设置的环境变量的键-值对	${MY_STAGE.MY_TASK.input.host} ${MY_STAGE.MY_TASK.input.username} ${MY_STAGE.MY_TASK.input.password} ${MY_STAGE.MY_TASK.input.privateKey} ${MY_STAGE.MY_TASK.input.passphrase} ${MY_STAGE.MY_TASK.input.script} ${MY_STAGE.MY_TASK.input.workingDirectory} ${MY_STAGE.MY_TASK.input.environmentVariables}
	Output	response：文件 $SCRIPT_RESPONSE_FILE 的内容 responseFilePath：$SCRIPT_RESPONSE_FILE 的值 exitCode：进程退出代码 logFilePath：包含 stdout 的文件的路径 errorFilePath：包含 stderr 的文件的路径	${MY_STAGE.MY_TASK.output.response} ${MY_STAGE.MY_TASK.output.responseFilePath} ${MY_STAGE.MY_TASK.output.exitCode} ${MY_STAGE.MY_TASK.output.logFilePath} ${MY_STAGE.MY_TASK.output.errorFilePath}

如何在任务之间使用变量绑定

此示例说明了如何在管道任务中使用变量绑定。

表 3-17. 语法格式示例

示例	语法
为管道通知和管道输出属性使用任务输出值	${<Stage Key>.<Task Key>.output.<Task output key>}
引用以前的任务输出值，作为当前任务的输入	${<Previous/Current Stage key>.<Previous task key not in current Task group>.output.<task output key>}

了解更多

要了解有关任务中绑定变量的更多信息，请参见：

- 如何在 vRealize Automation Code Stream 管道中使用变量绑定
- 如何在条件任务中使用变量绑定来运行或停止 vRealize Automation Code Stream 中的管道
- vRealize Automation Code Stream 中提供哪些类型的任务

如何发送有关 vRealize Automation Code Stream 中的管道的通知

通知是与团队进行通信，让他们了解管道在 vRealize Automation Code Stream 中的状态的方式。

要在管道运行时发送通知，可以基于整个管道、阶段或任务的状态配置 vRealize Automation Code Stream 通知。

- 电子邮件通知会在以下时候发送电子邮件：
  - 管道完成、等待、失败、取消或启动。
  - 阶段完成、失败或启动。
  - 任务完成、等待、失败或启动。
- 票证通知会在以下时候创建一个票证并将其分配给团队成员：
  - 管道失败或完成。
  - 阶段失败。
  - 任务失败。
- Webhook 通知会在以下时候向另一个应用程序发送一个请求：
  - 管道失败、完成、等待、取消或启动。
  - 阶段失败、完成或启动。
  - 任务失败、完成、等待或启动。

例如，可以在用户操作任务上配置一个电子邮件通知，以便在管道中的特定点获得批准。当管道运行时，此任务向必须批准该任务的人员发送电子邮件。如果用户操作任务设置了过期超时（以天、小时或分钟为单位），则要求的用户必须在任务过期之前批准管道。否则，管道将会按预期失败。

要在管道任务失败时创建 Jira 票证，可以配置通知。或者，要根据管道事件向 Slack 通道发送有关管道状态的请求，可以配置 Webhook 通知。

可以在所有类型的通知中使用变量。例如，可以在 Webhook 通知的 URL 中使用 **`${var}`**。

前提条件

- 验证是否是否以创建一个或多个管道。请参见第 5 章 vRealize Automation Code Stream 使用教程中的用例。
- 要发送电子邮件通知，请确认您可以访问正在运行的电子邮件服务器。要获得帮助，请咨询您的管理员。
- 要创建票证（如 Jira 票证），请确认端点存在。请参见什么是 vRealize Automation Code Stream 中的端点 。
- 要基于集成发送通知，可以创建 Webhook 通知。然后，确认 Webhook 已添加并正常运行。您可以将通知用于诸如 Slack、GitHub 或 GitLab 等应用程序。

步骤

1 打开管道。

2 要为整个管道的状态或阶段或任务的状态创建通知：

为以下状态创建通知：	执行的操作：
管道状态	单击管道画布上的空白区域。
阶段状态	单击管道的某个阶段中的空白区域。
任务的状态	单击管道的某个阶段中的任务。

3 单击**通知**选项卡。

4 单击**添加**，选择通知类型，并配置通知详细信息。

5 要在管道成功时创建 Slack 通知，请创建一个 Webhook 通知。

   a 选择 **Webhook**。

   b 要配置 Slack 通知，请输入信息。

   c 单击**保存**。

   d 当管道运行时，Slack 通道将收到管道状态的通知。例如，用户可能会在 Slack 通道上看到以下内容：

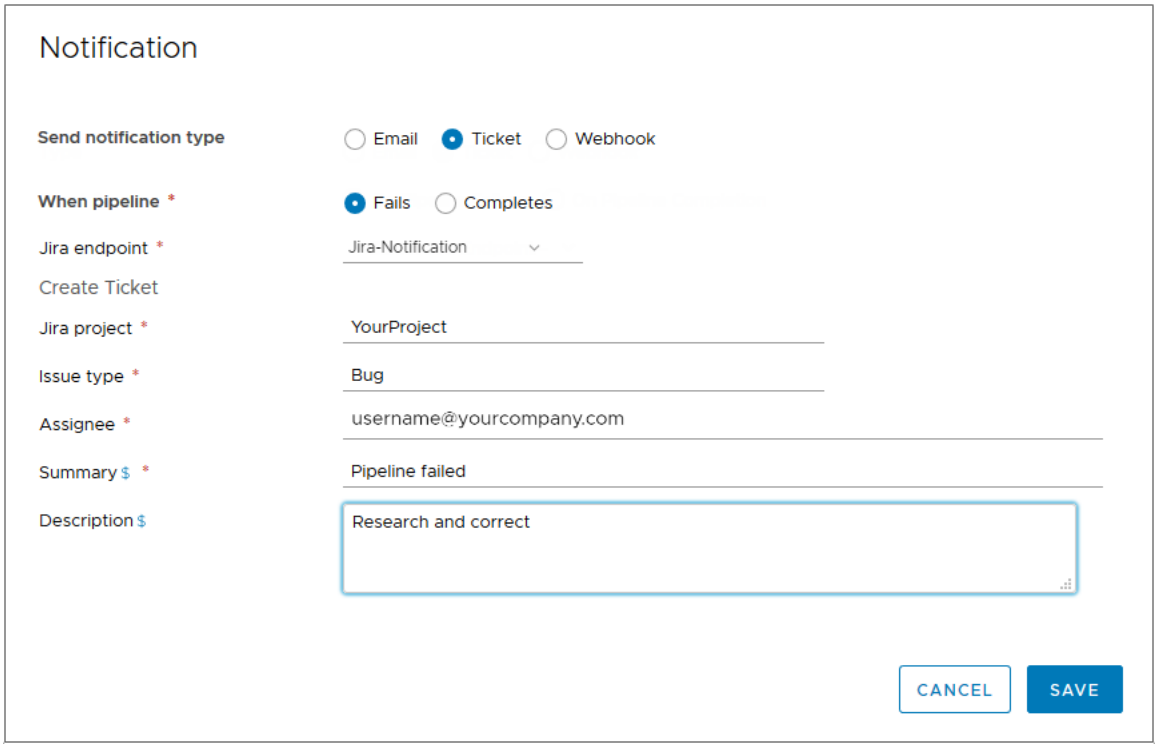
```
Codestream APP [12:01 AM]
Tested by User1 - Staging Pipeline 'User1-Pipeline', Pipeline ID
'e9b5884d809ce2755728177f70f8a' succeeded
```

6 要创建 Jira 票证，请配置票证信息。

a 选择**票证**。

b 要配置 Jira 通知，请输入 信息。

c 单击**保存**。



结果

恭喜！您已了解到，您可以在 vRealize Automation Code Stream 的管道的多个区域中创建各种类型的通知。

后续步骤

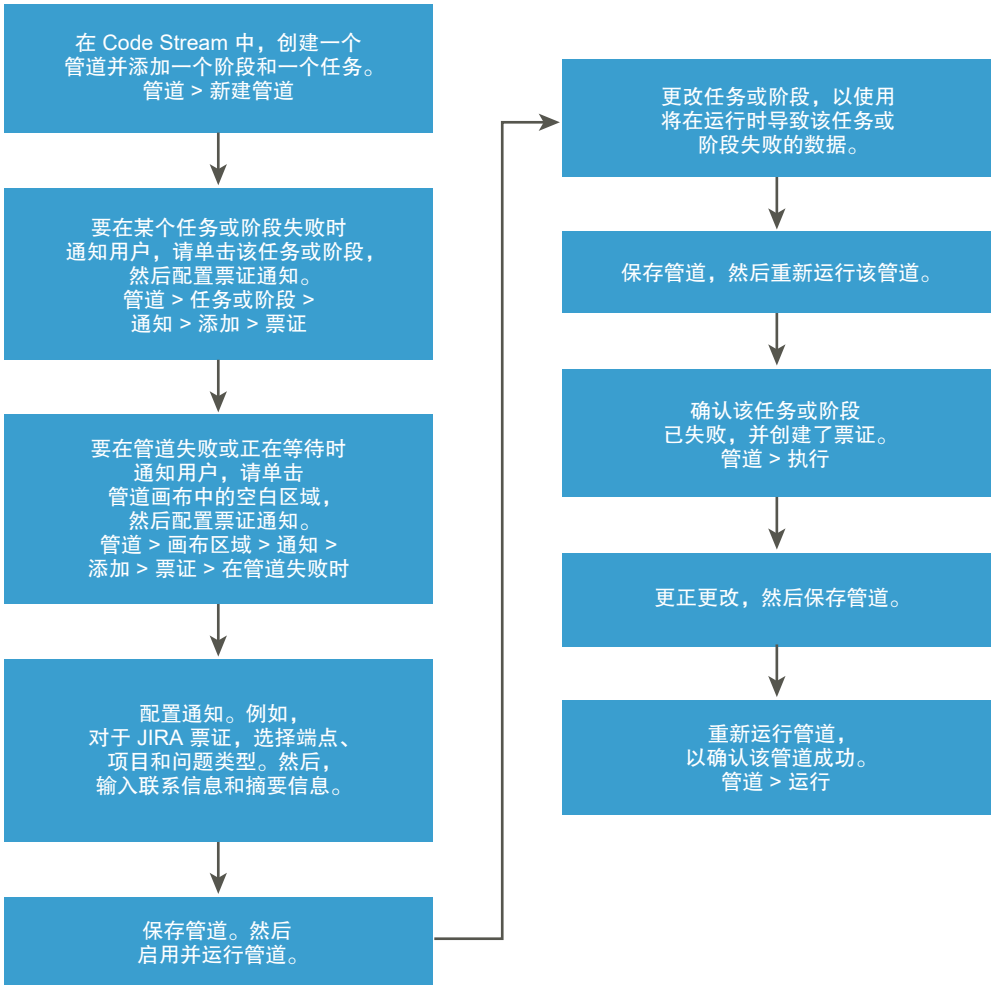
有关如何创建通知的详细示例，请参见管道任务失败时如何在 vRealize Automation Code Stream 中创建 Jira 票证。

管道任务失败时如何在 vRealize Automation Code Stream 中创建 Jira 票证

如果管道中的阶段或任务失败，则可以让 vRealize Automation Code Stream 创建 Jira 票证。您可以将票证分配给必须解决问题的人员。此外，还可以在管道正在等待或已成功时创建票证。

您可以在任务、阶段或管道上添加和配置通知。vRealize Automation Code Stream 将根据您在其上添加通知的任务、阶段或管道的状态创建票证。例如，如果某个端点不可用，则您可以让 vRealize Automation Code Stream 为因无法连接到该端点而失败的任务创建 Jira 票证。

还可以在管道成功时创建通知。例如，您可以通知 QA 团队管道已成功完成，以便该团队可以确认构建并运行不同的测试管道。或者，您可以通知性能团队，以便他们可以衡量管道的性能，并准备试运行环境或生产环境的更新。



此示例创建管道任务失败时的 Jira 票证。

前提条件

- 确认您具有有效的 Jira 帐户并且可以登录到 Jira 实例。
- 确认 Jira 端点存在且正常运行。

步骤

1 在管道中，创建一个任务。

2 在任务配置区域中，单击**通知**。

3 单击**添加**，然后配置票证信息。

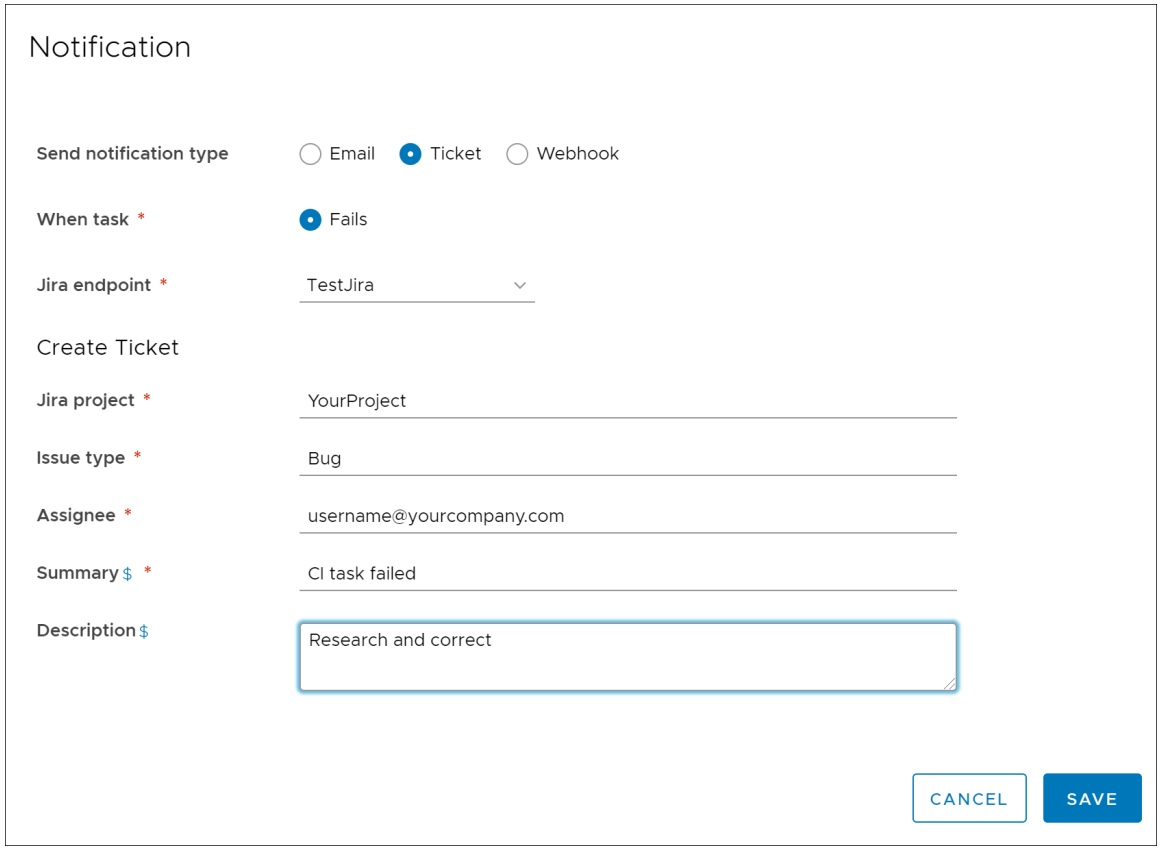
   a 单击**票证**。

   b 选择 Jira 端点。

c 输入 Jira 项目和问题类型。

d 输入将接收票证的人员的电子邮件地址。

e 输入票证的摘要和说明，然后单击**保存**。



4 保存管道，然后启用并运行管道

5 测试票证。

a 更改任务信息以包含导致任务失败的数据。

b 保存管道，然后重新运行该管道。

c 单击**执行**，并确认管道已失败。

d 在执行中，确认 vRealize Automation Code Stream 已创建并发送票证。

e 恢复任务信息以将其更正，然后重新运行管道并确保其成功。

结果

恭喜！您已让 vRealize Automation Code Stream 在管道任务失败时创建 Jira 票证，并将其分配给需要解决该问题的人员。

后续步骤

继续添加通知，以向您的团队发送有关管道的警示。

如何在 vRealize Automation Code Stream 中回滚部署

可以将回滚配置为一个管道，包含部署管道出现故障后将部署恢复到之前稳定状态的任务。要在发生故障时进行回滚，请将回滚管道连接到任务或阶段。

根据您的角色，回滚的原因可能有所不同。

- 作为发布工程师，我希望 vRealize Automation Code Stream 在发布期间验证是否成功，以便确定应该继续发布还是应该回滚。可能的故障包括任务失败、UserOps 拒绝、超出衡量指标阈值。
- 作为环境所有者，我希望重新部署以前的发布，以便能够快速将环境恢复到已知的良好状态。
- 作为环境所有者，我希望支持回滚蓝绿部署，以便可以最大程度地缩短因发布失败引起的停机时间。

使用智能管道模板创建 CD 管道并选中回滚选项时，会自动将回滚添加到管道中的任务。在此用例中，将使用智能管道模板为使用滚动升级部署模型部署到 Kubernetes 集群的应用程序定义回滚。智能管道模板会创建一个部署管道以及一个或多个回滚管道。

- 在部署管道中，如果“更新部署”或“验证部署”任务失败，则需要回滚。
- 在回滚管道中，使用旧映像更新部署。

您也可以使用空白模板手动创建回滚管道。创建回滚管道之前，您需要计划回滚流。有关回滚的更多背景信息，请参见在 vRealize Automation Code Stream 中计划回滚。

前提条件

- 验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是其成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。
- 设置 Kubernetes 集群，以便管道在其中部署应用程序。设置一个开发集群和一个生产集群。
- 确认您有 Docker 注册表安装程序。
- 标识将对所有工作（包括管道、端点和仪表板）进行分组的项目。
- 熟悉 CD 智能模板，如在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建中的 CD 部分中所述，例如：
  - 创建 Kubernetes 开发端点和生产端点，以便将应用程序映像部署到 Kubernetes 集群。
  - 准备创建命名空间、服务和部署的 Kubernetes YAML 文件。如果需要从专有存储库下载映像，则 YAML 文件必须包含具有 Docker 配置密钥的部分。

步骤

1. 单击**管道 > 新建管道 > 智能模板 > 持续交付**。
2. 在智能管道模板中输入信息。
   a. 选择一个项目。
   b. 输入管道名称，例如 **`RollingUpgrade-Example`**。
   c. 选择用于应用程序的环境。要将回滚添加到部署，必须选择**生产**。

d 单击**选择**，选择一个 Kubernetes YAML 文件，然后单击**处理**。

智能管道模板将显示可用的服务和部署环境。

e 选择管道将用于部署的服务。

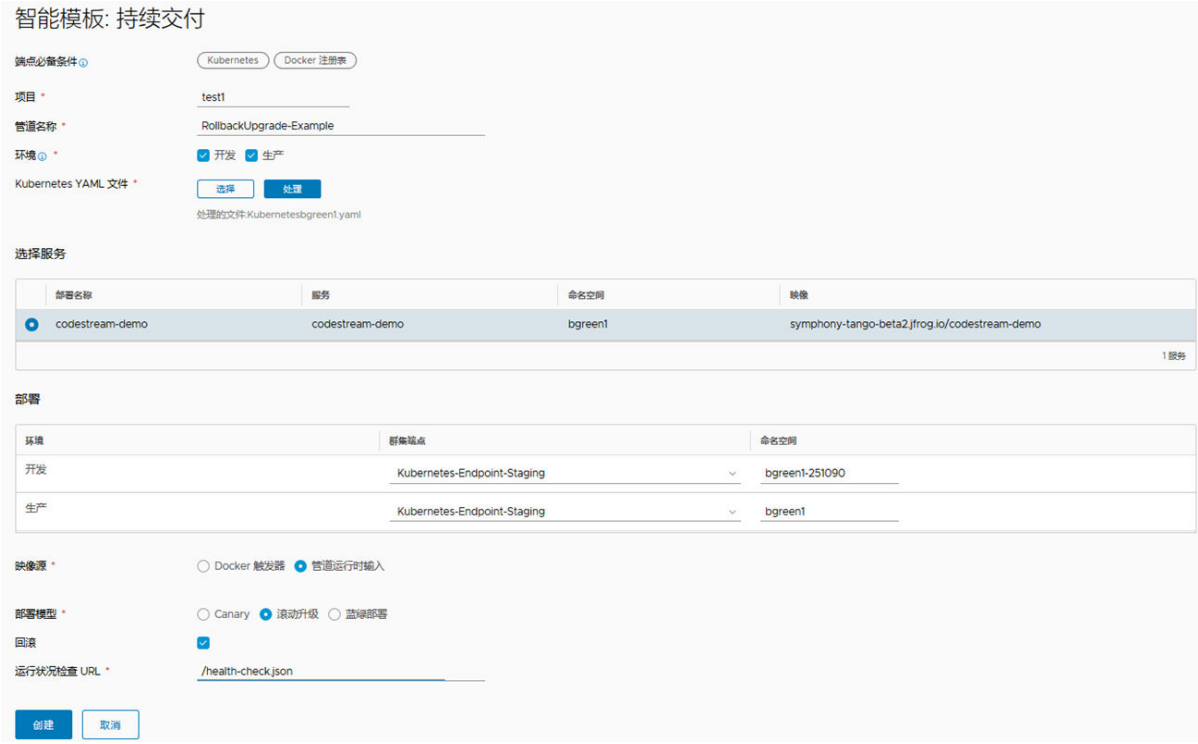
f 选择“开发”环境和“生产”环境对应的集群端点。

g 对于“映像源”，选择**管道运行时输入**。

h 对于“部署模型”，选择**滚动升级**。

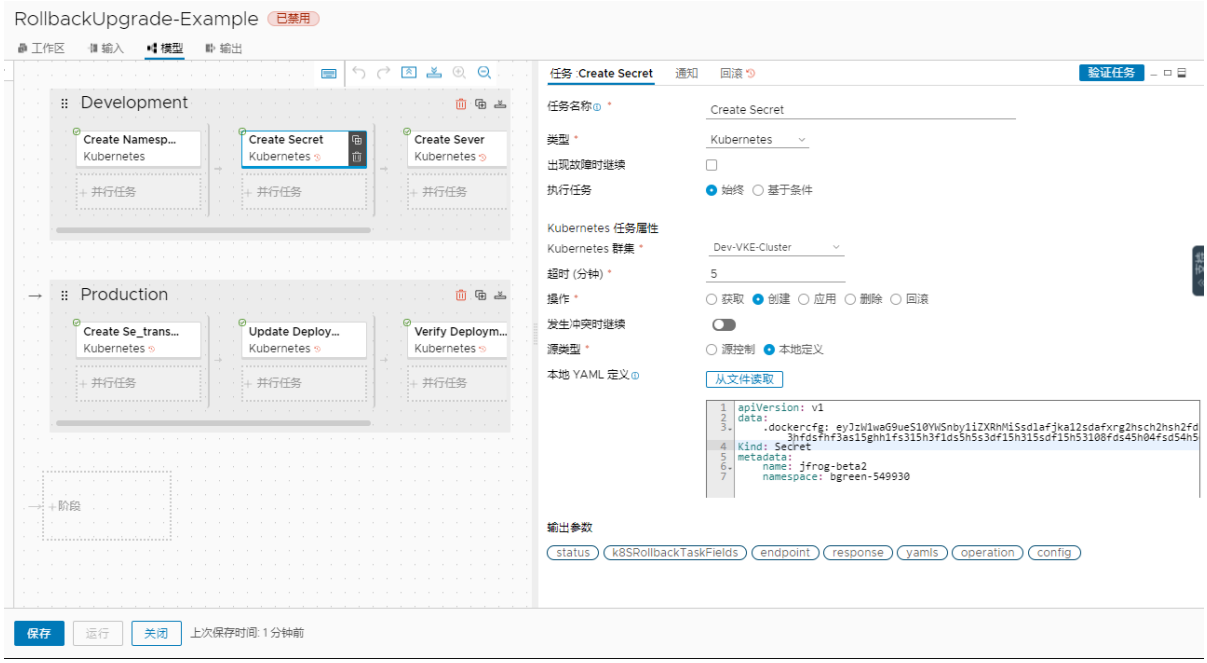
i 单击**回滚**。

j 提供**运行状况检查 URL**。



3 要创建名为 RollbackUpgrade-Example 的管道，请单击**创建**。

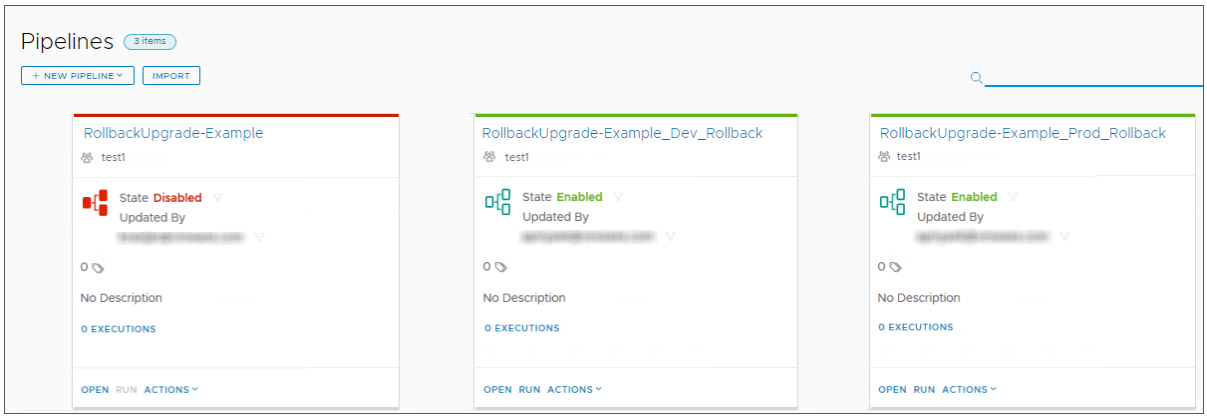
将显示名为 RollbackUpgrade-Example 的管道，并且在可以在开发阶段和生产阶段回滚的任务上显示回滚图标。



4 关闭管道。

在“管道”页面中，将显示创建的管道以及该管道中每个阶段的新管道。

- RollingUpgrade-Example。 vRealize Automation Code Stream 会停用默认创建的管道，确保在运行之前先进行审阅。
- RollingUpgrade-Example_Dev_Rollback。开发阶段中的任务（例如，**创建服务**、**创建密钥**、**创建部署**和**验证部署**）失败时，将调用此回滚开发管道。为确保开发任务回滚， vRealize Automation Code Stream 默认启用回滚开发管道。
- RollingUpgrade-Example_Prod_Rollback。生产阶段中的任务（例如，**部署阶段 1**、**验证阶段 1**、**部署发布阶段**、**完成发布阶段**和**验证发布阶段**）失败时，将调用此回滚生产管道。为确保生产任务回滚， vRealize Automation Code Stream 默认启用回滚生产管道。

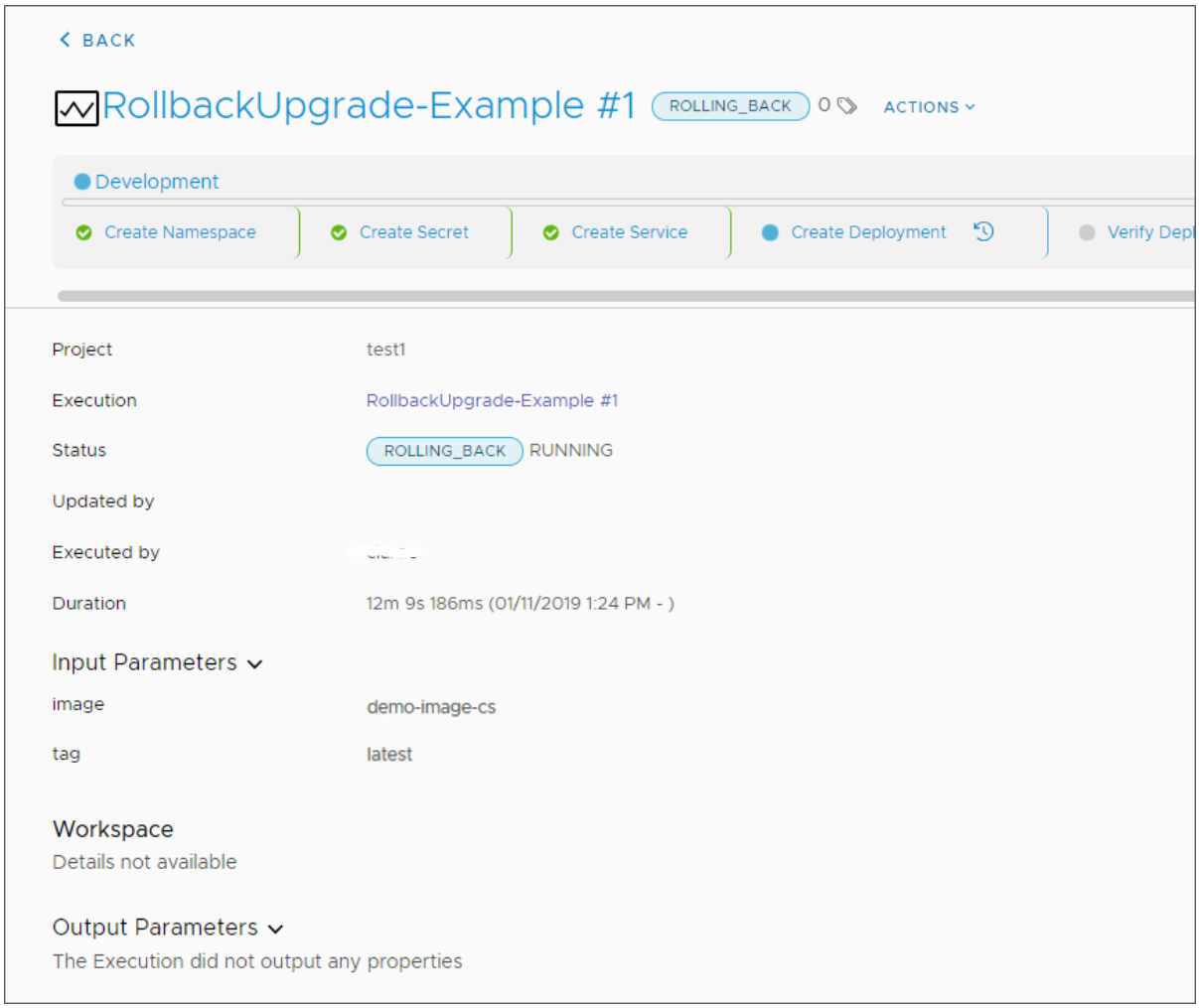


5 启用并运行已创建的管道。

开始运行时， vRealize Automation Code Stream 会提示您输入参数。您需要为所使用的 Docker 存储库中的端点提供映像和标记。

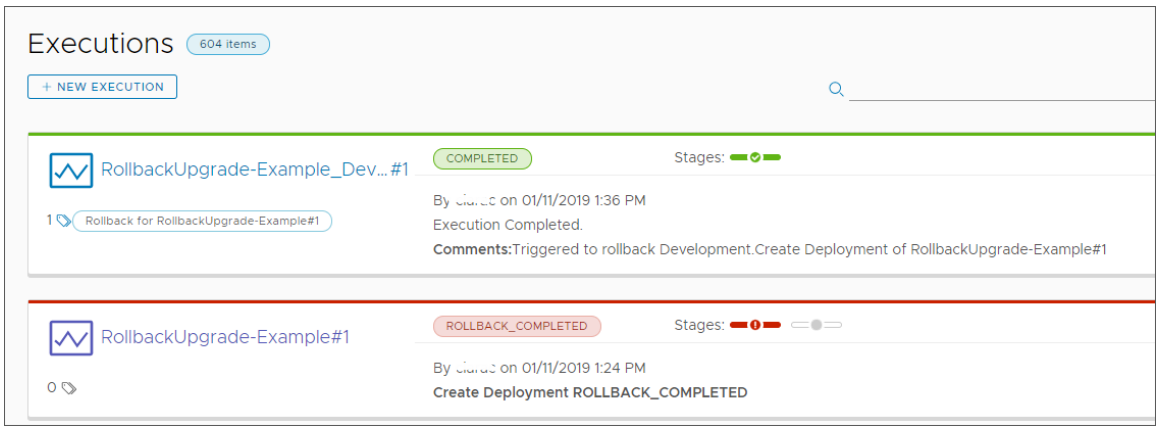
6 在“执行”页面中，选择**操作 > 查看执行**并监视管道执行。

管道将从 **RUNNING** 状态开始，并经历开发阶段中的各个任务。如果管道无法运行开发阶段中的某个任务，则名为 RollingUpgrade-Example_Dev_Rollback 的管道会触发并回滚部署，且管道状态会更改为 **ROLLING_BACK**。



回滚之后，“执行”页面将列出 RollingUpgrade-Example 管道的两个执行。

- 您创建的管道已回滚并显示 **ROLLBACK_COMPLETED**。
- 触发并执行了回滚的回滚开发管道将显示 **COMPLETED**。



结果

恭喜！您已成功定义包含回滚的管道并看到 vRealize Automation Code Stream 会在出现故障时回滚管道。

在 vRealize Automation Code Stream 中规划本地构建、集成和交付代码

4

在使用为您创建 CICD、CI 或 CD 管道的本地功能让 vRealize Automation Code Stream 构建、集成和交付代码之前，请规划本机构建。然后，可以使用其中一个智能管道模板创建管道，也可以通过手动添加阶段和任务进行创建。

为了规划持续集成和持续交付构建，我们提供了几个说明如何操作的示例。这些计划介绍了必备条件并概述了相关信息，有助于在构建管道时有效地准备和使用本地构建功能。

本章讨论了以下主题：

- 配置管道工作区
- 在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建
- 在使用智能管道模板之前在 vRealize Automation Code Stream 中计划持续集成本机构建
- 在使用智能管道模板之前在 vRealize Automation Code Stream 中计划持续交付本机构建
- 在手动添加任务之前在 vRealize Automation Code Stream 中规划 CICD 本地构建
- 在 vRealize Automation Code Stream 中计划回滚

配置管道工作区

要运行持续集成任务和自定义任务，必须为 vRealize Automation Code Stream 管道配置工作区。工作区支持 Docker 和 Kubernetes 端点。配置管道工作区时，必须包含构建器映像。

在管道工作区中，选择 Docker 或 Kubernetes，并包括 Docker 主机端点或 Kubernetes 主机端点。Docker 和 Kubernetes 平台管理 vRealize Automation Code Stream 为运行持续集成 (CI) 任务或自定义任务而部署的容器的整个生命周期。

- Docker 工作区需要 Docker 主机端点、构建器映像 URL、映像注册表、工作目录、缓存、环境变量、CPU 限制和内存限制。您还可以创建 Git 存储库的克隆。
- Kubernetes 工作区需要 Kubernetes 主机端点、构建器映像 URL、映像注册表、命名空间、NodePort、持久卷声明 (PVC)、工作目录、环境变量、CPU 限制和内存限制。您还可以创建 Git 存储库的克隆。

管道工作区配置包含许多通用参数以及特定于工作区类型的其他参数，如下表所述。

表 4-1. 工作区区域、详细信息和可用性

选择	说明	详细信息和可用性
类型	工作区的类型。	可用于 Docker 或 Kubernetes。
主机端点	运行持续集成 (CI) 任务和自定义任务的主机端点。	选择 Docker 主机端点时，可用于 Docker 工作区。 选择 Kubernetes 主机端点时，可用于 Kubernetes 工作区。
生成器映像 URL	生成器映像的名称和位置。使用此映像在 Docker 主机和 Kubernetes 集群上创建容器。持续集成 (CI) 任务和自定义任务在此容器内运行。	示例：**`fedora:latest`** 生成器映像必须具有 `curl` 或 `wget`。
映像注册表	如果生成器映像在注册表中可用，并且注册表需要凭据，则必须先创建映像注册表端点，然后在此处选择该端点，以便可以从注册表中提取映像。	可用于 Docker 和 Kubernetes 工作区。
工作目录	工作目录是容器内运行持续集成 (CI) 任务步骤的位置，也是 Git Webhook 触发管道运行时克隆代码的位置。	可用于 Docker 或 Kubernetes。
命名空间	如果未输入命名空间， vRealize Automation Code Stream 将在您提供的 Kubernetes 集群中创建唯一名称。	特定于 Kubernetes 工作区。
NodePort	vRealize Automation Code Stream 使用 NodePort 与 Kubernetes 集群中运行的容器进行通信。 如果未选择端口，则 vRealize Automation Code Stream 将使用 Kubernetes 分配的临时端口。您必须确保防火墙规则配置为允许输入临时端口范围 (30000-32767)。 如果输入端口，则必须确保集群中的其他服务尚未使用该端口，并且在防火墙规则中允许该端口。	特定于 Kubernetes 工作区。
持久卷声明	为 Kubernetes 工作区提供一种在各个管道运行过程中持久保留文件的方法。提供持久卷声明名称时，它可以存储日志、项目和缓存。 有关创建持久卷声明的详细信息，请参见 Kubernetes 文档，网址为 `https://kubernetes.io/docs/concepts/storage/persistent-volumes/`。	特定于 Kubernetes 工作区。
环境变量	管道运行时，此处传递的键-值对将可用于该管道中的所有持续集成 (CI) 任务和自定义任务。	可用于 Docker 或 Kubernetes。 可以在此处传递对变量的引用。 在工作区中提供的环境变量将传递到管道中的所有持续集成 (CI) 任务和自定义任务。 如果未在此处传递环境变量，则必须将这些变量明确传递到管道中的每个持续集成 (CI) 任务和自定义任务。
CPU 限制	持续集成 (CI) 容器或自定义任务容器的 CPU 资源限制。	默认值为 **1**。
内存限制	持续集成 (CI) 容器或自定义任务容器的内存限制。	单位为 `MB`。

表 4-1. 工作区区域、详细信息和可用性 （续）

选择	说明	详细信息和可用性
Git 克隆	如果选择 **Git 克隆**，并且 Git Webhook 调用管道，则会将代码克隆到工作区（容器）。	如果未启用 **Git 克隆**，则必须在管道中配置额外的显式持续集成 (CI) 任务以便先克隆代码，然后再执行其他步骤，如构建和测试。

缓存	在 vRealize Automation Code Stream 工作区中，可以缓存一组目录或文件，以便加快后续管道运行。这些目录的示例包括 `.m2` 和 `npm_modules`。如果不需要在管道运行之间缓存数据，则无需提供持久卷声明。
	容器中的工件（如文件或目录）将缓存以在各个管道运行过程中重用。例如，可以缓存 `node_modules` 或 `.m2` 文件夹。**缓存**接受路径列表。
	例如：
	`workspace:`
	`  type: K8S`
	`  endpoint: K8S-Micro`
	`  image: fedora:latest`
	`  registry: Docker Registry`
	`  path: ''`
	`  cache:`
	`  - /path/to/m2`
	`  - /path/to/node_modules`
		特定于工作区类型。
		在 Docker 工作区中，通过使用 Docker 主机中的共享路径持久保留缓存的数据、工件和日志来实现**缓存**。
		在 Kubernetes 工作区中，只有在提供持久卷声明时才能使用**缓存**。如果不提供持久卷声明，则不会启用**缓存**。

在管道工作区中使用 Kubernetes 端点时， vRealize Automation Code Stream 会创建必要的 Kubernetes 资源（如 ConfigMap、密钥和 Pod）以运行持续集成 (CI) 任务或自定义任务。 vRealize Automation Code Stream 使用 NodePort 与容器进行通信。

要在各个管道运行过程中共享数据，您必须提供持久卷声明， vRealize Automation Code Stream 会将此持久卷声明挂载到容器以存储数据，并将其用于后续管道运行。

在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建

要在 vRealize Automation Code Stream 中创建持续集成和持续交付 (CICD) 管道，可以使用 CICD 智能管道模板。要计划 CICD 本机构建，您需要在此示例计划中创建管道之前收集智能管道模板所需的信息。

要创建 CICD 管道，您必须对管道的持续集成 (CI) 阶段和持续交付 (CD) 阶段进行计划。

在智能管道模板中输入信息并将其保存后，该模板会创建一个包含阶段和任务的管道。它还会根据所选环境类型（例如，开发和生产）指示映像的部署目标。管道将发布容器映像，并执行运行该映像所需的操作。运行管道后，您可以在整个管道执行过程中监控趋势。

当管道包含 Docker Hub 中的映像时，在运行管道之前必须确保映像中已嵌入了 `cURL` 或 `wget`。当管道运行时， vRealize Automation Code Stream 会下载使用 `cURL` 或 `wget` 运行命令的二进制文件。

有关配置工作区的信息，请参见配置管道工作区。

计划持续集成 (CI) 阶段

要计划管道的 CI 阶段，您需要设置外部要求和内部要求，并确定智能管道模板的 CI 部分所需的信息。以下是摘要。

此示例使用 Docker 工作区。

您需要的端点和存储库：

- Git 源代码存储库，开发人员将代码签入到该存储库中。开发人员提交更改时， vRealize Automation Code Stream 会将最新代码提取到管道中。
- 存储库的 Git 端点，开发人员源代码驻留在该端点。
- Docker 生成主机的 Docker 端点，该端点将在容器内运行生成命令。
- Kubernetes 端点，以便 vRealize Automation Code Stream 可以将映像部署到 Kubernetes 集群。
- 生成器映像，该映像将创建持续集成测试在其中运行的容器。
- 映像注册表端点，以便 Docker 生成主机可以从该端点提取生成器映像。

您需要有权访问项目。项目会对所有工作（包括管道、端点和仪表板）进行分组。验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是其成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。

您将需要一个 Git Webhook，它允许 vRealize Automation Code Stream 在开发人员提交代码更改时使用 Git 触发器来触发您的管道。请参见如何使用 vRealize Automation Code Stream 中的 Git 触发器运行管道。

生成工具集：

- 生成类型，例如 Maven。
- 您使用的所有后处理生成工具，例如 JUnit、JaCoCo、Checkstyle 和 FindBugs。

发布工具：

- 诸如 Docker 等工具，该工具将部署生成容器。
- 映像标记，该标记是提交 ID 或内部版本号。

生成工作区：

- Docker 生成主机，该主机是 Docker 端点。
- 映像注册表。管道的 CI 部分将从所选注册表端点提取映像。容器将运行 CI 任务，并部署映像。如果注册表需要凭据，则必须创建映像注册表端点，然后在此处选择该端点，以便主机可以从注册表提取映像。
- 生成器映像的 URL，该映像将创建持续集成任务在其中运行的容器。

计划持续交付 (CD) 阶段

要计划管道的 CD 阶段，您需要设置外部要求和内部要求，并确定需要在智能管道模板的 CD 部分中输入的信息。

您需要的端点：

- Kubernetes 端点，以便 vRealize Automation Code Stream 可以将映像部署到 Kubernetes 集群。

环境类型和文件：

- vRealize Automation Code Stream 将应用程序部署到的所有环境类型，例如，开发和生产。智能管道模板将根据所选环境类型在管道中创建各个阶段和任务。

表 4-2. CICD 智能管道模板创建的管道阶段

管道内容	作用
生成-发布阶段	生成并测试代码，创建生成器映像，以及将映像发布到 Docker 主机。
开发阶段	使用 Amazon Web Services (AWS) 开发集群创建并部署映像。在此阶段，您可以在集群中创建命名空间，以及创建密钥。
生产阶段	使用 VMware Tanzu Kubernetes Grid Integrated Edition（以前称为 VMware Enterprise PKS）的生产版本将映像部署到生产 Kubernetes 集群。

- 在 CICD 智能管道模板的 CD 部分中选择的 Kubernetes YAML 文件。

  Kubernetes YAML 文件包括用于命名空间、服务和部署的三个必需部分以及一个用于密钥的可选部分。如果计划通过从专有存储库下载映像来创建管道，则必须包含具有 Docker 配置密钥的部分。 如果创建的管道仅使用公开发布的映像，则不需要密钥。以下示例 YAML 文件包含四个部分。

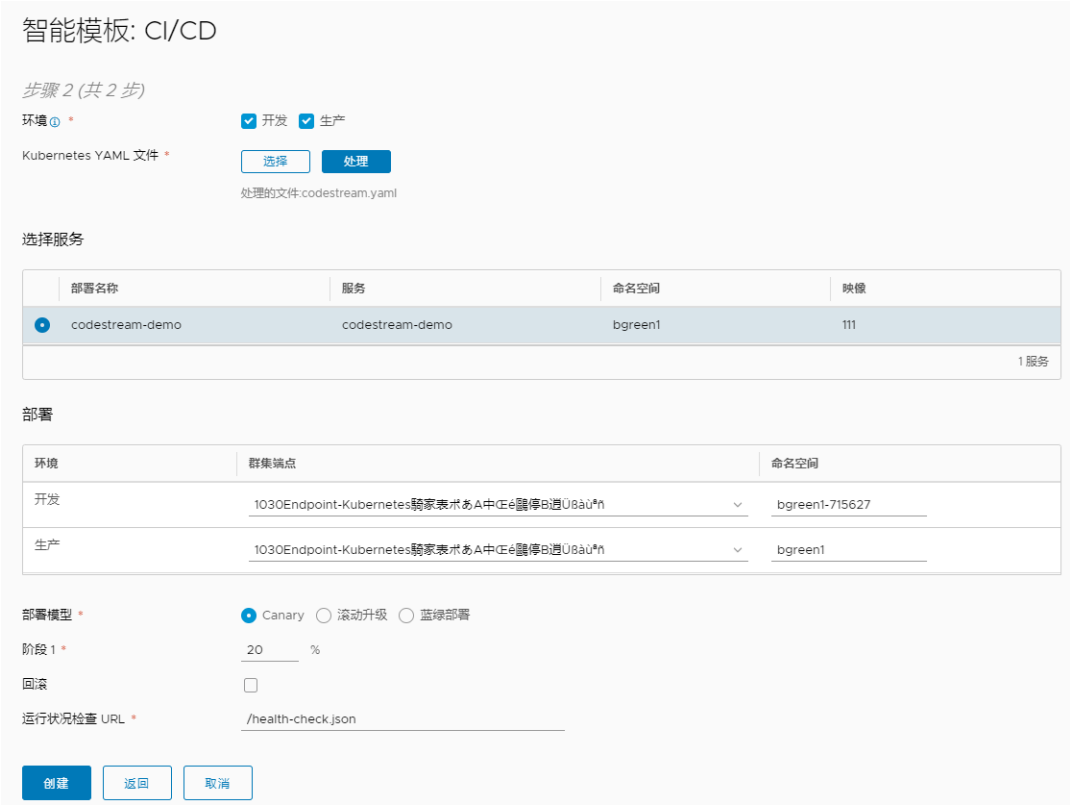
```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream
  namespace: codestream
---
apiVersion: v1
data:
  .dockerconfigjson:
eyJhdXRocyI6eyJodHRwczovL2luZ1234567890lci5pby92MS8iOnsidXNlcm5hbWUiOiJhdXRvbWF0aW9uYmV0YSI
sInBhc3N3b3JkIjoiVk13YXJlQDEyMyIsImVtYWlsIjoiYXV0b21hdGlvbmJldGF1c2VyQGdtYWlsLmNvbSIsImF1dG
giOiJZWFYwYjIxaGRHbHZibUpsZEdFNlZrMTNZWEpsUURFeU13PT0ifX19
kind: Secret
metadata:
  name: dockerhub-secret
  namespace: codestream
type: kubernetes.io/dockerconfigjson
---
apiVersion: v1
kind: Service
metadata:
  name: codestream-demo
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  ports:
    - port: 80
  selector:
```

```
    app: codestream-demo
    tier: frontend
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: codestream-demo
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  replicas: 10
  selector:
    matchLabels:
      app: codestream-demo
      tier: frontend
  template:
    metadata:
      labels:
        app: codestream-demo
        tier: frontend
    spec:
      containers:
      - name: codestream-demo
        image: automationbeta/codestream-demo:01
        ports:
        - containerPort: 80
          name: codestream-demo
      imagePullSecrets:
      - name: dockerhub-secret
```

注 Kubernetes YAML 文件也用于 CD 智能管道模板，例如，以下用例示例：

- 如何将 vRealize Automation Code Stream 中的应用程序部署到蓝绿部署
- 如何在 vRealize Automation Code Stream 中回滚部署
- 如何使用 vRealize Automation Code Stream 中的 Docker 触发器运行持续交付管道

要在智能模板中应用该文件，请单击**选择**，并选择 Kubernetes YAML 文件。然后单击**处理**。智能管道模板将显示可用的服务和部署环境。需要选择服务、集群端点和部署策略。例如，要使用 Canary 部署模型，请选择 **Canary** 并输入部署阶段所占的百分比。

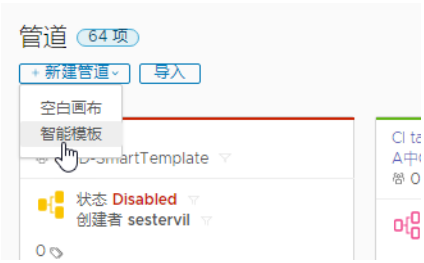


有关使用智能管道模板创建适用于蓝绿部署的管道的示例，请参见如何将 vRealize Automation Code Stream 中的应用程序部署到蓝绿部署。

如何使用智能管道模板创建 CICD 管道

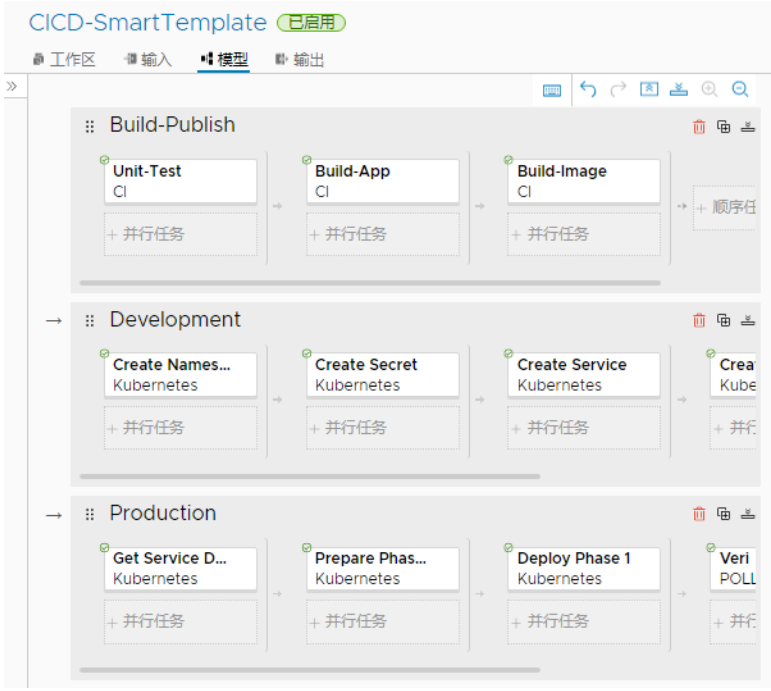
收集所有信息并设置所需的内容后，下文介绍了如何从 CICD 智能管道模板创建管道。

在“管道”中，您需要选择**新建管道 > 智能模板**。



选择 CICD 智能管道模板。

填写模板，并使用模板创建的阶段保存管道。如果需要进行任何最终更改，可以编辑并保存管道。



然后启用并运行管道。管道运行后，可以进行以下检查：

- 确认管道已成功完成。单击**执行**，然后搜索管道。如果失败，请更正任何错误并再次运行。
- 确认 Git Webhook 运行正常。Git **活动**选项卡将显示事件。单击**触发器 > Git > 活动**。
- 查看管道仪表板并检查趋势。单击**仪表板**，然后搜索您的管道仪表板。您还可以创建自定义仪表板以报告其他 KPI。

有关详细示例，请参见如何持续将来自 GitHub 或 GitLab 存储库的代码集成到 vRealize Automation Code Stream 中的管道。

在使用智能管道模板之前在 vRealize Automation Code Stream 中计划持续集成本机构建

要在 VMware Code Stream 中创建持续集成 (CI) 管道，可以使用持续集成智能管道模板。要计划持续集成本机构建，您需要在此示例计划中创建管道之前收集智能管道模板所需的信息。

当您填写智能管道模板时，会在存储库中创建持续集成管道，并执行操作以便管道可以运行。运行管道后，您可以在整个管道执行过程中监控趋势。

要在使用持续集成智能模板之前规划构建，您需要：

- 标识将对所有工作（包括管道、端点和仪表板）进行分组的项目。
- 按照在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建的持续交付部分中所述收集构建信息。

  例如，添加一个 Kubernetes 端点，其中 vRealize Automation Code Stream 将部署容器。

然后，使用持续集成智能管道模板创建管道。

在“管道”中，选择**智能模板**。

选择持续集成智能管道模板。

要保存管道及其创建的阶段，请填写模板，然后输入管道的名称。要保存管道及其创建的阶段，请单击**创建**。

vRealize Automation Code Stream 管道工作区支持使用 Docker 和 Kubernetes 执行持续集成任务和自定义任务。

有关配置工作区的信息，请参见配置管道工作区。

要进行任何最终更改，您可以编辑管道。然后，您可以启用管道并运行它。管道运行后：

- 验证管道是否成功。单击**执行**，然后搜索管道。如果失败，请更正所有错误并再次运行。
- 验证 Git webhook 是否正常运行。Git **活动**选项卡显示事件。单击**触发器 > Git > 活动**。
- 查看管道仪表板并检查趋势。单击**仪表板**，然后搜索管道仪表板。要报告更多关键性能指标，您可以创建自定义仪表板。

有关详细示例，请参见如何持续将来自 GitHub 或 GitLab 存储库的代码集成到 vRealize Automation Code Stream 中的管道。

在使用智能管道模板之前在 vRealize Automation Code Stream 中计划持续交付本机构建

要在 vRealize Automation Code Stream 中创建持续交付 (CD) 管道，可以使用持续交付智能管道模板。要计划持续交付本机构建，您需要在此示例计划中创建管道之前收集智能管道模板所需的信息。

当您填写智能管道模板时，会在存储库中创建持续交付管道，并执行操作以便管道可以运行。运行管道后，您可以在整个管道执行过程中监控趋势。

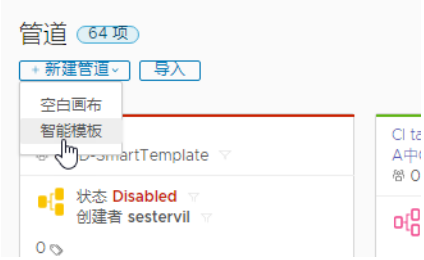
要在使用持续交付智能模板之前规划构建，您需要：

- 标识将对所有工作（包括管道、端点和仪表板）进行分组的项目。

- 按照在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建的持续交付部分中所述收集构建信息。例如：
  - 添加一个 Kubernetes 端点，其中 vRealize Automation Code Stream 将部署容器。
  - 准备创建命名空间、服务和部署的 Kubernetes YAML 文件。要从专有存储库下载映像，YAML 文件必须包含具有 Docker 配置密钥的部分。

然后，使用持续交付智能管道模板创建管道。

在“管道”中，选择**智能模板**。



选择持续交付智能管道模板。

填写模板，然后输入管道的名称。要保存管道及其创建的阶段，请单击**创建**。

vRealize Automation Code Stream 管道工作区支持使用 Docker 和 Kubernetes 执行持续集成任务和自定义任务。

有关配置工作区的信息，请参见配置管道工作区。

要进行任何最终更改，您可以编辑管道。然后，您可以启用管道并运行它。管道运行后：

- 确认管道已成功完成。单击**执行**，然后搜索管道。如果失败，请更正任何错误并再次运行。
- 确认 Git Webhook 运行正常。Git **活动**选项卡将显示事件。单击**触发器 > Git > 活动**。
- 查看管道仪表板并检查趋势。单击**仪表板**，然后搜索您的管道仪表板。要报告更多关键性能指标，您可以创建自定义仪表板。

有关详细示例，请参见如何持续将来自 GitHub 或 GitLab 存储库的代码集成到 vRealize Automation Code Stream 中的管道。

在手动添加任务之前在 vRealize Automation Code Stream 中规划 CICD 本地构建

要在 vRealize Automation Code Stream 中创建持续集成和持续交付 (CICD) 管道，您可以手动添加阶段和任务。要规划 CICD 本地构建，您需要收集所需的信息，然后创建管道并手动向其添加阶段和任务。

您必须规划管道的持续集成 (CI) 阶段和持续交付 (CD) 阶段。创建管道并运行后，您可以在整个管道执行过程中监控趋势。

当管道包含 Docker Hub 中的映像时，在运行管道之前必须确保映像中已嵌入了 `cURL` 或 `wget`。当管道运行时， vRealize Automation Code Stream 会下载使用 `cURL` 或 `wget` 运行命令的二进制文件。

vRealize Automation Code Stream 管道工作区支持使用 Docker 和 Kubernetes 执行持续集成任务和自定义任务。

有关配置工作区的信息，请参见配置管道工作区。

规划外部和内部要求

要规划管道的 CI 和 CD 阶段，以下要求指明了在创建管道之前必须执行的操作。

此示例使用 Docker 工作区。

要基于此示例计划创建管道，您将使用 Docker 主机、Git 存储库、Maven 和多个后处理生成工具。

您需要的端点和存储库：

- Git 源代码存储库，开发人员将代码签入到该存储库中。开发人员提交更改时， vRealize Automation Code Stream 会将最新代码提取到管道中。
- Docker 生成主机的 Docker 端点，该端点将在容器内运行生成命令。
- 生成器映像，该映像将创建持续集成测试在其中运行的容器。
- 映像注册表端点，以便 Docker 生成主机可以从该端点提取生成器映像。

您需要有权访问项目。项目会对所有工作（包括管道、端点和仪表板）进行分组。验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是其成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。

您将需要一个 Git Webhook，它允许 vRealize Automation Code Stream 在开发人员提交代码更改时使用 Git 触发器来触发您的管道。请参见如何使用 vRealize Automation Code Stream 中的 Git 触发器运行管道。

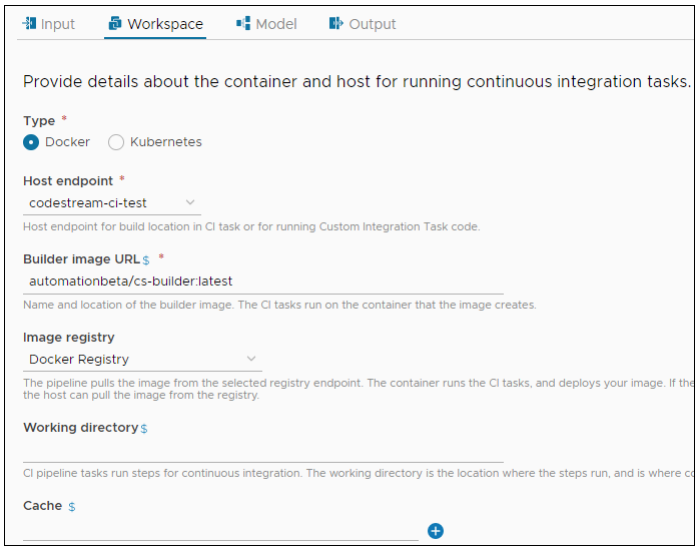
如何创建 CICD 管道和配置工作区

您需要创建管道，然后配置工作区、管道输入参数和任务。

要创建管道，您需要单击**管道 > 新建管道 > 空白画布**。

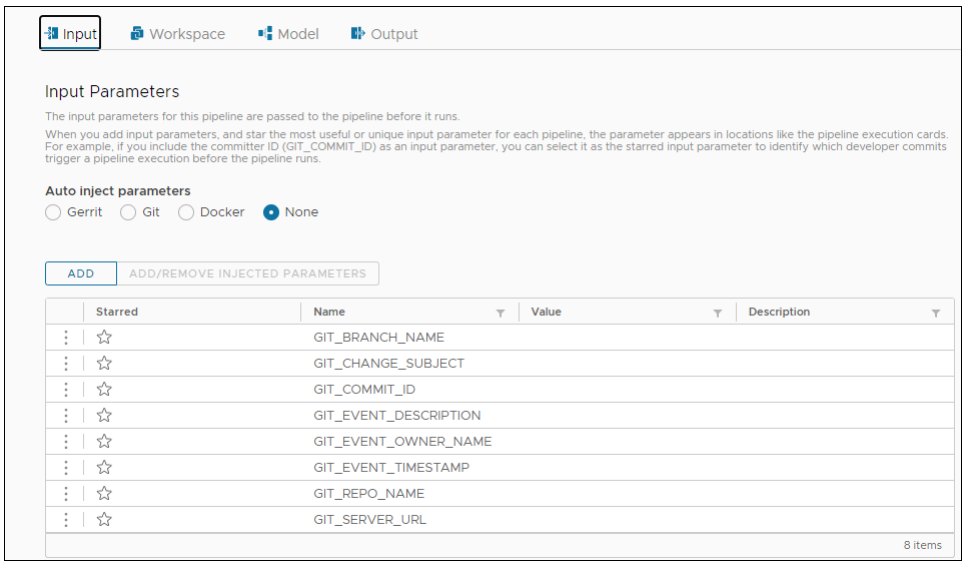
在“工作区”选项卡上，输入持续集成信息：

- 包括 Docker 生成主机。
- 输入您的生成器映像的 URL。
- 选择映像注册表端点，以便管道可以从中提取映像。容器将运行 CI 任务并部署映像。如果注册表需要凭据，您必须先创建映像注册表端点，然后在此时选择该端点，以便主机可以从注册表中提取映像。
- 添加必须缓存的工件。对于后续内部版本，目录等工件将作为依赖项下载。缓存是这些工件所驻留的位置。例如，依赖工件可以包括 Maven 的 `.m2` 目录，以及 Node.js 的 `node_modules` 目录。这些目录在各个管道执行过程中缓存，以便在生成期间节省时间。



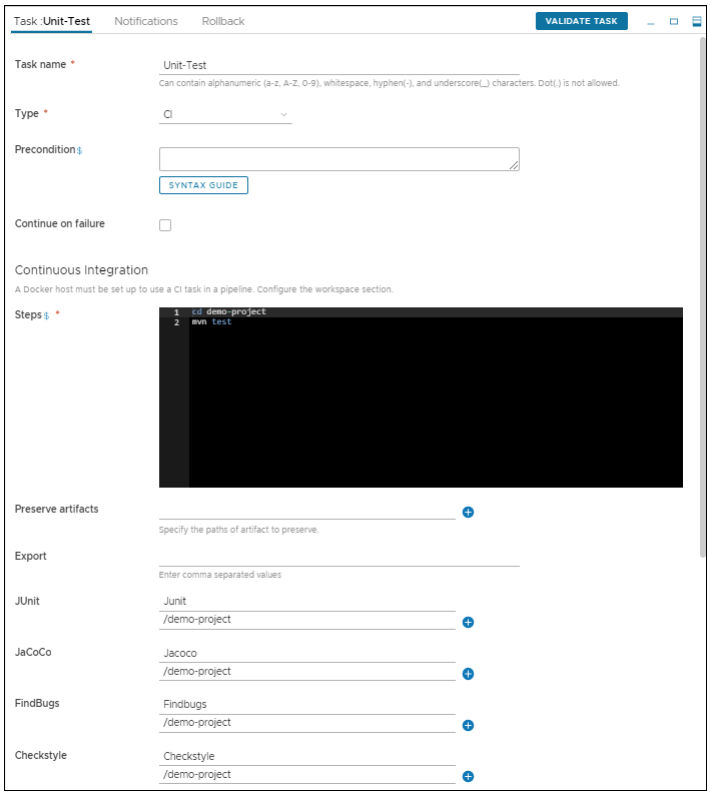
在“输入”选项卡上，配置管道输入参数。

- 如果管道使用来自 Git、Gerrit 或 Docker 触发器事件的输入参数，请为自动插入参数选择触发器类型。事件可以包括 Gerrit 或 Git 的更改主题，或者 Docker 的事件所有者名称。如果管道不使用从事件传递的任何输入参数，则将自动插入参数设置为**无**。
- 要对管道输入参数应用值和描述，请单击三个垂直圆点，然后单击**编辑**。所输入的值将用作任务、阶段或通知的输入。
- 要添加管道输入参数，请单击**添加**。例如，可以添加 `approvers` 以显示每次执行的默认值，但可以在运行时替代为不同的审批者。
- 要添加或移除插入的参数，请单击**添加/移除插入的参数**。例如，移除未使用的参数以减少结果页面的混乱，并仅显示所使用的输入参数。



配置管道以测试代码：

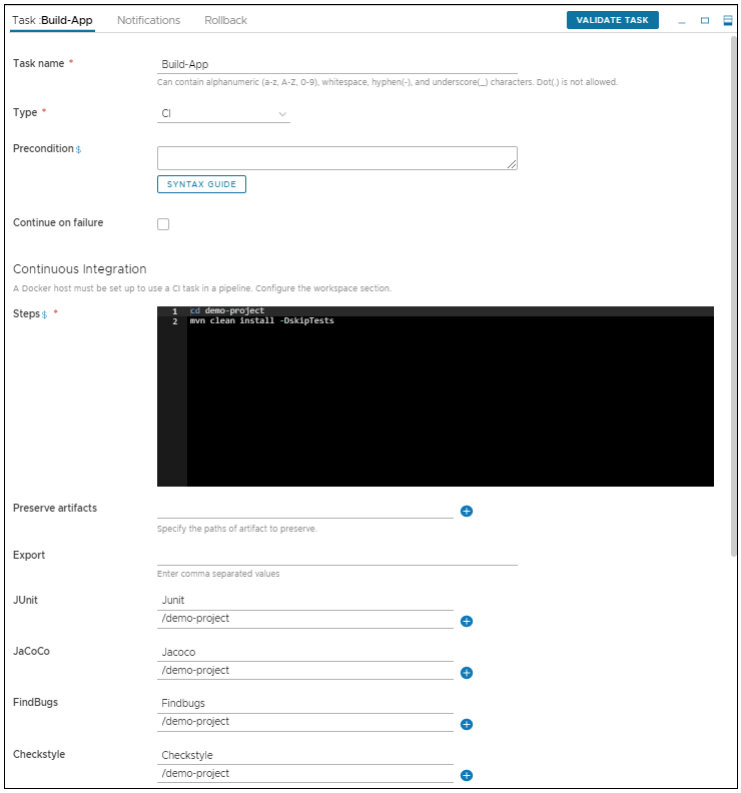
- 添加和配置 CI 任务。
- 其中包含使用代码来运行 `mvn test` 的步骤。
- 为识别任务运行后出现的任何问题，请运行后处理构建工具，例如 JUnit、JaCoCo、FindBugs 和 Checkstyle。



配置管道以生成代码：

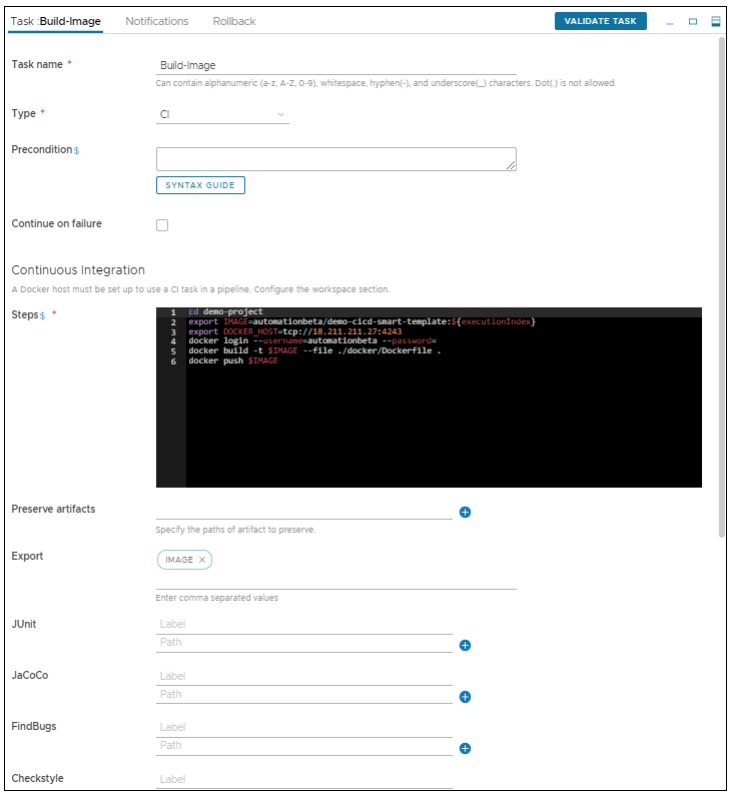
- 添加和配置 CI 任务。

- 包含使用代码运行 `mvn clean install` 的步骤。
- 包含位置和 JAR 文件名，并使其保留您的工件。



配置管道以将映像发布到 Docker 主机：

- 添加和配置 CI 任务。
- 添加提交、导出、生成和推送映像的步骤。
- 添加 `IMAGE` 的导出密钥以供下一个任务使用。



在配置工作区、输入参数、测试任务和生成任务后，保存您的管道。

如何启用和运行管道

为管道配置阶段和任务后，可以保存并启用管道。

然后，等待管道运行且完成，然后验证管道是否成功。如果失败，请更正任何错误并再次运行。

管道成功后，您可能想要确认以下事项：

- 检查管道执行情况并查看任务步骤的结果。
- 在管道执行的工作区中，找到有关容器和克隆 Git 存储库的详细信息。
- 在工作区中，查看后处理工具的结果，并检查错误、代码覆盖率、bug 和样式问题。
- 确认您的工件已保留。还要确认映像已使用 IMAGE 名称和值导出。
- 转到 Docker 存储库，并验证管道是否已发布您的容器。

有关显示 vRealize Automation Code Stream 如何持续集成代码的详细示例，请参见如何持续将来自 GitHub 或 GitLab 存储库的代码集成到 vRealize Automation Code Stream 中的管道。

在 vRealize Automation Code Stream 中计划回滚

如果管道执行失败，则可以使用回滚将环境恢复到先前稳定的状态。要使用回滚，请计划回滚流并了解如何实施。

回滚流规定了回滚失败部署所需的步骤。该流采用回滚管道的形式，该管道包括一个或多个连续任务，这些任务因执行和失败的部署类型而异。例如，传统应用程序的部署和回滚不同于 容器应用程序的部署和回滚。

要恢复到正常的部署状态，回滚管道通常包括以下任务：

- 清理状态或环境。
- 运行用户指定的脚本以恢复更改。
- 部署以前的部署修订版。

要将回滚添加到现有部署管道，请在运行部署管道之前，将回滚管道连接到部署管道中要回滚的任务或阶段。

如何配置回滚

要在部署中配置回滚，您需要：

- 创建部署管道。
- 确定部署管道中将触发回滚的潜在故障点，以便可以连接回滚管道。例如，可以将回滚管道连接到部署管道中的条件或轮询任务类型，以检查上一任务是否成功完成。有关条件任务的信息，请参见如何在条件任务中使用变量绑定来运行或停止 vRealize Automation Code Stream 中的管道。
- 确定将触发回滚管道的故障范围，例如任务或阶段失败。也可以将回滚连接到阶段。
- 确定出现故障时要执行的回滚任务。将创建包含这些任务的回滚管道。

可以手动创建回滚管道，也可以使用 vRealize Automation Code Stream 创建。

- 使用空白画布，可以手动创建一个遵循与现有部署管道并行流的回滚管道。然后，将回滚管道连接到部署管道中 出现故障时触发回滚的一个或多个任务。
- 通过智能管道模板，您可以使用回滚操作配置部署管道。然后，vRealize Automation Code Stream 会自动创建一个或多个具有预定义任务（用于在失败时回滚部署）的默认回滚管道。

有关如何使用智能管道模板为 CD 管道配置回滚的详细示例，请参见如何在 vRealize Automation Code Stream 中回滚部署。

如果我的部署管道有多个任务或阶段具有回滚，会怎样

如果有多个任务和阶段添加了回滚，请注意回滚顺序会有所不同。

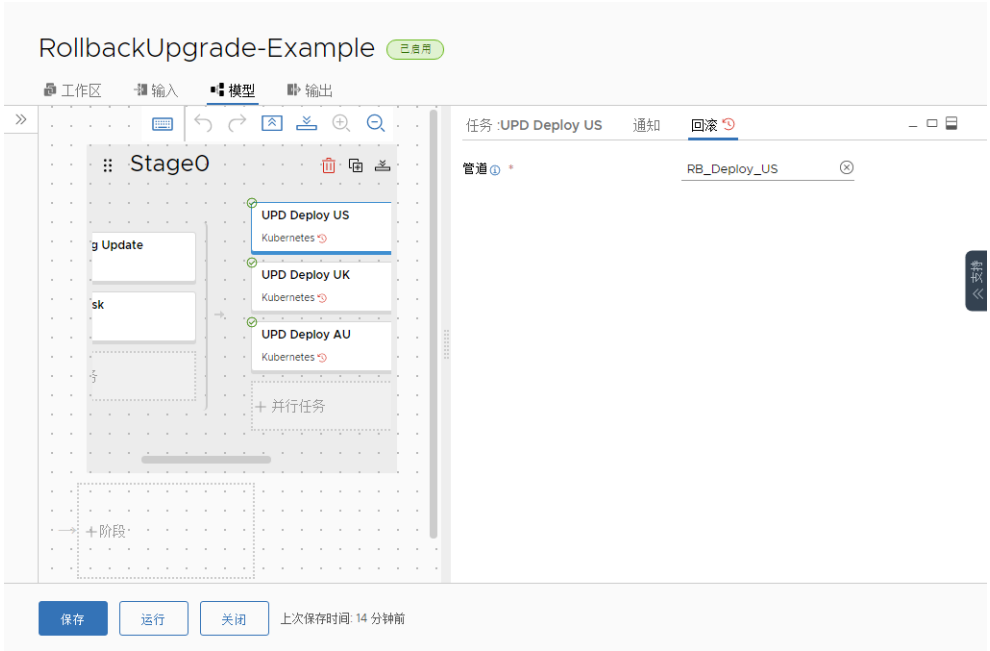
表 4-3. 确定回滚顺序

如果将回滚添加到...	回滚执行时间...
并行任务	如果其中一个并行任务失败，则所有并行任务完成或失败后回滚该任务。该任务失败后，不立即执行回滚。
阶段中的任务和阶段	如果任务失败，则运行任务回滚。如果任务属于一组并行任务，则所有并行任务完成或失败后回滚该任务。任务回滚完成或无法完成后，将运行阶段回滚。

考虑具有以下阶段或任务的管道：

- 具有回滚的生产阶段。
- 一组并行任务，且每个任务具有自己的回滚。

名为 **UPD Deploy US** 的任务具有回滚管道 **RB_Deploy_US**。如果 **UPD Deploy US** 失败，则回滚将遵循 **RB_Deploy_US** 管道中定义的流。



如果 **UPD Deploy US** 失败，则 **RB_Deploy_US** 管道在 **UPD Deploy UK** 和 **UPD Deploy AU** 都完成或失败后运行。**UPD Deploy US** 失败后，不立即执行回滚。而且，由于生产阶段也具有回滚，因此在 **RB_Deploy_US** 管道运行后，将运行阶段回滚管道。

vRealize Automation Code Stream 使用教程

5

vRealize Automation Code Stream 可对 DevOps 发布生命周期进行建模并提供支持，还会持续测试应用程序并将其发布到开发环境和生产环境。

您已设置所需的所有内容，以便可以使用 vRealize Automation Code Stream。请参见第 2 章 设置 vRealize Automation Code Stream 以对发布流程进行建模。

现在，您可以创建管道，以便在将开发人员代码发布到生产环境之前对其进行自动构建和测试。您可以让 vRealize Automation Code Stream 部署基于容器的或传统的应用程序。

表 5-1. 在 DevOps 生命周期中使用 vRealize Automation Code Stream

功能	可执行的操作示例
使用 vRealize Automation Code Stream 中的本地构建功能。	创建持续集成和持续交付 (CICD)、持续集成 (CI) 和持续交付 (CD) 管道，以持续集成、容器化和交付代码。 ■ 使用智能管道模板为您创建管道。 ■ 手动将阶段和任务添加到管道。
发布应用程序并自动发布。	以各种方式集成和发布应用程序。 ■ 持续将 GitHub 或 GitLab 存储库中的代码集成到管道。 ■ 集成 Docker 主机以运行持续集成任务，如此博客文章为 vRealize Automation Code Stream 创建 Docker 主机中所述。 ■ 使用 YAML 云模板自动部署应用程序。 ■ 自动将应用程序部署到 Kubernetes 集群。 ■ 将应用程序发布到蓝绿部署。 ■ 将 vRealize Automation Code Stream 与您自己的生成工具、测试工具和部署工具集成。 ■ 使用将 vRealize Automation Code Stream 与其他应用程序集成的 REST API。
跟踪趋势、衡量指标和关键绩效指标 (KPI)。	创建自定义仪表板并深入了解管道的性能。
解决问题。	管道运行失败时，让 vRealize Automation Code Stream 创建 Jira 票证。

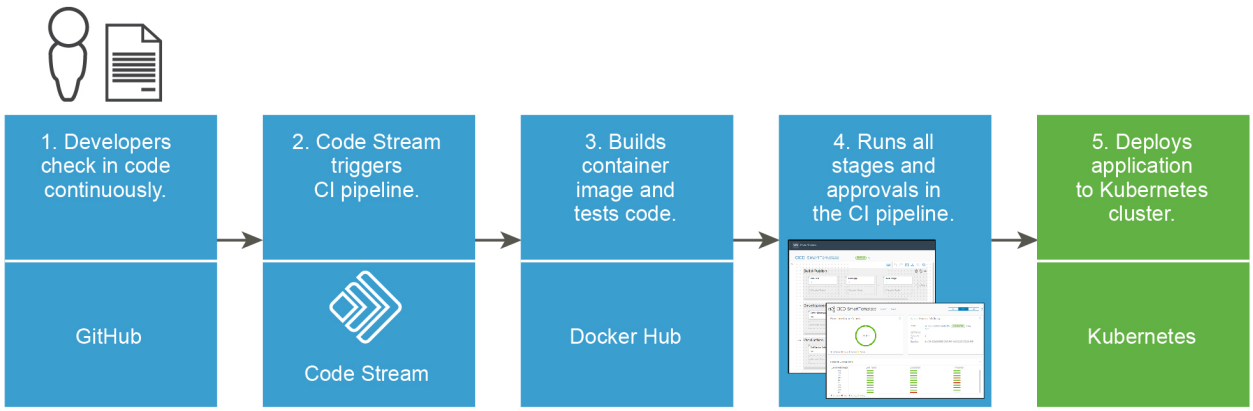
本章讨论了以下主题：

- 如何持续将来自 GitHub 或 GitLab 存储库的代码集成到 vRealize Automation Code Stream 中的管道
- 如何自动执行从 vRealize Automation Code Stream 中的 YAML 云模板部署的应用程序的发布
- 如何自动执行 vRealize Automation Code Stream 中应用程序到 Kubernetes 集群的发布

- 如何将 vRealize Automation Code Stream 中的应用程序部署到蓝绿部署
- 如何将自己的生成工具、测试工具和部署工具与 vRealize Automation Code Stream 集成
- 如何在下一个任务中使用云模板任务的资源属性
- 如何使用 REST API 将 vRealize Automation Code Stream 与其他应用程序集成
- 如何在 vRealize Automation Code Stream 中利用代码形式的管道

如何持续将来自 GitHub 或 GitLab 存储库的代码集成到 vRealize Automation Code Stream 中的管道

作为开发人员，您想要持续集成来自 GitHub 存储库或 GitLab Enterprise 存储库的代码。每次开发人员更新其代码并提交存储库更改时，vRealize Automation Code Stream 可以侦听这些更改并触发管道。



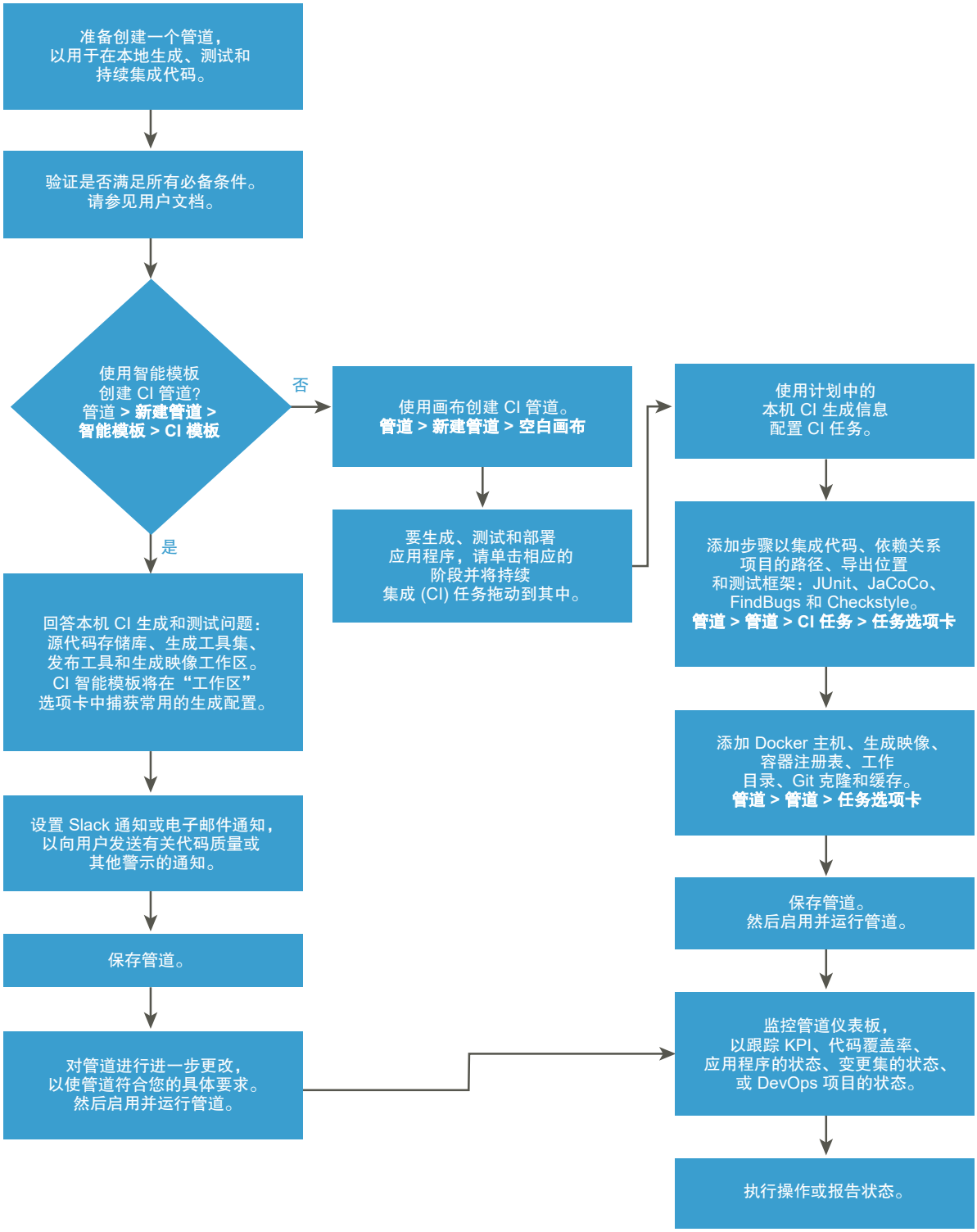
要让 vRealize Automation Code Stream 在代码发生更改时触发管道，可以使用 Git 触发器。每次提交代码更改时，vRealize Automation Code Stream 便会触发管道。

vRealize Automation Code Stream 管道工作区支持使用 Docker 和 Kubernetes 执行持续集成任务和自定义任务。

有关配置工作区的详细信息，请参见配置管道工作区。

以下流程图显示了使用智能管道模板创建管道或手动构建管道时可以采用的工作流。

图 5-1. 使用智能管道模板或手动创建管道的工作流



以下示例使用 Docker 工作区。

要生成代码，请使用 Docker 主机。您将使用 JUnit 和 JaCoCo 作为测试框架工具并将其包含到管道中，它们将运行单元测试和代码覆盖率测试。

然后，您可以使用持续集成智能管道模板创建持续集成管道，该管道可构建和测试代码并将代码部署到项目团队在 AWS 上的 Kubernetes 集群。要存储持续集成任务的代码依赖关系工件以便在代码生成时节省时间，可以使用缓存。

在生成和测试代码的管道任务中，可以包含多个持续集成步骤。这些步骤可驻留在同一个工作目录中，管道触发时 vRealize Automation Code Stream 会将源代码克隆到该目录。

要将代码部署到 Kubernetes 集群，可以在管道中使用 Kubernetes 任务。之后，必须启用并运行管道。然后，对存储库中的代码进行更改，并观察管道触发器。要在管道运行之后监控并报告管道趋势，请使用仪表板。

在以下示例中，要创建将代码持续集成到管道的持续集成管道，请使用持续集成智能管道模板。此示例使用 Docker 工作区。

（可选）可以手动创建管道，并向其添加阶段和任务。有关计划生成持续集成和手动创建管道的更多信息，请参见在手动添加任务之前在 vRealize Automation Code Stream 中规划 CICD 本地构建。

前提条件

- 计划生成持续集成。请参见在使用智能管道模板之前在 vRealize Automation Code Stream 中计划持续集成本机构建。
- 确认 GitLab 源代码存储库存在。要获得帮助，请咨询您的 vRealize Automation Code Stream 管理员。
- 添加 Git 端点。例如，请参见如何使用 vRealize Automation Code Stream 中的 Git 触发器运行管道。
- 要让 vRealize Automation Code Stream 侦听 GitHub 存储库或 GitLab 存储库中的更改并在发生更改时触发管道，请添加 Webhook。例如，请参见如何使用 vRealize Automation Code Stream 中的 Git 触发器运行管道。
- 添加 Docker 主机端点，该端点将为持续集成任务创建一个容器以供多个持续集成任务使用。有关端点的更多信息，请参见什么是 vRealize Automation Code Stream 中的端点 。
- 获取映像 URL、生成主机和构建映像的 URL。要获得帮助，请咨询您的 vRealize Automation Code Stream 管理员。
- 确认您使用 JUnit 和 JaCoCo 作为测试框架工具。
- 为持续集成构建设置外部实例：Jenkins、TFS 或 Bamboo。Kubernetes 插件部署代码。要获得帮助，请咨询您的 vRealize Automation Code Stream 管理员。

步骤

1 遵循必备条件。

2 要使用智能管道模板创建管道，请打开持续集成智能管道模板并填写表单。

   a 单击**管道 > 新建管道 > 智能模板 > 持续集成**。

   b 回答模板中有关源代码存储库、生成工具集、发布工具和生成映像工作区的问题。

   c 添加要向团队发送的 Slack 通知或电子邮件通知。

   d 要让智能管道模板创建管道，请单击**创建**。

   e 要对管道进行任何进一步更改，请单击**编辑**，进行更改，然后单击**保存**。

   f 启用并运行管道。

3 要手动创建管道，请将各个阶段和任务添加到画布中，并在持续集成任务中包括本地持续集成构建信息。

   a 单击**管道 > 新建管道 > 空白画布**。

   b 单击相关阶段，从导航窗格将多个持续集成任务拖动到该阶段。

   c 要配置持续集成任务，请单击该任务，然后单击**任务**选项卡。

   d 添加用于持续集成代码的步骤。

   e 包括依赖关系工件的路径。

   f 添加导出位置。

   g 添加要使用的测试框架工具。

   h 添加 Docker 主机和生成映像。

   i 添加容器注册表、工作目录和缓存。

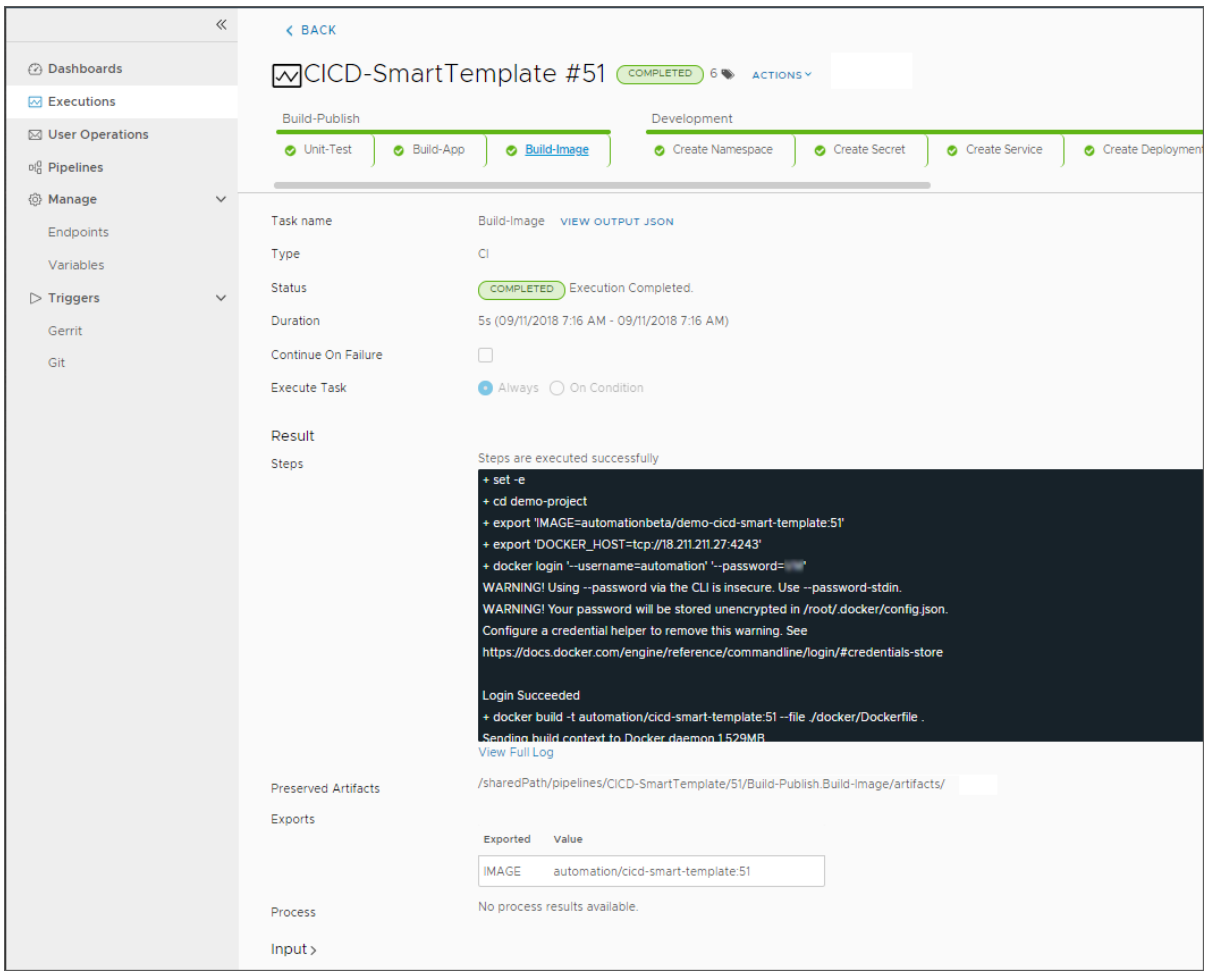
   j 保存管道，然后将其启用。

4 更改 GitHub 存储库或 GitLab 存储库中的代码。

   Git 触发器将激活管道，该管道开始运行。

5 要验证代码更改是否触发了管道，请单击**触发器 > Git > 活动**。

6 要查看管道的执行，请单击**执行**，然后验证相应的步骤是否已创建并导出生成映像。



7 要监控管道仪表板以便可以跟踪 KPI 和趋势，请单击**仪表板 > 管道仪表板**。

结果

恭喜！您已创建一个管道，该管道可持续将来自 GitHub 存储库或 GitLab 存储库的代码集成到管道中并部署构建映像。

后续步骤

有关更多信息，请参见供 vRealize Automation Code Stream 管理员和开发人员使用的更多资源。

如何自动执行从 vRealize Automation Code Stream 中的 YAML 云模板部署的应用程序的发布

作为开发人员，您需要一个管道，以在每次提交更改时都从内部部署 GitHub 实例中获取自动化云模板。您需要使用该管道将 WordPress 应用程序部署到 Amazon Web Services (AWS) EC2 或数据中心。vRealize Automation Code Stream 将从该管道调用云模板，并自动执行该云模板的持续集成和持续交付 (CICD) 以部署您的应用程序。

要创建并触发管道，您需要一个 VMware 云模板。

对于 vRealize Automation Code Stream 云模板任务中的**云模板源**，可以选择以下两项之一：

- **Cloud Assembly 模板**，作为源控制。在这种情况下，不需要 GitLab 或 GitHub 存储库。
- **源控制**（如果使用 GitLab 或 GitHub 作为源控制）。在这种情况下，必须具有 Git webhook 并通过该 webhook 触发管道。

如果您的 GitHub 存储库中有 YAML 云模板，并且希望在管道中使用该云模板，则需要执行以下操作。

1 在 vRealize Automation Cloud Assembly 中，将云模板推送到 GitHub 存储库。

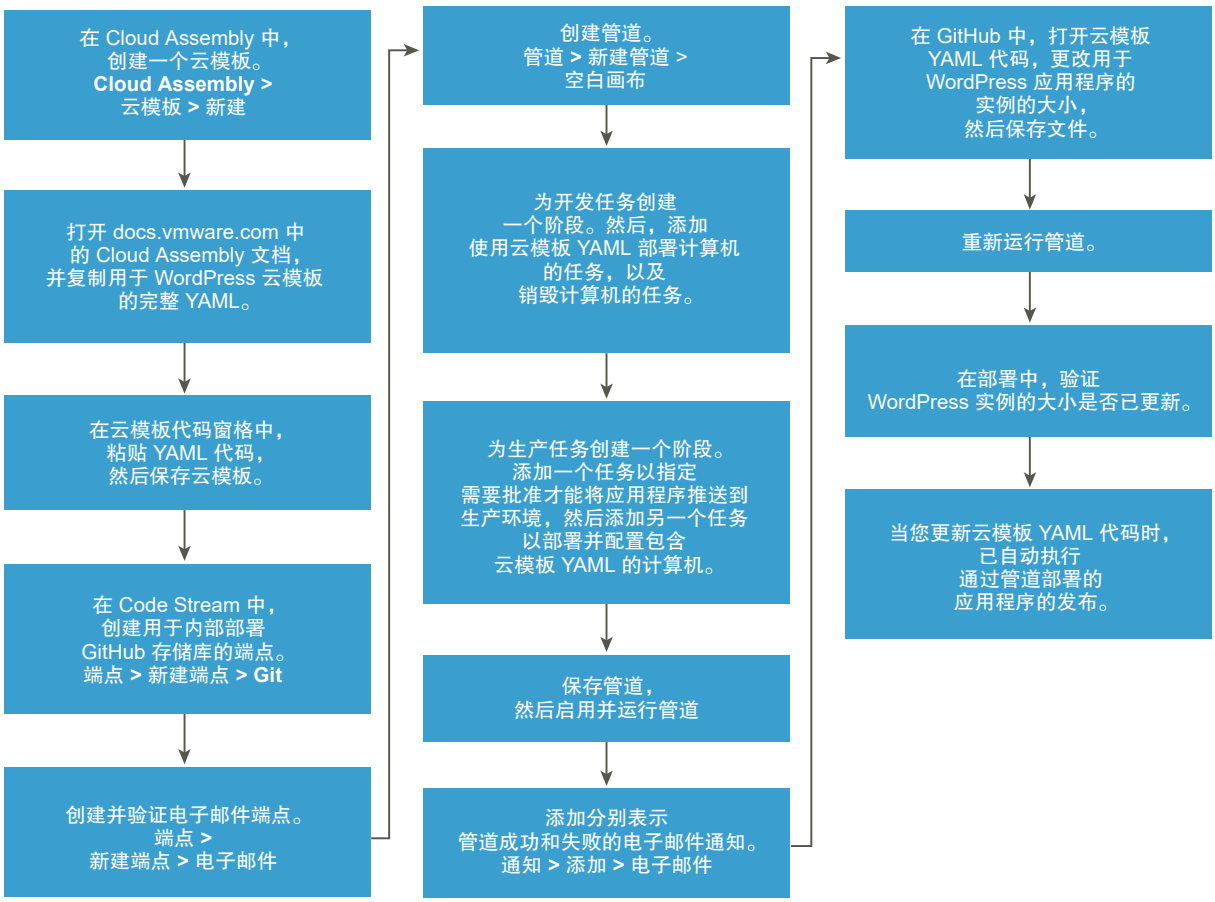
2 在 vRealize Automation Code Stream 中，创建 Git 端点。然后，创建使用 Git 端点和管道的 Git webhook。

3 要触发管道，请更新 GitHub 存储库中的任何文件并提交所做的更改。

如果您的 GitHub 存储库中没有 YAML 云模板，并且希望使用源控制中的云模板，请使用以下过程了解如何操作。该过程展示了如何为 WordPress 应用程序创建云模板，并从内部部署 GitHub 存储库中触发该云模板。每当您更改 YAML 云模板时，管道都会触发并自动发布应用程序。

- 在 vRealize Automation Cloud Assembly 中，将添加云帐户、添加云区域并创建云模板。
- 在 vRealize Automation Code Stream 中，将为托管云模板的内部部署 GitHub 存储库添加端点。然后，将云模板添加到管道中。

以下用例示例展示了如何使用内部部署 GitHub 存储库中的云模板。



前提条件

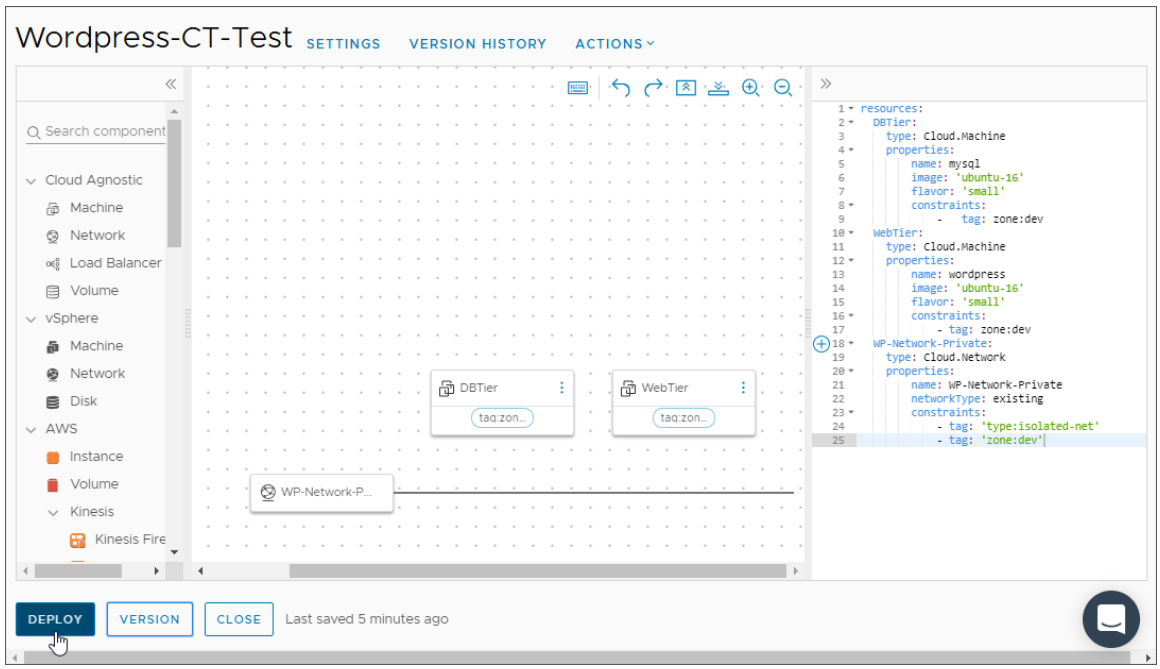
- 在 vRealize Automation Cloud Assembly 基础架构中添加一个云帐户和一个云区域。请参见 vRealize Automation Cloud Assembly 文档。
- 要按照以下过程创建云模板，请将 WordPress YAML 代码复制到剪贴板。请参见 vRealize Automation Cloud Assembly 文档中 WordPress 用例中的云模板 YAML 代码。
- 将 WordPress 应用程序的 YAML 代码添加到 GitHub 实例。
- 为 Git 触发器添加 Webhook，以便管道可以在您提交更改时提取 YAML 代码。在 vRealize Automation Code Stream 中，单击**触发器 > Git > 适用于 Git 的 Webhook**。
- 要使用云模板任务，您必须具有任意 vRealize Automation Cloud Assembly 角色。

步骤

1 在 vRealize Automation Cloud Assembly 中，执行以下步骤。

   a 单击 **VMware Cloud Templates**，然后为 WordPress 应用程序创建云模板和部署。

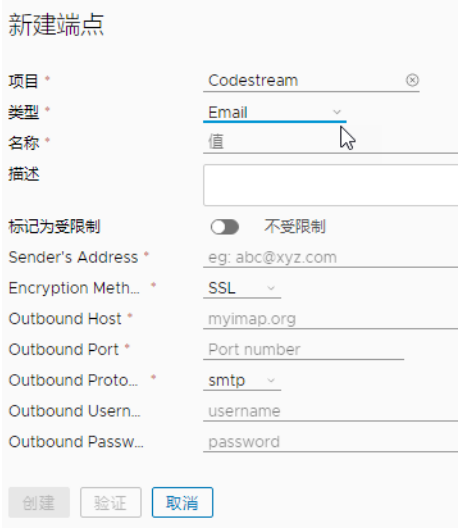
   b 将已复制到剪贴板的 WordPress YAML 代码粘贴到云模板，然后部署该云模板。



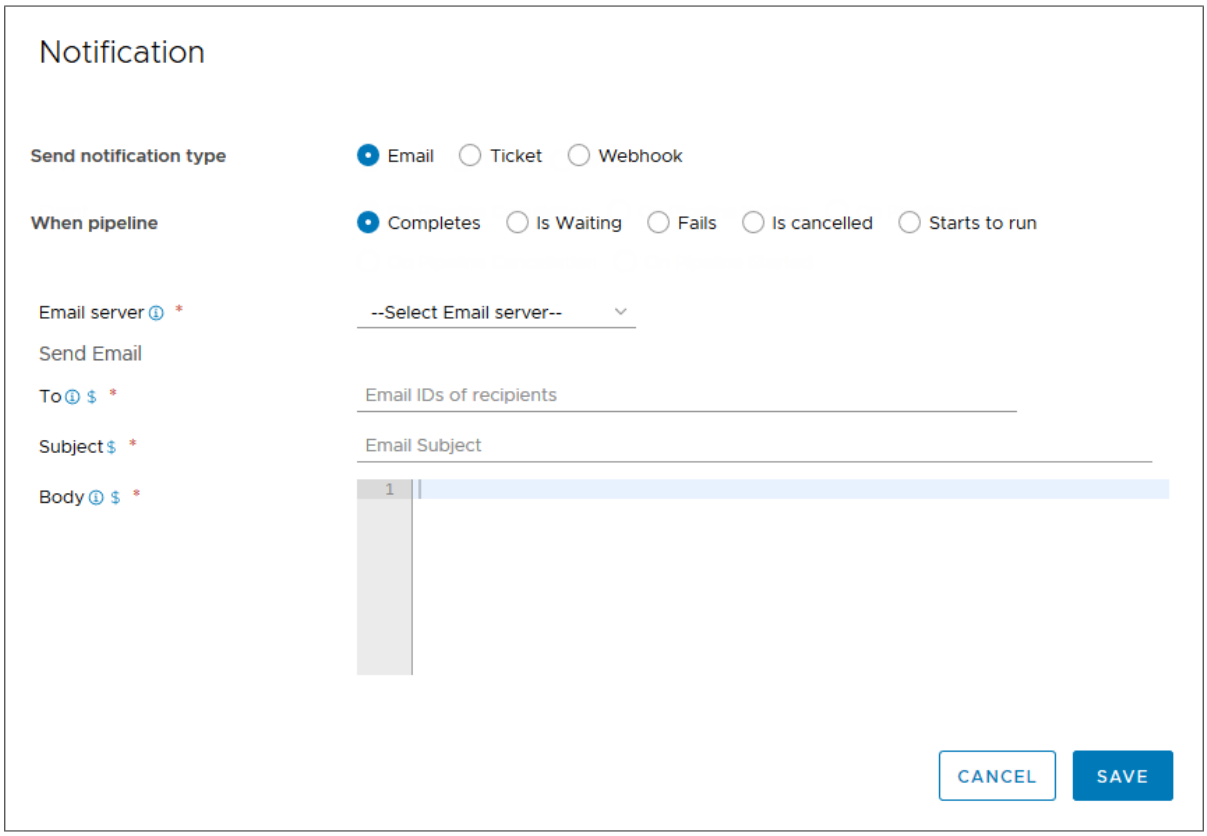
2 在 vRealize Automation Code Stream 中，创建端点。

a 为 YAML 文件所在的内部部署 GitHub 存储库添加 Git 端点。

b 添加电子邮件端点，以便可以在管道运行时向用户通知管道状态。



3 创建一个管道，并添加分别表示管道成功和失败的电子邮件通知。



4 添加开发阶段和云模板任务。

a 添加用于部署计算机的云模板任务，并将该任务配置为使用 WordPress 应用程序的云模板 YAML。

```
resources:
  DBTier:
    type: Cloud.Machine
    properties:
        name: mysql
        image: 'ubuntu-16'
        flavor: 'small'
        constraints:
            -   tag: zone:dev
  WebTier:
    type: Cloud.Machine
    properties:
        name: wordpress
        image: 'ubuntu-16'
        flavor: 'small'
        constraints:
            - tag: zone:dev
 WP-Network-Private:
    type: Cloud.Network
    properties:
        name: WP-Network-Private
        networkType: existing
        constraints:
            - tag: 'type:isolated-net'
            - tag: 'zone:dev'
```

b 添加销毁计算机以释放资源的云模板任务。

5 添加生产阶段，并包含批准任务和部署任务。

a 添加用户操作任务，以指定需要批准才能将 WordPress 应用程序推送到生产环境。

b 添加用于部署计算机的云模板任务，并将其配置为使用 WordPress 应用程序的云模板 YAML。

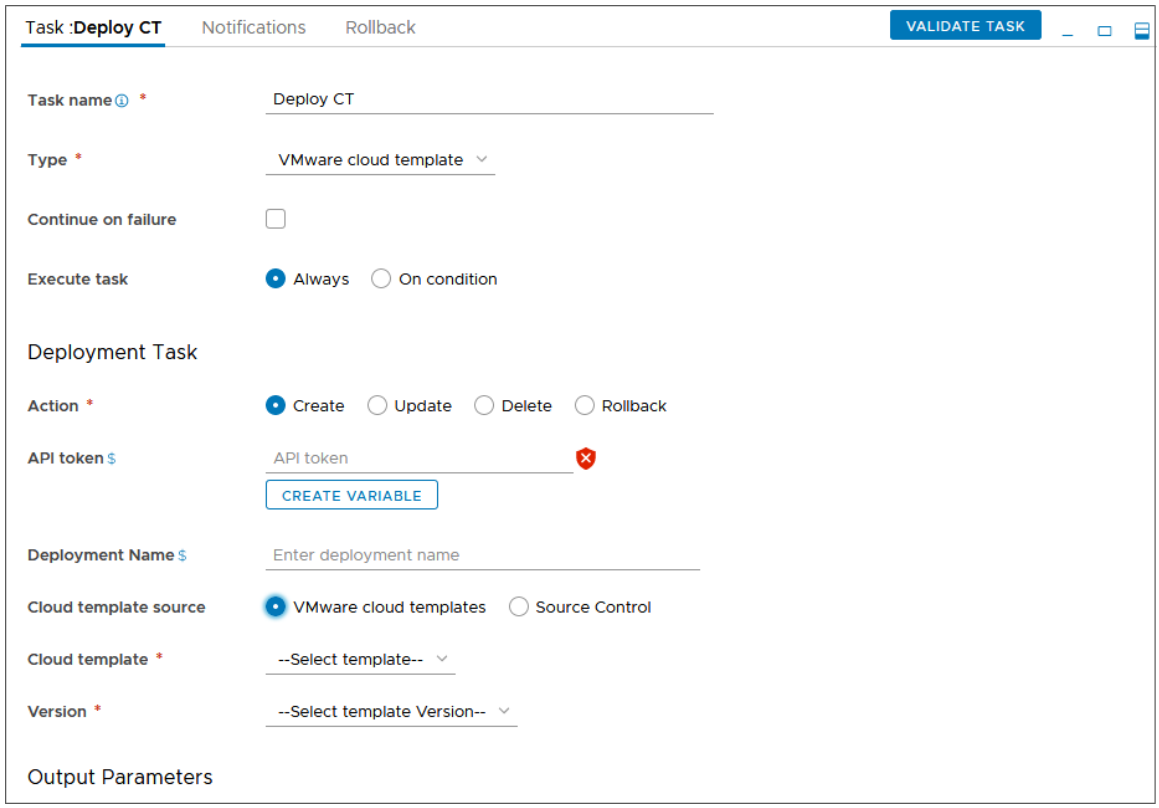
选择**创建**时，部署名称必须唯一。如果将名称留空， vRealize Automation Code Stream 将为其分配唯一的随机名称。

以下是您在自己的用例中选择**回滚**时必须了解的内容：如果选择**回滚**操作并输入**回滚版本**，则版本必须采用 **n-x** 形式。例如，**n-1**、**n-2**、**n-3** 等。如果在 vRealize Automation Code Stream 以外的任何位置创建并更新部署，则允许回滚。

登录到 vRealize Automation Code Stream 时，它将获取用户令牌，令牌有效时间为 30 分钟。如果管道持续运行时间长，则云模板任务之前任务的运行时间为 30 分钟或更长时间时，用户令牌将过期。因此，云模板任务将失败。

为确保管道的运行时间可超过 30 分钟，可以输入可选的 API 令牌。当 vRealize Automation Code Stream 调用云模板时，API 令牌将持续存在，并且云模板任务将继续使用 API 令牌。

使用 API 令牌作为变量时，将对其进行加密。否则，将以纯文本形式使用。



6 运行管道。

要验证每个任务是否已成功完成，请单击正在执行的任务，然后检查部署详细信息中的状态，以查看详细资源信息。

7 在 GitHub 中，将 WordPress 服务器实例的特定实例从 `small` 修改为 `medium`。

当您提交更改时，管道将触发。它会从 GitHub 存储库中提取更新的代码，并构建您的应用程序。

```
WebTier:
    type: Cloud.Machine
    properties:
        name: wordpress
        image: 'ubuntu-16'
        flavor: 'medium'
        constraints:
            - tag: zone:dev
```

8 重新运行管道，验证管道是否已成功并且已将 WordPress 实例的特定实例从 `small` 更改为 `medium`。

结果

恭喜！您已自动执行从 YAML 云模板部署的应用程序的发布。

后续步骤

要了解有关如何使用 vRealize Automation Code Stream 的更多信息，请参见第 5 章 vRealize Automation Code Stream 使用教程。

有关其他参考，请参见 供 vRealize Automation Code Stream 管理员和开发人员使用的更多资源。

如何自动执行 vRealize Automation Code Stream 中应用程序到 Kubernetes 集群的发布

作为 vRealize Automation Code Stream 管理员或开发人员，您可以使用 vRealize Automation Code Stream 和 VMware Tanzu Kubernetes Grid Integrated Edition（以前称为 VMware Enterprise PKS）自动将软件应用程序部署到 Kubernetes 集群。此用例提到其他方法，这些方法可用于自动执行应用程序的发布。

在此用例中，您将创建包含两个阶段的管道，并使用 Jenkins 生成和部署应用程序。

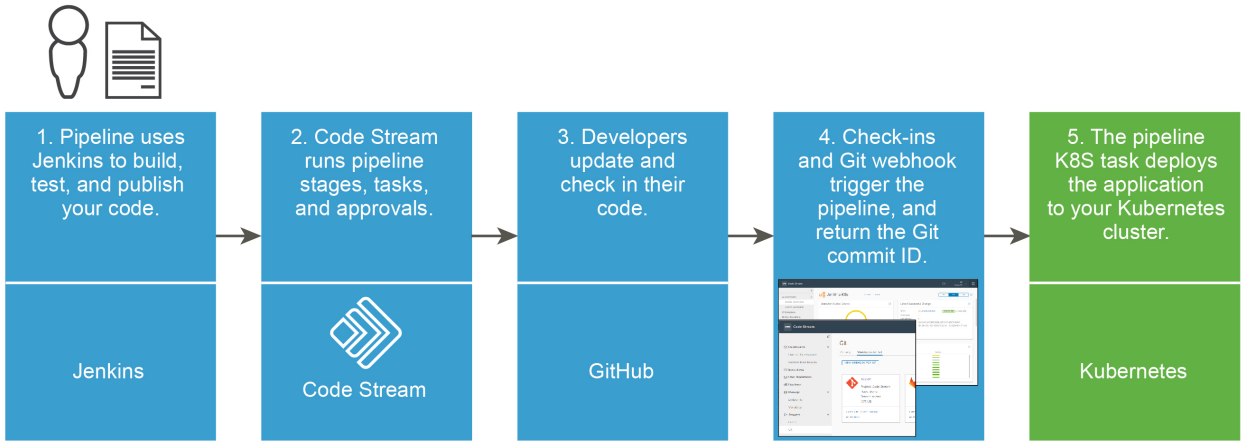
- 第一个阶段是进行开发。此阶段使用 Jenkins 从 GitHub 存储库中的分支提取代码，然后生成、测试并发布该代码。
- 第一个阶段是进行部署。此阶段运行需要关键用户批准的用户操作任务，然后管道才可以将应用程序部署到 Kubernetes 集群。

在管道工作区中使用 Kubernetes 端点时， vRealize Automation Code Stream 会创建必要的 Kubernetes 资源（如 ConfigMap、密钥和 Pod）以运行持续集成 (CI) 任务或自定义任务。 vRealize Automation Code Stream 使用 NodePort 与容器进行通信。

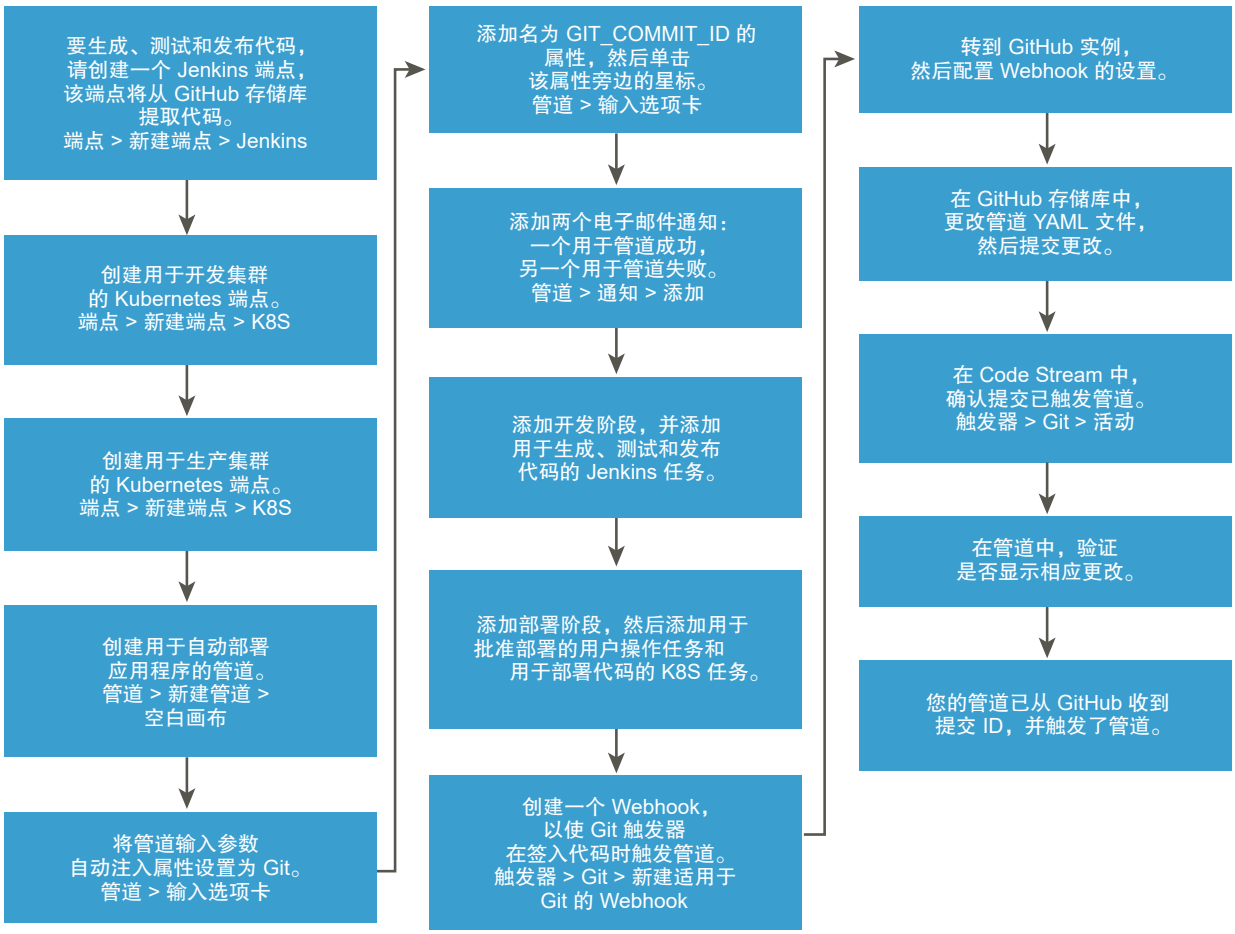
要在各个管道运行过程中共享数据，您必须提供持久卷声明， vRealize Automation Code Stream 会将此持久卷声明挂载到容器以存储数据，并将其用于后续管道运行。

vRealize Automation Code Stream 管道工作区支持使用 Docker 和 Kubernetes 执行持续集成任务和自定义任务。

有关配置工作区的详细信息，请参见配置管道工作区。



开发工具、开发实例和管道 YAML 文件必须可用，以便管道可以生成、测试、发布和部署应用程序。管道会将应用程序部署到 AWS 上的 Kubernetes 集群开发实例和生产实例。



可用于自动执行应用程序的发布的其他方法是：

- 可以使用 vRealize Automation Code Stream 本地构建功能和 Docker 生成主机，而不必使用 Jenkins 来构建应用程序。

- 可以将应用程序部署到 Amazon Web Services (AWS) 集群，而不必部署到 Kubernetes 集群。

有关使用 vRealize Automation Code Stream 本地构建功能和 Docker 主机的更多信息，请参见：

- 在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建
- 在手动添加任务之前在 vRealize Automation Code Stream 中规划 CICD 本地构建

前提条件

- 确认要部署的应用程序代码位于工作 GitHub 存储库中。
- 验证您是否具有 Jenkins 工作实例。
- 验证您是否具有工作电子邮件服务器。
- 在 vRealize Automation Code Stream 中，创建连接到电子邮件服务器的电子邮件端点。
- 在 Amazon Web Services (AWS) 上设置两个分别用于开发和生产的 Kubernetes 集群，管道会将应用程序部署到这两个集群。
- 验证 GitHub 存储库是否包含管道的 YAML 代码，或者验证是否包含定义环境的元数据和规范的 YAML 文件。

步骤

1 在 vRealize Automation Code Stream 中，单击**端点 > 新建端点**，然后创建 Jenkins 端点以在管道中用于从 GitHub 存储库提取代码。

2 要创建 Kubernetes 端点，请单击**新建端点**。

   a 创建用于 Kubernetes 开发集群的端点。

   b 创建用于 Kubernetes 生产集群的端点。

   Kubernetes 集群的 URL 可能包含端口号，也可能不包含端口号。

   例如：

   `https://10.111.222.333:6443`

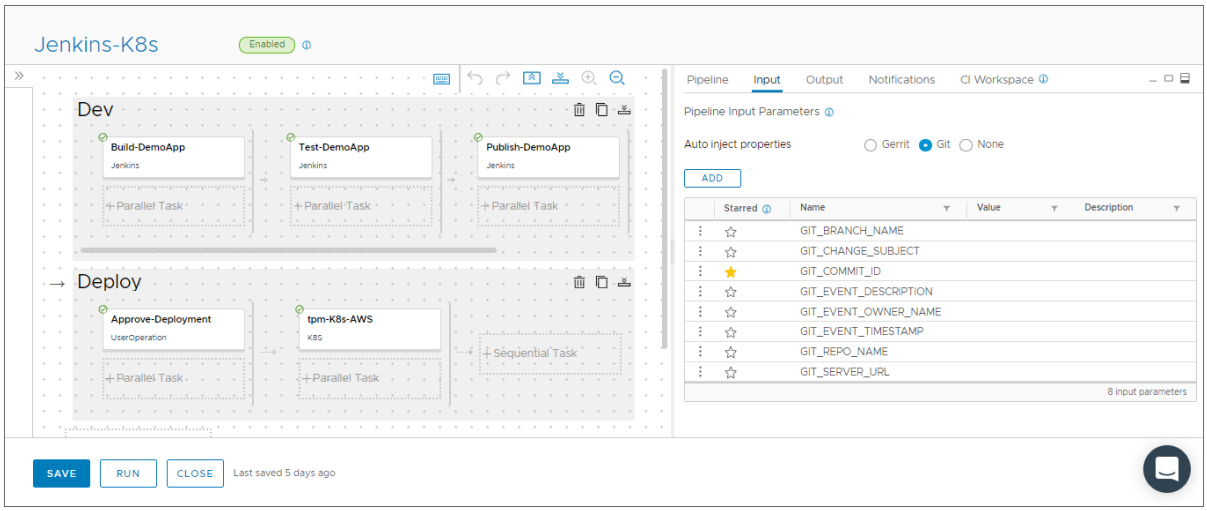
   `https://api.kubernetesserver.fa2c1d78-9f00-4e30-8268-4ab81862080d.k8s-user.com`

3 创建一个管道以用于将应用程序（例如 Wordpress）的容器部署到 Kubernetes 开发集群，并设置管道的输入属性。

a 要使管道能够识别 GitHub 中会触发管道的代码提交，请在管道中单击**输入**选项卡并选择**自动注入属性**。

b 添加名为 **GIT_COMMIT_ID** 的属性，然后单击该属性旁边的星标。

当管道运行时，管道执行将显示 Git 触发器返回的提交 ID。

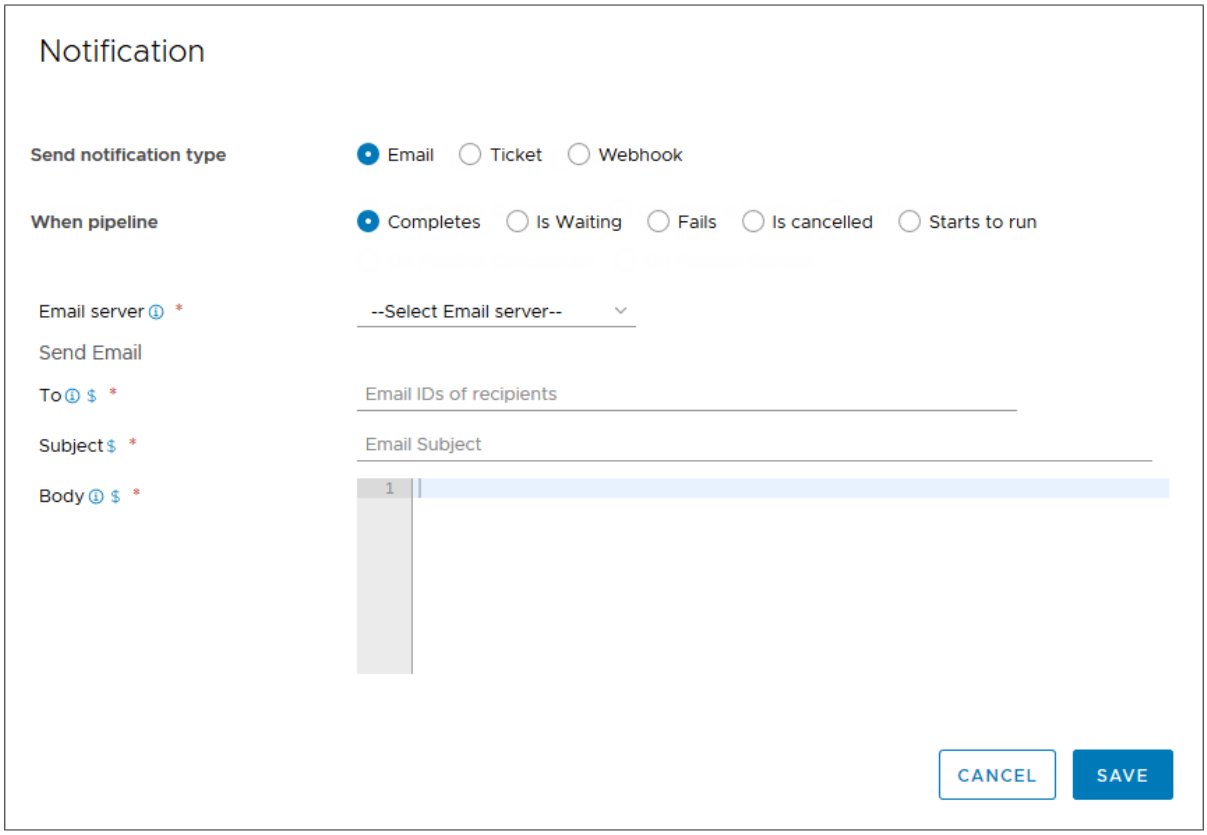


4 添加通知以在管道成功或失败时发送电子邮件。

a 在管道中，单击**通知**选项卡，然后单击**添加**。

b 要添加在管道完成运行时发送的电子邮件通知，请依次选择**电子邮件**和**完成**。然后选择电子邮件服务器，输入电子邮件地址，并单击**保存**。

c 要添加在管道失败时发送的另一个电子邮件通知，请选择**失败**，然后单击**保存**。

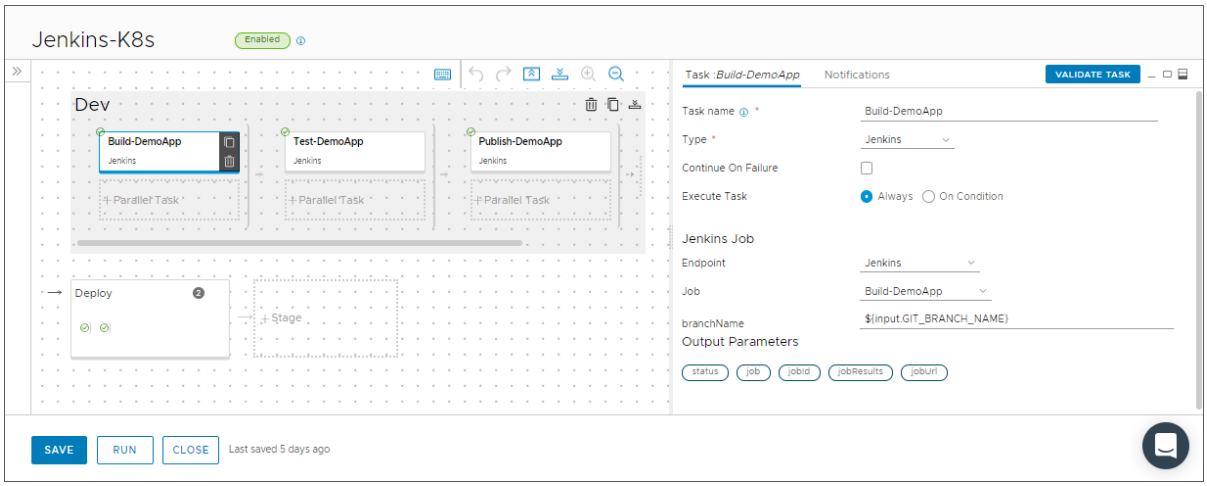


5 向管道添加开发阶段，并添加生成、测试和发布应用程序的任务。然后验证每个任务。

a 要生成应用程序，请添加使用 Jenkins 端点的 Jenkins 任务，并从 Jenkins 服务器运行生成作业。然后，要使管道能够提取代码，请按以下格式输入 Git 分支：
`${input.GIT_BRANCH_NAME}`

b 要测试应用程序，请添加使用同一个 Jenkins 端点的 Jenkins 任务，并从 Jenkins 服务器运行测试作业。然后输入与上述相同的 Git 分支。

c 要发布应用程序，请添加使用同一个 Jenkins 端点的 Jenkins 任务，并从 Jenkins 服务器运行发布作业。然后输入与上述相同的 Git 分支。

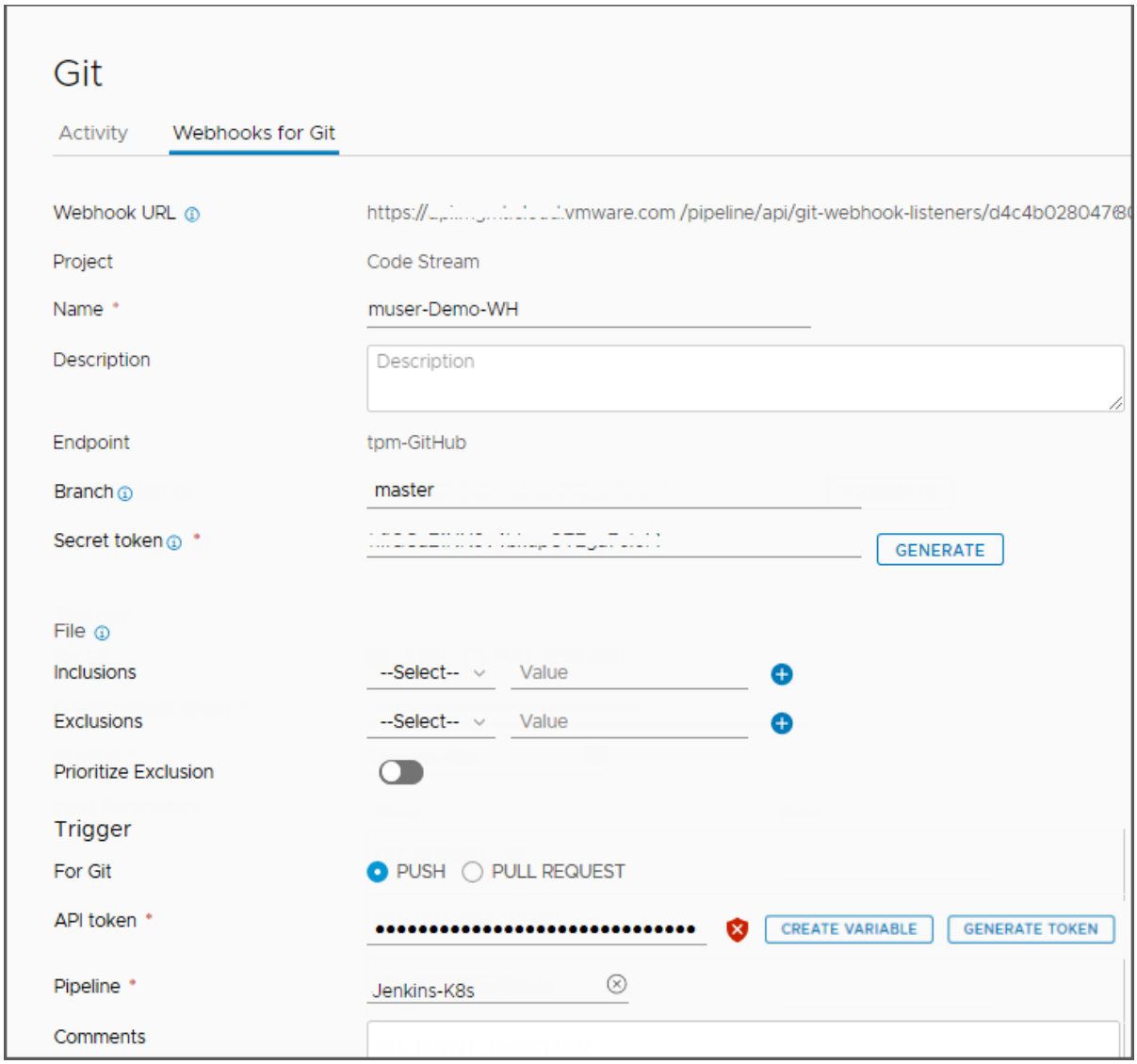


6 向管道添加部署阶段，然后添加需要批准才能部署应用程序的任务，并添加将应用程序部署到 Kubernetes 集群的另一个任务。然后验证每个任务。

a 要指定需要批准才能部署应用程序，请添加一个用户操作任务，添加必须批准该任务的用户的电子邮件地址，并输入消息。然后启用**发送电子邮件**。

b 要部署应用程序，请添加一个 Kubernetes 任务。然后，在 Kubernetes 任务属性中，选择您的 Kubernetes 开发集群，选择**创建**操作，并选择**本地定义**负载源。然后选择本地 YAML 文件。

7 添加可使 vRealize Automation Code Stream 能够使用 Git 触发器的 Git Webhook，该触发器会在开发人员提交代码时触发管道。



8 要测试管道，请转到 GitHub 存储库，更新应用程序 YAML 文件，并提交更改。

a 在 vRealize Automation Code Stream 中，验证是否显示该提交。

a 单击**触发器 > Git > 活动**。

b 查找管道的触发器。

c 单击**仪表板 > 管道仪表板**。

d 在管道仪表板上，在最新的成功更改区域中找到 GIT_COMMIT_ID。

9 检查管道代码并验证是否显示相应更改。

结果

恭喜！您已自动执行软件应用程序到 Kubernetes 集群的部署。

示例： 将应用程序部署到 Kubernetes 集群的示例管道 YAML

对于此示例中使用的管道类型，YAML 类似于以下代码：

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
eyJzeW1waG9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJ1c2VybmFtZSI6InRhbmdvLWJldGEyIiwicGFzc3dvcmQiOi
JhRGstcmVOLW1UQi1IejciLCJlbWFpbCI6InRhbmdvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoiZEdGdVoyOHRZbVYw
WVRJNllVUnJMWEpsVGkxdFZFSXRTSG8zIn19
kind: Secret
metadata:
  name: jfrog
  namespace: ${input.GIT_BRANCH_NAME}
type: kubernetes.io/dockercfg
---
apiVersion: v1
kind: Service
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  ports:
    - port: 80
  selector:
    app: codestream
    tier: frontend
  type: LoadBalancer
---
apiVersion: extensions/v1
kind: Deployment
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  selector:
    matchLabels:
      app: codestream
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: codestream
```

```
        tier: frontend
    spec:
      containers:
      - name: codestream
        image: cas.jfrog.io/codestream:${input.GIT_BRANCH_NAME}-${Dev.PublishApp.output.jobId}
        ports:
        - containerPort: 80
          name: codestream
      imagePullSecrets:
      - name: jfrog
```

后续步骤

要将软件应用程序部署到 Kubernetes 生产集群，请再次执行相应步骤并选择生产集群。

要了解有关将 vRealize Automation Code Stream 与 Jenkins 集成的更多信息，请参见如何将 vRealize Automation Code Stream 与 Jenkins 集成。

如何将 vRealize Automation Code Stream 中的应用程序部署到蓝绿部署

蓝绿部署是一个部署模型，该模型使用两台 Docker 主机，您可以在 Kubernetes 集群中以相同方式部署和配置这两台主机。使用蓝绿部署模型，可以减少 vRealize Automation Code Stream 中的管道部署应用程序时环境中出现的停机。

部署模型中的蓝部署实例和绿部署实例分别用于不同用途。一次只有一个实例可以接受部署应用程序的实时流量，而且每个实例在特定时间接受该流量。蓝部署实例接收应用程序的第一个版本，而绿部署实例接收第二个版本。

蓝绿部署环境中的负载均衡器将确定实时流量在部署应用程序时采用的路由。使用蓝绿部署模型，您的环境将保持正常运行，用户不会注意到任何停机，而且管道会持续将应用程序集成并部署到生产环境。

在 vRealize Automation Code Stream 中创建的管道以两个阶段表示蓝绿部署模型。一个是开发阶段，另一个是生产阶段。

vRealize Automation Code Stream 管道工作区支持使用 Docker 和 Kubernetes 执行持续集成任务和自定义任务。

有关配置工作区的信息，请参见配置管道工作区。

表 5-2. 蓝绿部署的开发阶段任务

任务类型	任务
Kubernetes	为蓝绿部署创建命名空间。
Kubernetes	为 Docker Hub 创建密钥。
Kubernetes	创建用于部署应用程序的服务。
Kubernetes	创建蓝部署。

表 5-2. 蓝绿部署的开发阶段任务 （续）

任务类型	任务
Poll	验证蓝部署。
Kubernetes	移除命名空间。

表 5-3. 蓝绿部署的生产阶段任务

任务类型	任务
Kubernetes	绿部署将从蓝部署获取服务详细信息。
Kubernetes	获取绿部署副本集的详细信息。
Kubernetes	创建绿部署，并使用密钥来提取容器映像。
Kubernetes	更新服务。
Poll	验证部署在生产 URL 上是否成功。
Kubernetes	完成蓝部署。
Kubernetes	移除蓝部署。

要使用您自己的蓝绿部署模型来部署应用程序，请在 vRealize Automation Code Stream 中创建包含两个阶段的管道。第一个阶段包含将应用程序部署到蓝部署实例的蓝部署任务，而第二个阶段包含将应用程序部署到绿部署实例的绿部署任务。

可以使用 CICD 智能管道模板创建管道。模板将为您创建各个管道阶段和任务，并将部署选择包括在内。

如果您手动创建管道，则必须计划管道阶段。例如，请参见在手动添加任务之前在 vRealize Automation Code Stream 中规划 CICD 本地构建。

在此示例中，您使用 CICD 智能管道模板来创建蓝绿部署管道。

前提条件

- 确认您可以访问 AWS 上正常工作的 Kubernetes 集群。
- 确认您已设置一个蓝绿部署环境，并且已将蓝部署实例和绿部署实例配置为完全相同。
- 在 vRealize Automation Code Stream 中创建一个 Kubernetes 端点，用于将应用程序映像部署到 AWS 上的 Kubernetes 集群。
- 熟悉如何使用 CICD 智能管道模板。请参见在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建。

步骤

1 单击**管道 > 新建管道 > 智能模板 > CI/CD 模板**。

2 输入 CICD 智能管道模板 CI 部分的信息，然后单击**下一步**。

要获取帮助，请参见在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建。

3 完成智能管道模板的 CD 部分

a 选择用于应用程序部署的环境。例如，**Dev** 和 **Prod**。

b 选择管道将用于部署的服务。

c 在“部署”区域，选择“开发”环境和“生产”环境对应的集群端点。

d 对于“生产部署模型”，选择 **蓝绿部署**，然后单击**创建**。

结果

恭喜！您已使用智能管道模板创建了一个管道，该管道可将应用程序部署到 AWS 上的 Kubernetes 生产集群中的蓝绿部署实例。

示例： 适用于部分蓝绿部署任务的示例 YAML 代码

蓝绿部署的 Kubernetes 管道任务中显示的 YAML 代码类似以下创建命名空间、服务和部署的示例。如果需要从专有存储库下载映像，则 YAML 文件必须包含具有 Docker 配置密钥的部分。请参见在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建中的 CD 部分。

智能管道模板创建管道之后，您可以根据自己的部署需求修改任务。

用于创建示例命名空间的 YAML 代码：

```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream-82855
  namespace: codestream-82855
```

用于创建示例服务的 YAML 代码：

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: codestream-demo
  name: codestream-demo
  namespace: bluegreen-799584
spec:
  minReadySeconds: 0
  ports:
  - port: 80
  selector:
    app: codestream-demo
    tier: frontend
  type: LoadBalancer
```

用于创建示例部署的 YAML 代码：

```
apiVersion: extensions/v1
kind: Deployment
metadata:
  labels:
    app: codestream-demo
  name: codestream-demo
  namespace: bluegreen-799584
spec:
  minReadySeconds: 0
  replicas: 1
  selector:
    matchLabels:
      app: codestream-demo
      tier: frontend
  template:
    metadata:
      labels:
        app: codestream-demo
```

```
        tier: frontend
    spec:
      containers:
      - image: ${input.image}:${input.tag}
        name: codestream-demo
        ports:
        - containerPort: 80
          name: codestream-demo
      imagePullSecrets:
      - name: jfrog-2
      minReadySeconds: 0
```

后续步骤

要了解有关如何使用 vRealize Automation Code Stream 的更多信息，请参见第 5 章 vRealize Automation Code Stream 使用教程。

要回滚部署，请参见如何在 vRealize Automation Code Stream 中回滚部署。

有关其他参考，请参见 供 vRealize Automation Code Stream 管理员和开发人员使用的更多资源。

如何将自己的生成工具、测试工具和部署工具与 vRealize Automation Code Stream 集成

作为 DevOps 管理员或开发人员，您可以创建自定义脚本来扩展 vRealize Automation Code Stream 的功能。

使用您的脚本，可以将 vRealize Automation Code Stream 与您自己用于生成、测试和部署应用程序的持续集成 (CI) 和持续交付 (CD) 工具以及 API 集成。如果不想公开您的应用程序 API，则自定义脚本特别有用。

自定义脚本几乎可以执行与生成工具、测试工具和部署工具与 vRealize Automation Code Stream 集成所需的任何操作。例如，脚本可与管道工作区配合工作，以支持构建和测试应用程序的持续集成任务以及部署应用程序的持续交付任务。当管道已完成或发生其他许多事件时，自定义脚本可以向 Slack 发送消息。

vRealize Automation Code Stream 管道工作区支持使用 Docker 和 Kubernetes 执行持续集成任务和自定义任务。

有关配置工作区的详细信息，请参见配置管道工作区。

您可以采用受支持的语言之一编写自定义脚本。在脚本中，可以包括业务逻辑，以及定义输入和输出。输出类型可以包括数字、字符串、文本和密码。可以使用不同的业务逻辑、输入和输出创建自定义脚本的多个版本。

在自定义任务中使用管道运行某个版本的脚本。所创建的脚本位于 vRealize Automation Code Stream 实例中。

当管道使用自定义集成时，如果您尝试删除该自定义集成，则会显示一条错误消息，指示无法将其删除。

如果删除自定义集成，将移除自定义脚本的所有版本。如果现有管道的自定义任务使用任何版本的脚本，则该管道将失败。为了确保现有管道不会失败，可以弃用并撤消不再希望使用的脚本版本。如果没有管道使用该版本，便可以将其删除。

表 5-4. 编写自定义脚本之后执行的操作

执行的操作...	有关该操作的更多信息...
向管道添加自定义任务。	自定义任务： ■ 在与管道中其他 CI 任务相同的容器中运行。 ■ 包含输入变量和输出变量，脚本会在管道运行自定义任务之前填充这些变量。 ■ 支持您在脚本中定义为输入和输出的多种数据类型和各种元数据类型。
在自定义任务中选择脚本。	在脚本中声明输入属性和输出属性。
保存管道，然后启用并运行管道。	当管道运行时，自定义任务将调用指定版本的脚本并运行其中的业务逻辑，从而将构建工具、测试工具和部署工具与 vRealize Automation Code Stream 集成。
管道运行之后，查看执行。	验证管道是否产生预期结果。

使用调用自定义集成版本的自定义任务时，可以在管道**工作区**选项卡上将自定义环境变量作为名称-值对包括在内。当生成器映像创建运行 CI 任务并部署映像的工作区容器时， vRealize Automation Code Stream 会将环境变量传递到该容器。

例如，当您的 vRealize Automation Code Stream 实例需要使用 Web 代理，并且使用 Docker 主机为自定义集成创建容器时， vRealize Automation Code Stream 将运行管道并将 Web 代理设置变量传递到该容器。

表 5-5. 环境变量名称-值对示例

名称	值
HTTPS_PROXY	http://10.0.0.255:1234
https_proxy	http://10.0.0.255:1234
NO_PROXY	10.0.0.32, *.dept.vsphere.local
no_proxy	10.0.0.32, *.dept.vsphere.local
HTTP_PROXY	http://10.0.0.254:1234
http_proxy	http://10.0.0.254:1234
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

名称-值对在用户界面中如下所示：

此示例创建的自定义集成将 vRealize Automation Code Stream 连接到 Slack 实例并向 Slack 通道发布消息。

前提条件

- 要编写自定义脚本，请确认您掌握以下语言之一：Python 2、Python 3 或 Node.js，或者掌握以下任意 shell 语言：Bash、sh 或 zsh。
- 使用已安装的 Node.js 或 Python 运行时生成容器映像。

步骤

1 创建自定义集成。

   a 单击**自定义集成 > 新建**，然后输入相关的名称。

   b 选择首选运行时环境。

   c 单击**创建**。

   脚本将打开并显示代码，该脚本包含所需的运行时环境。例如，`runtime: "nodejs"`。脚本必须包含运行时，因为生成器映像使用该运行时来确保添加到管道中的自定义任务在管道运行时将成功。否则，自定义任务将失败。

   自定义集成 YAML 的主要区域包括运行时、代码、输入属性和输出属性。此过程介绍了各种类型和语法。

自定义集成 YAML 键	说明
runtime	vRealize Automation Code Stream 运行代码的任务运行时环境，可以是以下不区分大小写的字符串之一： ■ nodejs ■ python2 ■ python3 ■ shell 如果未提供任何内容，将默认采用 shell。
code	作为自定义任务的一部分运行的自定义业务逻辑。
inputProperties	作为自定义任务配置的一部分进行捕获的输入属性数组。这些属性通常在代码中使用。
outputProperties	可从自定义任务中导出以传播到管道的输出属性数组。

2 使用可用的数据类型和元数据在脚本中声明输入属性。

   输入属性将在 YAML 的 `code:` 节中作为上下文传入到脚本。

自定义任务 YAML 输入键	说明	必需
type	要显示的输入类型： ■ **text** ■ **textarea** ■ **number** ■ **checkbox** ■ **password** ■ **select**	是
name	自定义任务的输入的名称或字符串，将插入到自定义集成 YAML 代码中。对于为自定义集成定义的每个输入属性必须唯一。	是
title	管道模型画布上自定义任务的输入属性的文本字符串标签。如果留空，默认使用 **name**。	否
required	确定用户在配置自定义任务时是否必须填写输入属性。设置为 true 或 false。若为 true，则如果当用户在管道画布上配置自定义任务时未提供值，任务的状态仍为未配置。	否
placeHolder	未提供值时输入属性条目区域中的默认文本。映射到 html 占位符属性。仅某些输入属性类型提供此支持。	否
defaultValue	当自定义任务在管道模型页面上显示时，用于填充输入属性条目区域的默认值。	否
bindable	确定在为管道画布上的自定义任务建模时，输入属性是否接受美元符号变量。在标题旁边添加 **$** 指示符。仅某些输入属性类型提供此支持。	否
labelMessage	作为用户的帮助工具提示的字符串。在输入标题旁边添加一个工具提示图标 **i**。	否
enum	接受显示“选择”输入属性选项的值数组。仅某些输入属性类型提供此支持。 当用户选择并保存一个选项以用于自定义任务时，**inputProperty** 的值与此值相对应，并显示在自定义任务建模中。 例如，值 2015。 ■ 2015 ■ 2016 ■ 2017 ■ 2018 ■ 2019 ■ 2020	否

自定义任务 YAML 输入键	说明	必需
options	使用 **optionKey** 和 **optionValue** 接受对象数组。 ■ **optionKey**。传播到任务的代码部分的值。 ■ **optionValue**。在用户界面中显示选项的字符串。 仅某些输入属性类型提供此支持。 选项： **optionKey**：key1。选择并保存以用于自定义任务后，此 inputProperty 的值将对应于代码部分中的 **key1**。 **optionValue**：'Label for 1'。**key1** 在用户界面中的显示值，不会显示在自定义任务的任何其他位置。 **optionKey**：key2 **optionValue**：'Label for 2' **optionKey**：key3 **optionValue**：'Label for 3'	否
minimum	接受作为此输入属性的有效最小值的数字。仅数字类型的输入属性提供此支持。	否
maximum	接受作为输入属性的有效最大值的数字。仅数字类型的输入属性提供此支持。	否

表 5-6. 支持用于自定义脚本的数据类型和元数据

支持的数据类型	支持用于输入的元数据
■ 字符串 ■ 文本 ■ List：显示为任意类型的列表 ■ Map：显示为 map[string]any ■ Secure：显示为密码文本框，在保存自定义任务时进行加密 ■ 数字 ■ Boolean：显示为文本框 ■ URL：与 string 相同，需要进行附加验证 ■ 选择和单选按钮	■ type：String 或 Text 中的一种... ■ default：默认值 ■ options：要与选择或单选按钮一起使用的选项的列表或映射 ■ min：最小值或最小大小 ■ max：最大值或最大大小 ■ title：文本框的详细名称 ■ placeHolder：UI 占位符 ■ description：将成为工具提示

例如：

```
inputProperties:
        - name: message
          type: text
          title: Message
          placeHolder: Message for Slack Channel
          defaultValue: Hello Slack
          bindable: true
          labelInfo: true
          labelMessage: This message is posted to the Slack channel link provided in the
code
```

3 在脚本中声明输出属性。

脚本将从业务逻辑（脚本的 `code:` 节）捕获输出属性，您可以在该节中声明输出的上下文。

当管道运行时，您可以输入任务输出的响应代码。例如，**`200`**。

vRealize Automation Code Stream 针对每个 **outputProperty** 支持的键。

键	说明
type	当前包括 **`label`** 的单个值。
name	自定义集成 YAML 的代码块发出的键。
title	用户界面中显示 **outputProperty** 的标签。

例如：

```
outputProperties:
  - name: statusCode
    type: label
    title: Status Code
```

4 要与自定义脚本的输入和输出交互，请使用 **`context`** 获取输入属性或设置输出属性。

对于输入属性：`(context.getInput("key"))`

对于输出属性：`(context.setOutput("key", "value"))`

对于 Node.js：

```
var context = require("./context.js")
var message = context.getInput("message");
//Your Business logic
context.setOutput("statusCode", 200);
```

对于 Python：

```
from context import getInput, setOutput
message = getInput('message')
//Your Business logic
setOutput('statusCode', '200')
```

对于 Shell：

```
# Input, Output properties are environment variables
echo ${message} # Prints the input message
//Your Business logic
export statusCode=200 # Sets output property statusCode
```

5 在 `code:` 节中，声明自定义集成的所有业务逻辑。

例如，对于 Node.js 运行时环境：

```
code: |
    var https = require('https');
    var context = require("./context.js")

    //Get the entered message from task config page and assign it to message var
    var message = context.getInput("message");
    var slackPayload = JSON.stringify(
        {
            text: message
        });

    const options = {
        hostname: 'hooks.slack.com',
        port: 443,
        path: '/YOUR_SLACK_WEBHOOK_PATH',
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Content-Length': Buffer.byteLength(slackPayload)
        }
    };

    // Makes a https request and sets the output with statusCode which
    // will be displayed in task result page after execution
    const req = https.request(options, (res) => {
        context.setOutput("statusCode", res.statusCode);
    });

    req.on('error', (e) => {
        console.error(e);
    });
    req.write(slackPayload);
    req.end();
```

6 对自定义集成脚本进行版本控制并发布之前，请下载适用于 Python 或 Node.js 的上下文文件，并测试脚本中包含的业务逻辑。

a 将指针放在脚本中，然后单击画布顶部的上下文文件按钮。例如，如果脚本采用 Python，则单击 **CONTEXT.PY**。

b 修改并保存文件。

c 在开发系统上，借助上下文文件运行并测试自定义脚本。

7 将某个版本应用到自定义集成脚本。

a 单击**版本**。

b 输入版本信息。

c 单击**发行版本**，以在自定义任务中选择脚本。

d 要创建版本，请单击**创建**。

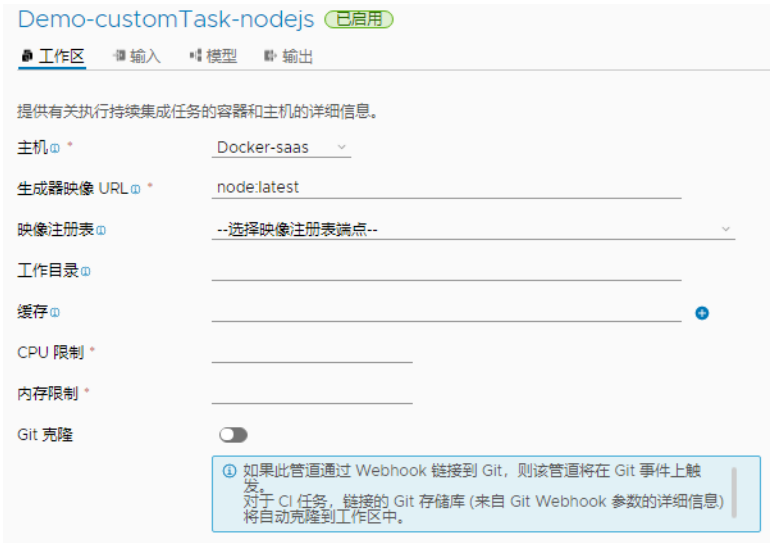
8 要保存脚本，请单击**保存**。

9 在管道中，配置工作区。

此示例使用 Docker 工作区。

a 单击**工作区**选项卡。

b 选择 Docker 主机和生成器映像 URL。



10 向管道添加自定义任务并配置该自定义任务。

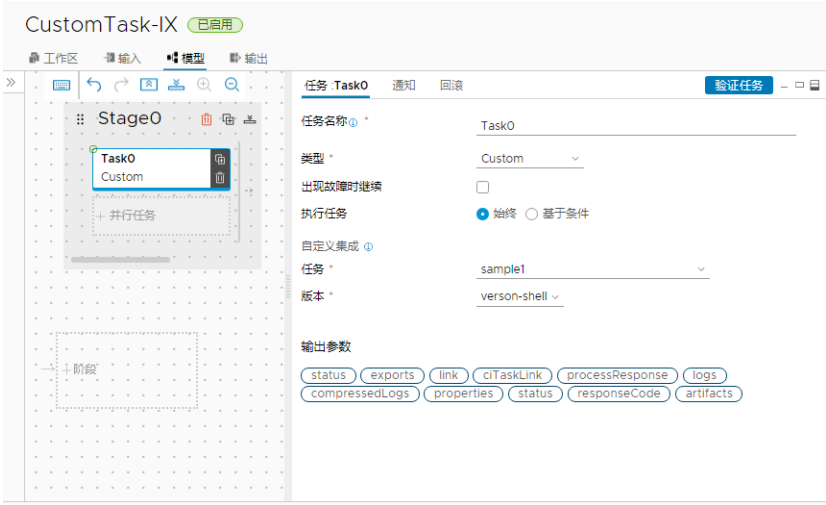
a 单击**模型**选项卡。

b 添加任务，为类型选择**自定义**，然后输入相关的名称。

c 选择您的自定义集成脚本和版本。

d 要在 Slack 中显示自定义消息，请输入消息文本。

输入的任何文本将替代自定义集成脚本中的 `defaultValue`。例如：



11 保存并启用管道。

a 单击**保存**。

b 在“管道”选项卡中，单击**启用管道**以使圆圈移动到右侧。

12 运行管道。

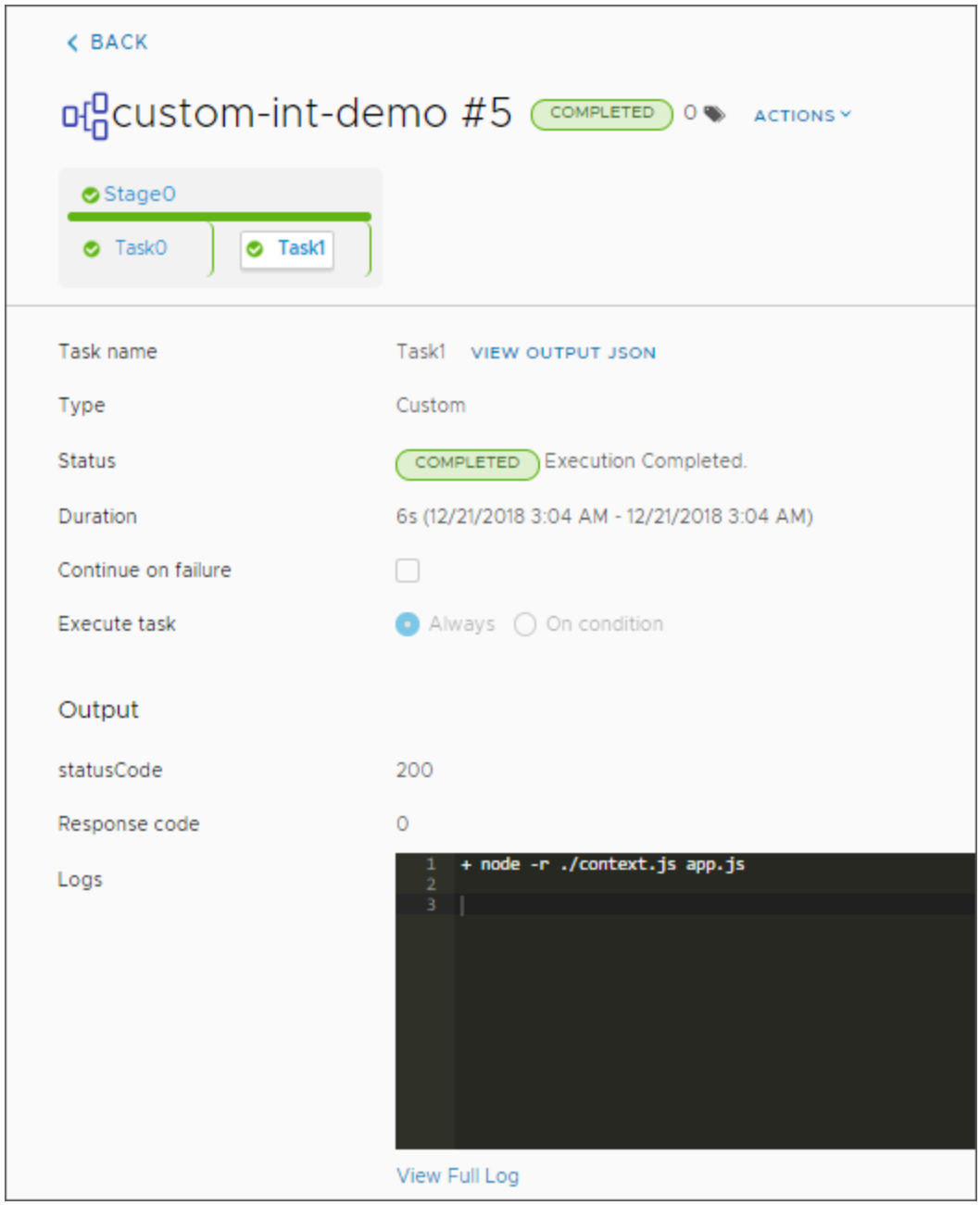
a 单击**运行**。

b 查看管道执行。

c 确认输出包含预期的状态代码、响应代码、状态和已声明的输出。

您已将 **statusCode** 定义为输出属性。例如，**statusCode** 的值为 200 可能表示 Slack 发布成功，而 **responseCode** 的值为 0 可能表示脚本已成功而未发生错误。

d 要确认执行日志中的输出，请单击**执行**，单击管道的链接，单击该任务，然后查找已记录的数据。例如：



13 如果发生错误，请解决问题，然后重新运行管道。

如果基础映像中缺少某个文件或模块，您必须创建包含缺少的文件的另一个基础映像。然后，提供 Docker 文件，并通过管道推送映像。

结果

恭喜！您已创建一个自定义集成脚本，该脚本将 vRealize Automation Code Stream 连接到 Slack 实例并向 Slack 通道发布消息。

后续步骤

继续创建自定义集成以支持在管道中使用自定义任务，以便扩展 vRealize Automation Code Stream 在自动执行软件发布生命周期方面的功能。

如何在下一个任务中使用云模板任务的资源属性

在 vRealize Automation Code Stream 中使用云模板任务时，一个常见的问题是如何在管道的后续任务中使用该任务的输出。要使用云模板任务的输出（例如云计算机），必须了解如何在云模板任务的部署详细信息中查找资源属性，以及云计算机的 IP 地址。

例如，VMware 云模板的部署详细信息包括云计算机资源及其 IP 地址。在管道中，可以使用云计算机和 IP 地址作为变量，将云模板任务绑定到 REST 任务。

您用于查找云计算机 IP 地址的方法并不常用，因为 VMware 云模板的部署必须在部署详细信息可用之前完成。然后，可以使用 VMware 云模板部署中的资源绑定管道任务。

- 管道的云模板任务中显示的资源属性在 vRealize Automation Cloud Assembly 的 VMware 云模板中进行定义。
- 您可能不知道该云模板的部署何时完成。
- 部署完成后， vRealize Automation Code Stream 中的云模板任务只能显示 VMware 云模板的输出属性。

部署应用程序和调用各种 API 时，此示例尤为有用。例如，如果使用的云模板任务调用 VMware 云模板，且该云模板使用 REST API 部署 Wordpress 应用程序，则可以在部署详细信息中找到已部署计算机的 IP 地址，并使用 API 对其进行测试。

云模板任务可显示即输即显自动填充详细信息，从而支持您使用变量绑定。如何绑定变量由您决定。

此示例展示了如何执行以下操作：

- 在已运行并成功的管道中查找云模板任务的部署详细信息和资源属性。
- 在部署详细信息的资源部分中查找云计算机 IP 地址。
- 在管道中的云模板任务之后添加 REST 任务。
- 通过在 REST 任务的 URL 中使用云计算机 IP 地址，将云模板任务绑定到 REST 任务。
- 运行管道并监视从云模板任务到 REST 任务的绑定工作。

前提条件

- 确认您具有已进行版本控制的工作 VMware 云模板。
- 确认已在 vRealize Automation Cloud Assembly 中成功部署 VMware 云模板。
- 确认您的管道包括使用该 VMware 云模板的云模板任务。

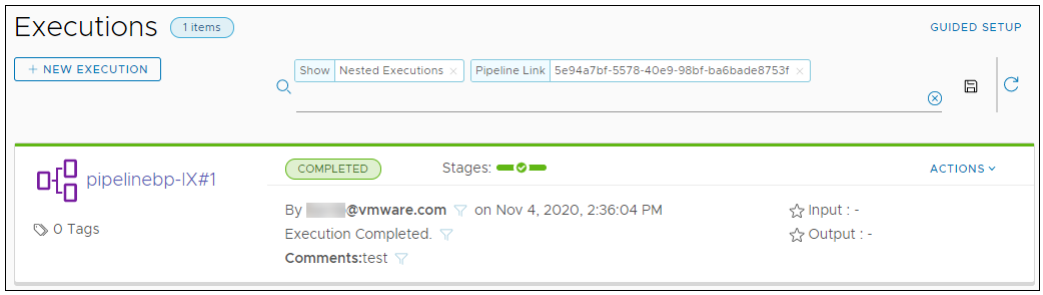
- 确认您的管道已运行并成功。

步骤

1 在管道中，在云模板任务部署详细信息的资源部分中找到云计算机的 IP 地址。

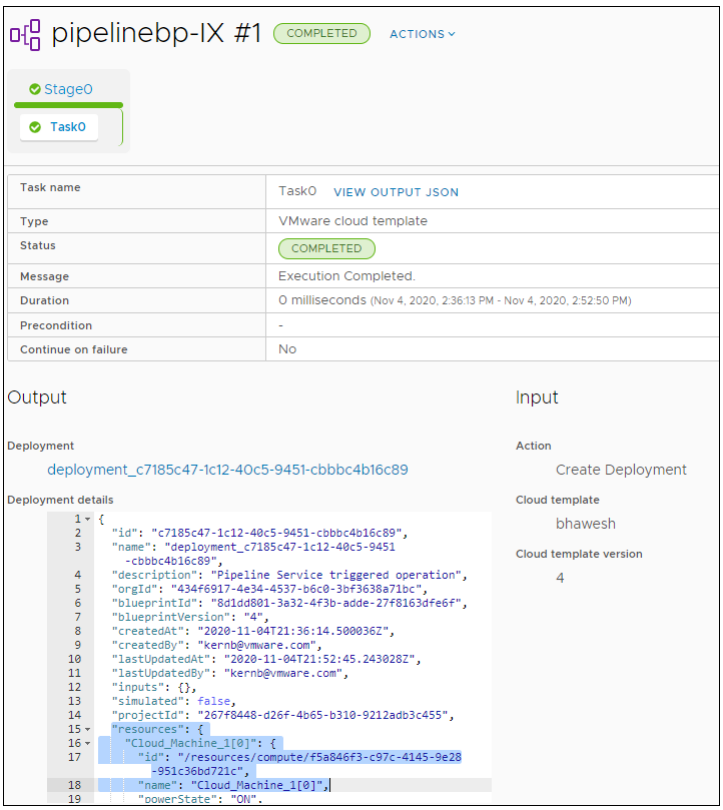
   a 单击**操作 > 查看执行**。

   b 在成功的管道运行中，单击指向管道执行的链接。



   c 在管道名称下，单击指向**任务**的链接。

d 在“输出”区域中，找到部署详细信息。

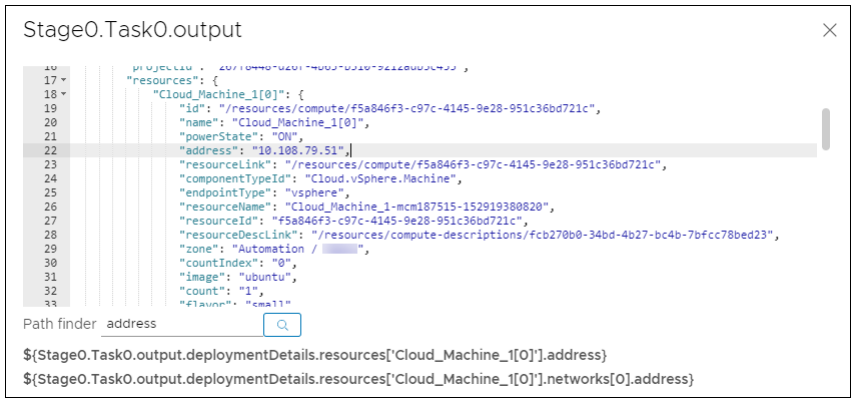


e 在部署详细信息的资源部分中，找到云计算机名称。

将在 REST 任务的 URL 中包括云计算机名称的语法。

f 要查找云模板任务的输出属性的绑定表达式，请单击**查看输出 JSON**，搜索地址属性，然后找到云计算机 IP 地址。

绑定表达式显示在 JSON 输出中的属性和搜索图标下方。



地址资源属性显示云计算机 IP 地址。例如：

```
"resources": {
        "Cloud_Machine_1[0]": {
            "name": "Cloud_Machine_1[0]",
            "powerState": "ON",
            "address": "10.108.79.51",
            "resourceName": "Cloud_Machine_1-mcm187515-152919380820"
```

2 返回管道模型，然后在 REST 任务中输入 URL。

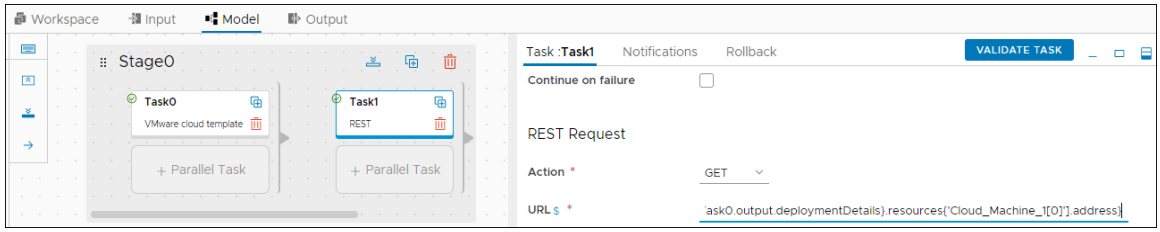
a 单击**操作 > 查看管道**。

b 单击 REST 任务。

c 在“REST 请求”的“URL”区域中，输入 **$**，选择 **Stage**、**Task**、**output**、**deploymentDetails**，然后输入 **resources**。

可以使用通过自动填充实现即输即显的功能，直到必须输入 **resources**。

d 输入部署详细信息中云计算机资源的其余信息，如下所示：
{'Cloud_Machine_1[0]'].address}



对于云计算机条目，必须使用所示的方括号表示形式。

完整的 URL 格式为：`$ {Stage0.Task0.output.deploymentDetails.resources{'Cloud_Machine_1[0]'].address}`

3 运行管道并监视 REST 任务是否使用云模板任务输出中的云计算机和 IP 地址作为要测试的 URL。

结果

恭喜！您在云模板任务的部署详细信息和 JSON 输出中找到了云计算机名称和 IP 地址，并使用这些信息将云模板任务输出绑定到管道中的 REST 任务 URL 输入。

后续步骤

继续了解如何使用绑定变量将云模板任务中的资源与管道中的其他任务绑定。

如何使用 REST API 将 vRealize Automation Code Stream 与其他应用程序集成

vRealize Automation Code Stream 提供一个 REST 插件，借助该插件可以将 vRealize Automation Code Stream 与使用 REST API 的其他应用程序集成，以便持续开发并交付必须相互交互的软件应用程序。该 REST 插件调用一个 API，后者在 vRealize Automation Code Stream 与其他应用程序之间发送和接收信息。

借助该 REST 插件，您可以：

- 将基于 REST API 的外部系统集成到 vRealize Automation Code Stream 管道中。
- 将 vRealize Automation Code Stream 管道集成到外部系统的流程中。

该 REST 插件可与任何 REST API 配合工作，并且支持使用 GET、POST、PUT、PATCH 和 DELETE 方法在 vRealize Automation Code Stream 与其他应用程序之间发送或接收信息。

表 5-7. 准备管道以通过 REST API 进行通信

执行的操作	结果
向管道添加 REST 任务。	该 REST 任务在应用程序之间传递信息，并且提供管道阶段中某个连续任务的状态信息。
在 REST 任务中，选择 REST 操作并包含 URL。	当管道运行时，管道任务将调用该 URL。 对于 POST、PUT 和 PATCH 操作，必须包含负载。在负载中，可以在管道运行时将管道与任务属性相绑定。
考虑此示例。	REST 插件的示例用法： 您可以添加 REST 任务以在生成的 Git 提交上创建一个标记，并让该任务发布请求以从存储库获取签入 ID。该任务可以向存储库发送负载并为生成创建一个标记，而存储库可以返回带有标记的响应。

与使用 REST 插件来调用 API 一样，可以在管道中包含 Poll 任务以调用并轮询某个 REST API，直到它完成并且管道任务满足退出条件为止。

还可以使用 REST API 导入和导出管道，以及使用示例脚本运行管道。

以下过程将获取简单 URL。

步骤

1 要创建管道，请单击**管道 > 新建管道 > 空白画布**。

2 在管道阶段中，单击 **+ 连续任务**。

3 在任务窗格中，添加 REST 任务：

   a 输入任务的名称。

   b 在“类型”下拉菜单中，选择 **REST**。

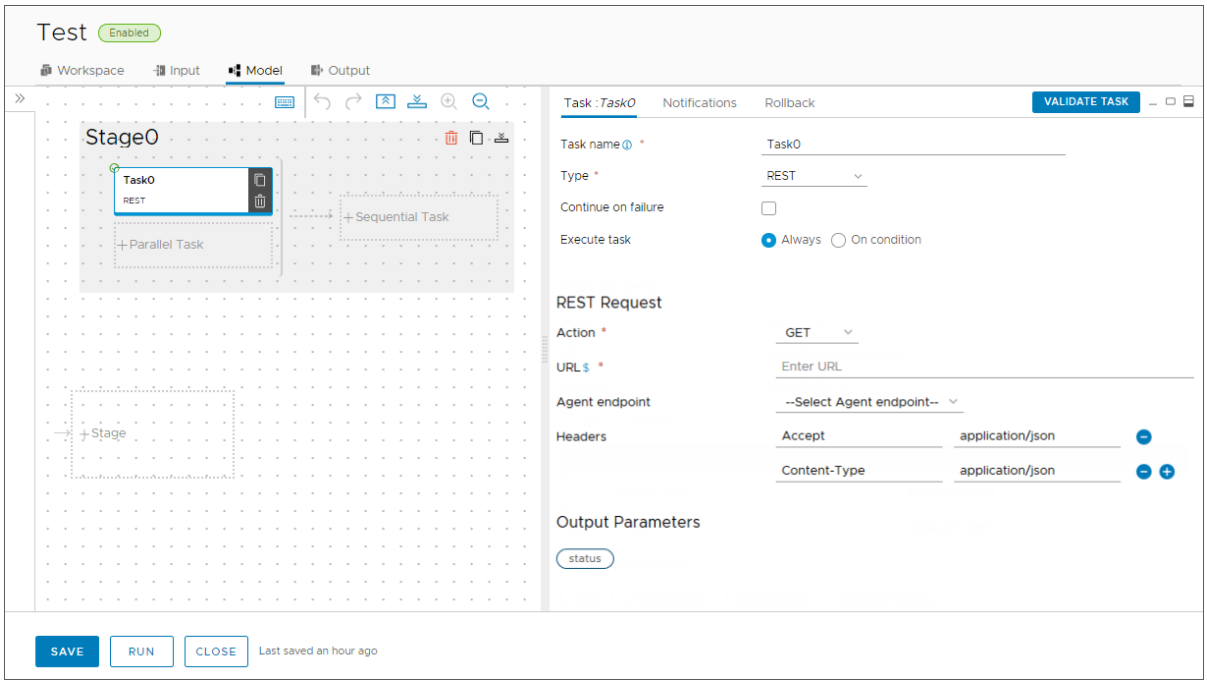
   c 在“REST 请求”区域，选择 **GET**。

   要让 REST 任务从其他应用程序请求数据，请选择 GET 方法。要向其他应用程序发送数据，请选择 POST 方法。

   d 输入用于标识 REST API 端点的 URL。例如，`https://www.google.com`。

   要使 REST 任务从其他应用程序导入数据，可以包含负载变量。例如，对于导入操作，可以输入 `${Stage0.export.responseBody}`。如果响应数据大小超过 5 MB，REST 任务可能会失败。

   e 要为任务提供授权，请单击**添加标头**，然后输入标头键和值。



4 要保存管道，请单击**保存**。

5 在“管道”选项卡中，单击**启用管道**。

6 单击**保存**，然后单击**关闭**。

7 单击**运行**。

8 要监视管道的运行状况，请单击**执行**。

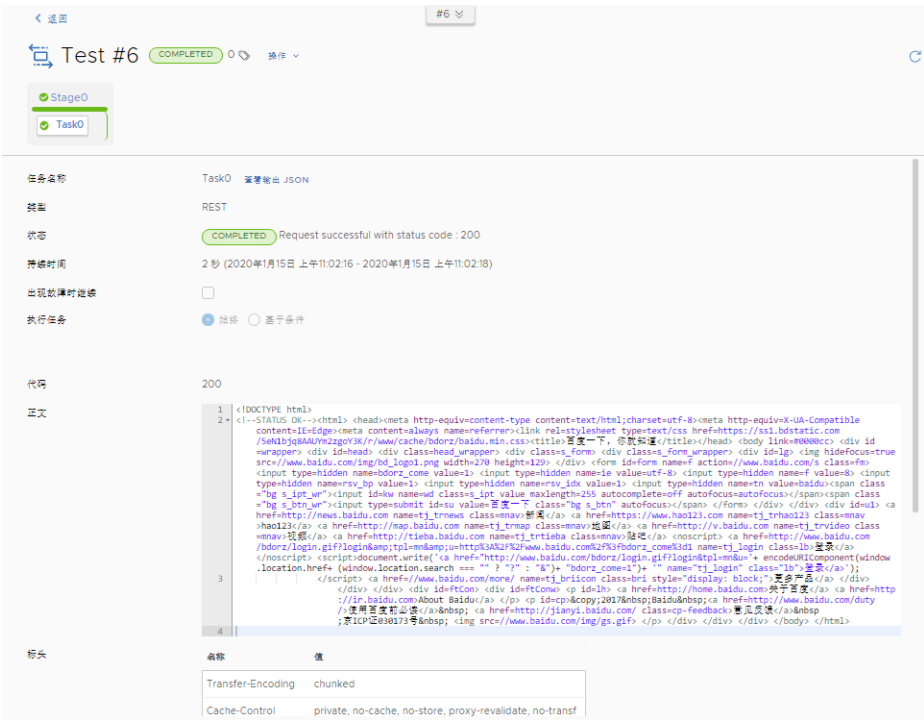
9 要验证 REST 任务是否返回预期信息，请检查管道执行和任务结果。

a 管道完成后，要确认其他应用程序返回了所请求的数据，请单击管道执行的链接。

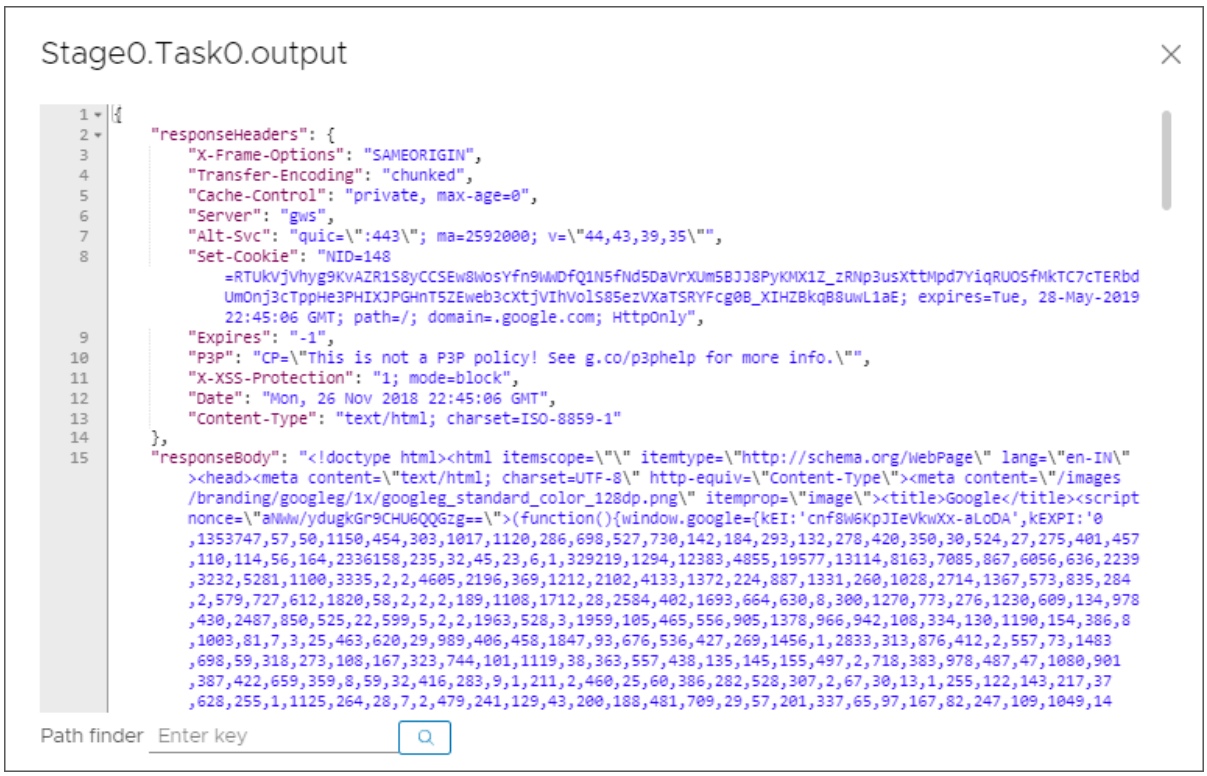
b 单击管道中的 REST 任务。

c 在管道执行中，单击相关任务，查看任务详细信息，并验证 REST 任务是否返回了预期结果。

任务详细信息显示响应代码、正文、标头键和值。



10 要查看 JSON 输出，请单击**查看输出 JSON**。




```
{
    "responseHeaders": {
        "X-Frame-Options": "SAMEORIGIN",
        "Transfer-Encoding": "chunked",
        "Cache-Control": "private, max-age=0",
        "Server": "gws",
        "Alt-Svc": "quic=\":443\"; ma=2592000; v=\"44,43,39,35\"",
        "Set-Cookie": "NID=148
            =RTUkVjVhyg9KvAZR1S8yCCSEw8WosYfn9WWDfQ1N5fNd5DaVrXUm5BJJ8PyKMX1Z_zRNp3usXttMpd7YiqRUOSfMkTC7cTERbd
            UmOnj3cTppHe3PHIXJPGHnT5ZEweb3cXtjVIhVolS85ezVXaTSRYFcg0B_XIHZBkqB8uwL1aE; expires=Tue, 28-May-2019
            22:45:06 GMT; path=/; domain=.google.com; HttpOnly",
        "Expires": "-1",
        "P3P": "CP=\"This is not a P3P policy! See g.co/p3phelp for more info.\"",
        "X-XSS-Protection": "1; mode=block",
        "Date": "Mon, 26 Nov 2018 22:45:06 GMT",
        "Content-Type": "text/html; charset=ISO-8859-1"
    },
    "responseBody": "<!doctype html><html itemscope=\"\" itemtype=\"http://schema.org/WebPage\" lang=\"en-IN\"
        ><head><meta content=\"text/html; charset=UTF-8\" http-equiv=\"Content-Type\"><meta content=\"/images
        /branding/googleg/1x/googleg_standard_color_128dp.png\" itemprop=\"image\"><title>Google</title><script
        nonce=\"aNWw/ydugkGr9CHU6QQGzg==\">(function(){window.google={kEI:'cnf8W6KpJIeVkwXx-aLoDA',kEXPI:'0
        ,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457
        ,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,6056,636,2239
        ,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284
        ,2,579,727,612,1820,58,2,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
        ,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8
        ,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483
        ,698,59,318,273,108,167,323,744,101,1119,38,363,557,438,135,145,155,497,2,718,383,978,487,47,1080,901
        ,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37
        ,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,247,109,1049,14
```


结果

恭喜！您已配置了一个 REST 任务，该任务调用 REST API 并使用 REST 插件在 vRealize Automation Code Stream 与其他应用程序之间发送信息。

后续步骤

继续在管道中使用 REST 任务运行命令并将 vRealize Automation Code Stream 与其他应用程序集成，以便开发并交付软件应用程序。可以考虑使用 Poll 任务对 API 进行轮询直到完成，并且管道任务满足退出条件。

如何在 vRealize Automation Code Stream 中利用代码形式的管道

作为 DevOps 管理员或开发人员，您可能希望在 vRealize Automation Code Stream 中使用 YAML 代码（而不是用户界面）创建管道。创建代码形式的管道时，可以使用任何编辑器并在管道代码中插入注释。

在管道代码中，可以引用外部配置，如环境变量和安全凭据。如果更新在管道代码中使用的变量，则在更新时可以不必更新管道代码。

可以将管道 YAML 代码用作模板以克隆和创建其他管道，以及与其他管道共享模板。

可以将管道代码模板存储在源代码管理存储库中，以便对其进行版本控制并跟踪更新。通过使用源代码管理系统，可以轻松备份管道代码，并根据需要进行还原。

前提条件

- 确认您具有代码编辑器。
- 如果计划将管道代码存储在源代码管理存储库中，请确认您可以访问工作实例。

步骤

1 在代码编辑器中，创建一个文件。

2 复制并粘贴示例管道代码，然后对其进行更新以反映您的特定管道需求。

3 要在管道代码中包含端点，请复制并粘贴示例端点代码，然后对其进行更新以反映您的端点。

   在管道工作区中使用 Kubernetes 端点时， vRealize Automation Code Stream 会创建必要的 Kubernetes 资源（如 ConfigMap、密钥和 Pod）以运行持续集成 (CI) 任务或自定义任务。 vRealize Automation Code Stream 使用 NodePort 与容器进行通信。

   vRealize Automation Code Stream 管道工作区支持使用 Docker 和 Kubernetes 执行持续集成任务和自定义任务。

   有关配置工作区的详细信息，请参见配置管道工作区。

4 保存代码。

5 要存储管道代码并进行版本控制，请将代码签入到源代码管理存储库中。

6 创建持续集成和交付管道时，必须导入 Kubernetes YAML 文件。

   要导入 Kubernetes YAML 文件，请在智能管道模板的“持续交付”区域中选择该文件，然后单击**处理**。或者，使用 API。

结果

通过使用代码示例，您创建了表示管道和端点的 YAML 代码。

示例： 管道和端点的示例 YAML 代码

此示例 YAML 代码包含一些区域，用于表示管道中 vRealize Automation Code Stream 本地构建、阶段、任务、通知等的工作区。

有关支持插件的代码示例，请参见第 6 章 将 vRealize Automation Code Stream 连接到端点

```
---
kind: PIPELINE
name: myPipelineName
tags:
  - tag1
  - tag2


# Ready for execution
enabled: false

#Max number of concurrent executions
concurrency: 10
```

```
#Input Properties
input:
  input1: '30'
  input2: 'Hello'

#Output Properties
output:
  BuildNo: '${Dev.task1.buildNo}'
  Image: '${Dev.task1.image}'

#Workspace Definition
ciWorkspace:
  image: docker:maven-latest
  path: /var/tmp
  endpoint: my-k8s
  cache:
    - ~/.m2

# Starred Properties
starred:
  input: input1
  output: output1

# Stages in order of execution
stageOrder:
  - Dev
  - QA
  - Prod

# Task Definition Section
stages:
  Dev:
    taskOrder:
      - Task1, Task6
      - Task2 Long, Task Long Long
      - Task5
    tasks:
      Task1:
        type: jenkins
        ignoreFailure: false
        preCondition: ''
        endpoints:
          jenkinsServer: myJenkins
        input:
          job: Add Two Numbers
          parameters:
            number1: 10
            number2: 20
      Task2:
        type: blah
        # repeats like Task1 above
  QA:
    taskOrder:
      - TaskA
      - TaskB
```

```
    tasks:
      TaskA:
        type: ssh
        ignoreFailure: false
        preCondition: ''
        input:
          host: x.y.z.w
          username: abcd
          password: ${var.mypassword}
          script: >
            echo "Hello, remote server"
      TaskB:
        type: blah
        # repeats like TaskA above

# Notificatons Section
notifications:
  email:
    - stage: Dev #optional ; if not found - use pipeline scope
      task: Task1 #optional; if not found use stage scope
      event: SUCCESS
      endpoint: default
      to:
         - user@yourcompany.com
         - abc@yourcompany.com
      subject: 'Pipeline ${name} has completed successfully'
      body: 'Pipeline ${name} has completed successfully'

  jira:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskA #optional; if not found use stage scope
      event: FAILURE
      endpoint: myJiraServer
      issuetype: Bug
      project: Test
      assignee: abc
      summary: 'Pipeline ${name} has failed'
      description: |-
        Pipeline ${name} has failed
        Reason - ${resultsText}
  webhook:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskB #optional; if not found use stage scope
      event: FAILURE
      agent: my-remote-agent
      url: 'http://www.abc.com'
      headers: #requestHeaders: '{"build_no":"123","header2":"456"}'
          Content-Type: application/json
          Accept: application/json
      payload: |-
        Pipeline ${name} has failed
        Reason - ${resultsJson}
---
```

此 YAML 代码表示一个示例 Jenkins 端点。

```
---
name: My-Jenkins
tags:
- My-Jenkins
- Jenkins
kind: ENDPOINT
properties:
  offline: true
  pollInterval: 15.0
  retryWaitSeconds: 60.0
  retryCount: 5.0
  url: http://urlname.yourcompany.com:8080
description: Jenkins test server
type: your.jenkins:JenkinsServer
isLocked: false
---
```

此 YAML 代码表示一个示例 Kubernetes 端点。

```
---
name: my-k8s
tags: [
  ]
kind: ENDPOINT
properties:
  kubernetesURL: https://urlname.examplelocation.amazonaws.com
  userName: admin
  password: encryptedpassword
description: ''
type: kubernetes:KubernetesServer
isLocked: false
---
```

后续步骤

运行管道，并根据需要进行任何调整。请参见如何运行管道和查看结果。

将 vRealize Automation Code Stream 连接到端点

6

vRealize Automation Code Stream 通过插件与开发工具集成。支持的插件包括 Jenkins、Bamboo、vRealize Operations、Bugzilla、Team Foundation Server、Git 等。

此外，也可以开发自己的插件，以便将 vRealize Automation Code Stream 与其他开发应用程序相集成。

要将 vRealize Automation Code Stream 与 Jira 相集成，不需要外部插件，因为 vRealize Automation Code Stream 将 Jira 票证创建功能作为通知类型包括在内。要创建有关管道状态的 Jira 票证，必须添加 Jira 端点。

本章讨论了以下主题：

- 什么是 vRealize Automation Code Stream 中的端点
- 如何将 vRealize Automation Code Stream 与 Jenkins 集成
- 如何将 vRealize Automation Code Stream 与 Git 集成
- 如何将 vRealize Automation Code Stream 与 Gerrit 集成
- 如何将 vRealize Automation Code Stream 与 vRealize Orchestrator 集成

什么是 vRealize Automation Code Stream 中的端点

端点是 DevOps 应用程序的实例，它连接到 vRealize Automation Code Stream 并为管道运行提供数据，例如数据源、存储库或通知系统。

您在 vRealize Automation Code Stream 中的角色决定了您使用端点的方式。

- 管理员和开发人员可以创建、更新、删除和查看端点。
- 管理员可以将端点标记为受限制，并运行使用受限制端点的管道。
- 具有查看者角色的用户可以查看端点，但不能创建、更新或删除端点。

有关详细信息，请参见如何管理 vRealize Automation Code Stream 中的用户访问和批准。

要将 vRealize Automation Code Stream 连接到端点，您可以在管道中添加任务，并对其进行配置，使其与端点进行通信。要验证 vRealize Automation Code Stream 是否能连接到端点，请单击**验证**。然后，当您运行管道时，管道任务将连接到端点以运行任务。

有关使用这些端点的任务类型的信息，请参见 vRealize Automation Code Stream 中提供哪些类型的任务。

表 6-1. vRealize Automation Code Stream 支持的端点

端点	功能	支持的版本	要求
Bamboo	创建生成计划。	6.9.*	
Docker	本地构建可以使用 Docker 主机进行部署。		当管道包含 Docker Hub 中的映像时，在运行管道之前必须确保映像中已嵌入了 `cURL` 或 `wget`。当管道运行时，vRealize Automation Code Stream 会下载使用 `cURL` 或 `wget` 运行命令的二进制文件。
Docker 注册表	注册容器映像，以便 Docker 生成主机可以提取映像。	2.7.1	
Gerrit	连接到 Gerrit 服务器以进行检查和触发	2.14.*	
Git	在开发人员更新代码并将其签入存储库时触发管道。	Git Hub Enterprise 2.1.8 Git Lab Enterprise 11.9.12-ee	
Jenkins	生成代码工件。	1.6.* 和 2.*	
Jira	创建管道任务失败时的 Jira 票证。	8.3.*	
Kubernetes	自动执行部署、扩展和管理容器化应用程序的步骤。	支持的所有版本（对于 vRealize Automation Cloud Assembly 8.4 及更高版本） 1.18（对于 vRealize Automation Cloud Assembly 8.3 及更低版本）	在管道工作区中使用 Kubernetes 端点时， vRealize Automation Code Stream 会创建必要的 Kubernetes 资源（如 ConfigMap、密钥和 Pod）以运行持续集成 (CI) 任务或自定义任务。vRealize Automation Code Stream 使用 NodePort 与容器进行通信。 有关配置工作区的详细信息，请参见配置管道工作区。
PowerShell	创建在 Windows 或 Linux 计算机上运行 PowerShell 脚本的任务。	4 和 5	
SSH	创建在 Windows 或 Linux 计算机上运行 SSH 脚本的任务。	7.0	
TFS，即 Team Foundation Server	管理源代码、自动生成、测试和相关活动。	2015 和 2017	
vRealize Orchestrator	安排和自动执行生成过程中的工作流。	7.* 和 8.*	

示例：GitHub 端点的 YAML 代码

此示例 YAML 代码定义了一个 GitHub 端点，您可以在 Git 任务中引用该端点。

```
---
name: github-k8s
tags: [
  ]
kind: ENDPOINT
properties:
  serverType: GitHub
  repoURL: https://github.com/autouser/testrepok8s
  branch: master
  userName: autouser
  password: encryptedpassword
  privateToken: ''
description: ''
type: scm:git
isLocked: false
---
```

如何将 vRealize Automation Code Stream 与 Jenkins 集成

vRealize Automation Code Stream 提供一个 Jenkins 插件，该插件可触发用于生成和测试源代码的 Jenkins 作业。Jenkins 插件运行测试用例，并且可以使用自定义脚本。

要在管道中运行 Jenkins 作业，需要使用 Jenkins 服务器，并在 vRealize Automation Code Stream 中添加 Jenkins 端点。然后创建管道并向其添加 Jenkins 任务。

在 vRealize Automation Code Stream 中使用 Jenkins 任务和 Jenkins 端点时，可以创建支持 Jenkins 中多分支作业的管道。多分支作业包括 Git 存储库每个分支中的单个作业。在 vRealize Automation Code Stream 中创建支持多分支作业的管道时：

- Jenkins 任务可以运行位于 Jenkins 服务器上多个文件夹中的 Jenkins 作业。
- 可以在 Jenkins 任务配置中覆盖文件夹路径，以便使用不同的文件夹路径，这将覆盖在 vRealize Automation Code Stream 的 Jenkins 端点中定义的默认路径。
- vRealize Automation Code Stream 中的多分支管道检测 Git 存储库或 GitHub 存储库中类型为 `.groovy` 的 Jenkins 作业文件，并开始为在存储库中扫描到的每个分支创建作业。
- 可以使用在 Jenkins 任务配置中提供的路径覆盖在 Jenkins 端点中定义的默认路径，并运行与主 Jenkins 作业中的任何分支相关联的作业和管道。

前提条件

- 设置运行 1.561 或更高版本的 Jenkins 服务器。
- 验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。
- 确认作业在 Jenkins 服务器中存在，以便管道任务可以运行该作业。

步骤

1 添加并验证 Jenkins 端点。

   a 单击**端点 > 新建端点**。

   b 选择一个项目，并为端点类型选择 **Jenkins**。然后输入名称和描述。

   c 如果该端点是基础架构中的关键业务组件，请启用**标记为受限制**。

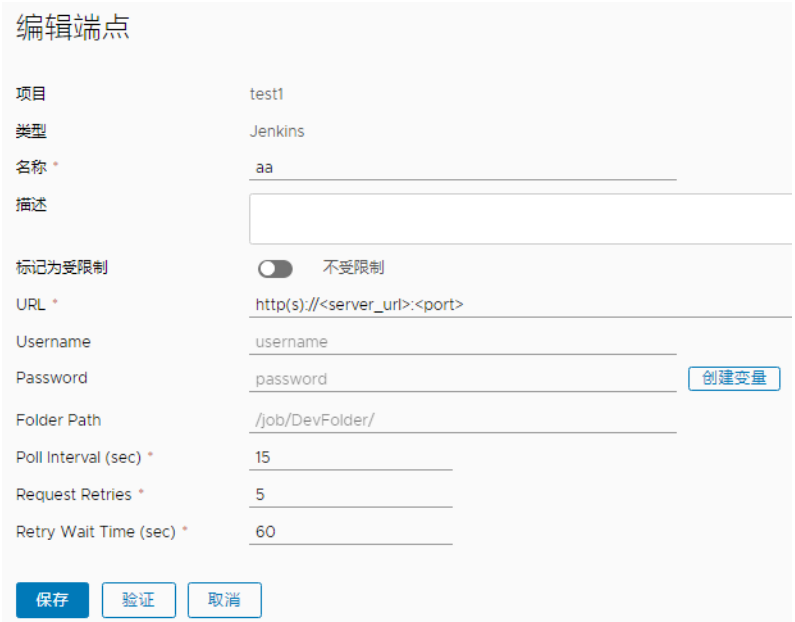
   d 输入 Jenkins 服务器的 URL。

e 输入用于登录到 Jenkins 服务器的用户名和密码。然后输入其余信息。

表 6-2. Jenkins 端点的其余信息

端点条目	说明
文件夹路径	用于对作业进行分组的文件夹的路径。Jenkins 可以运行该文件夹中的所有作业。可以创建子文件夹。例如： ■ `folder_1` 可以包括 `job_1` ■ `folder_1` 可以包括 `folder_2`，后者可以包括 `job_2` 为 `folder_1` 创建端点时，文件夹路径为 `job/folder_1`，并且端点仅列出 `job_1`。 要获取名为 `folder_2` 的子文件夹中的作业列表，必须创建另一个使用文件夹路径 `/job/folder_1/job/folder_2/` 的端点。
多分支 Jenkins 作业的文件夹路径	要支持多分支 Jenkins 作业，请在 Jenkins 任务中输入完整路径，其中包括 Jenkins 服务器 URL 和完整的作业路径。在 Jenkins 任务中包括文件夹路径时，该路径将替代 Jenkins 端点上显示的路径。对于 Jenkins 任务中的自定义文件夹路径， vRealize Automation Code Stream 仅显示该文件夹中的作业。 ■ 例如：`https://server.yourcompany.com/job/project` ■ 如果管道还必须触发主 Jenkins 作业，请使用：`https://server.yourcompany.com/job/project/job/main`
URL	Jenkins 服务器的主机 URL。以 `protocol://host:port` 格式输入 URL。例如：`http://192.10.121.13:8080`
轮询间隔	vRealize Automation Code Stream 向 Jenkins 服务器轮询更新的间隔持续时间。
请求重试计数	Jenkins 服务器的已调度生成请求的重试计数。
重试等待时间	重试 Jenkins 服务器的生成请求之前等待的秒数。

f 单击**验证**，然后验证端点是否连接到 vRealize Automation Code Stream。如果未连接，请更正任何错误，然后单击**保存**。



2 要生成代码，请创建管道，然后添加使用 Jenkins 端点的任务。

a 单击**管道 > 新建管道 > 空白画布**。

b 单击默认阶段。

c 在“任务”区域，输入任务的名称。

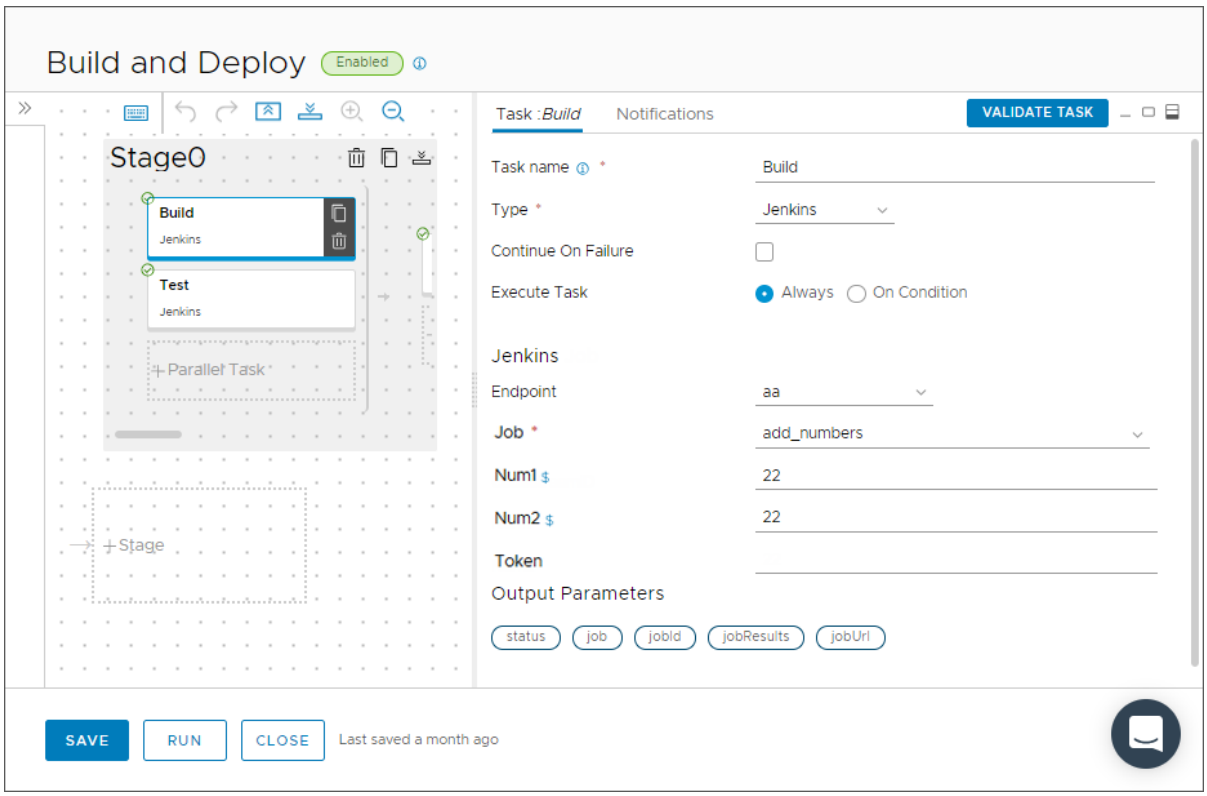
d 选择 **Jenkins** 作为任务类型。

e 选择您创建的 Jenkins 端点。

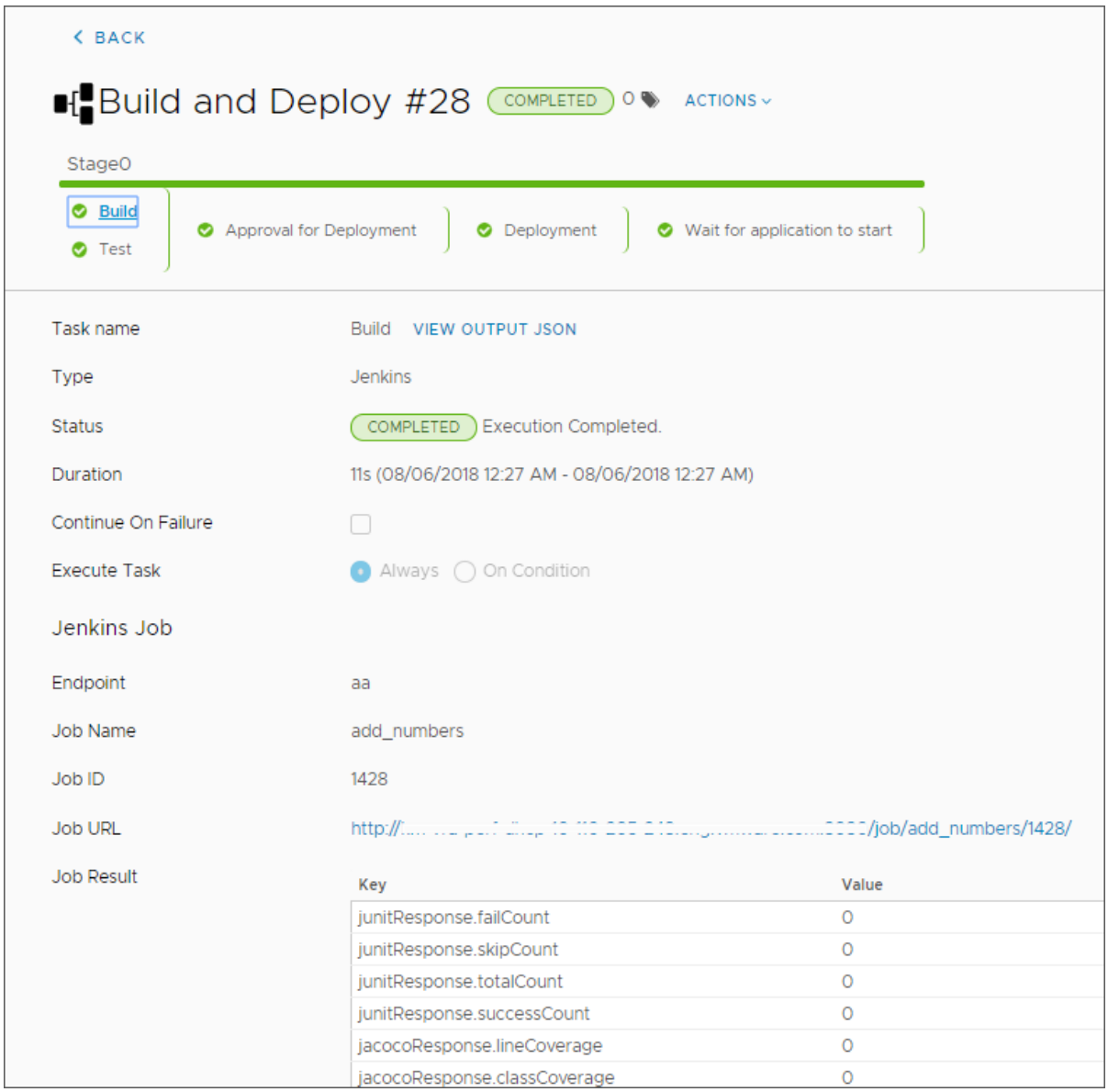
f 从下拉菜单中，从 Jenkins 服务器中选择管道将运行的作业。

g 输入作业的参数。

h 输入 Jenkins 作业的身份验证令牌。

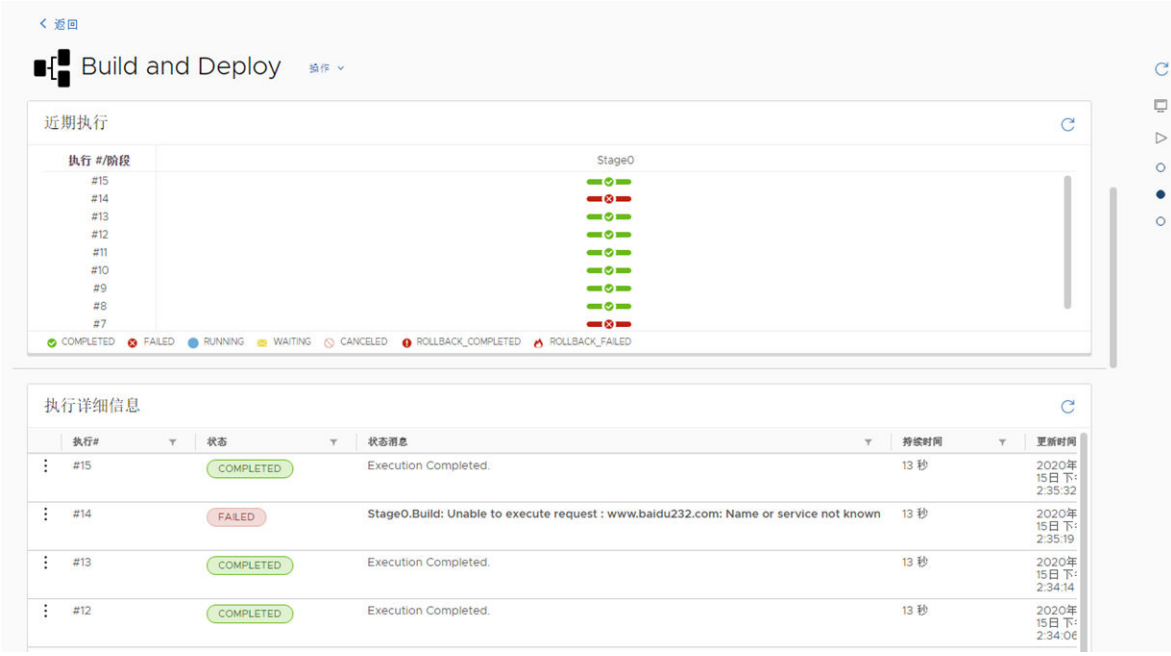


3 启用并运行管道，然后查看管道执行。



4 在管道仪表板中查看执行详细信息和状态。

可以确定任何失败以及失败原因。还可以查看有关管道执行持续时间、完成和失败的趋势。



结果

恭喜！通过添加端点，创建管道并配置用于生成代码的 Jenkins 任务，您已将 vRealize Automation Code Stream 与 Jenkins 集成。

示例： Jenkins 生成任务的示例 YAML

对于此示例中使用的 Jenkins 生成任务类型，YAML 类似于以下代码，并且通知已打开：

```
test:
  type: Jenkins
  endpoints:
    jenkinsServer: jenkins
  input:
    job: Add two numbers
    parameters:
      Num1: '23'
      Num2: '23'
```

后续步骤

有关更多信息，请参见其他节。请参见第 6 章 将 vRealize Automation Code Stream 连接到端点。

如何将 vRealize Automation Code Stream 与 Git 集成

如果您的 GitHub、GitLab 或 Bitbucket 存储库中发生了代码更改， vRealize Automation Code Stream 会提供一种触发管道的方式。Git 触发器在您要监控的存储库的分支上使用 Git 端点。 vRealize Automation Code Stream 通过 Webhook 连接到 Git 端点。

要在 vRealize Automation Code Stream 中定义 Git 端点，请选择一个项目，然后输入端点所在的 Git 存储库的分支。该项目将管道与端点和其他相关对象进行分组。当您在 Webhook 定义中选择项目时，您可以选择要触发的端点和管道。

注 如果定义包含您的端点的 Webhook，但稍后编辑了该端点，则无法在 Webhook 中更改端点详细信息。要更改端点详细信息，必须删除 Webhook，然后重新定义包含该端点的 Webhook。请参见如何使用 vRealize Automation Code Stream 中的 Git 触发器运行管道。

可以通过使用同一 Git 端点并在 Webhook 配置页面中为分支名称提供不同的值，为不同的分支创建多个 Webhook。要为同一 Git 存储库中的其他分支创建其他 Webhook，无需针对多个分支多次克隆 Git 端点。只需在 Webhook 中提供分支名称，即可重用 Git 端点。如果 Git Webhook 中的分支与端点中的分支相同，则无需在 Git Webhook 页面中提供分支名称。

前提条件

- 验证您能否访问计划连接的 GitHub、GitLab 或 Bitbucket 存储库。
- 验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是其成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。

步骤

1 定义 Git 端点。

   a 单击**端点 > 新建端点**。

   b 选择一个项目，并为端点类型选择 **Git**。然后输入名称和说明。

   c 如果该端点是基础架构中的关键业务组件，请启用**标记为受限制**。

   在管道中使用受限制的端点时，管理员可以运行管道，并且必须批准管道执行。如果端点或变量标记为受限制，则非管理员用户触发管道时，管道将在该任务处暂停，并等待管理员恢复。

   项目管理员可以启动包括受限制端点或变量的管道，前提是这些资源位于用户作为项目管理员的项目中。

   当非管理员用户尝试运行包含受限制资源的管道时，该管道将在使用受限制资源的任务处停止。然后，管理员必须恢复该管道。

   有关受限制资源以及包含**管理受限制的管道**权限的自定义角色的详细信息，请参见：

   - 如何管理 vRealize Automation Code Stream 中的用户访问和批准
   - 第 2 章 设置 vRealize Automation Code Stream 以对发布流程进行建模

   d 选择其中一个受支持的 Git 服务器类型。

e 输入存储库的 URL 并在路径中包含服务器的 API 网关。例如：

对于 GitHub，输入：**`https://api.github.com/vmware-example/repo-example`**

对于 BitBucket，输入：**`https://api.bitbucket.org/{user}/{repo name}`** 或 **`http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}}`**

f 输入端点所在的存储库中的分支。

g 选择身份验证类型，并输入 GitHub、GitLab 或 BitBucket 的用户名。然后输入与用户名匹配的专用令牌。

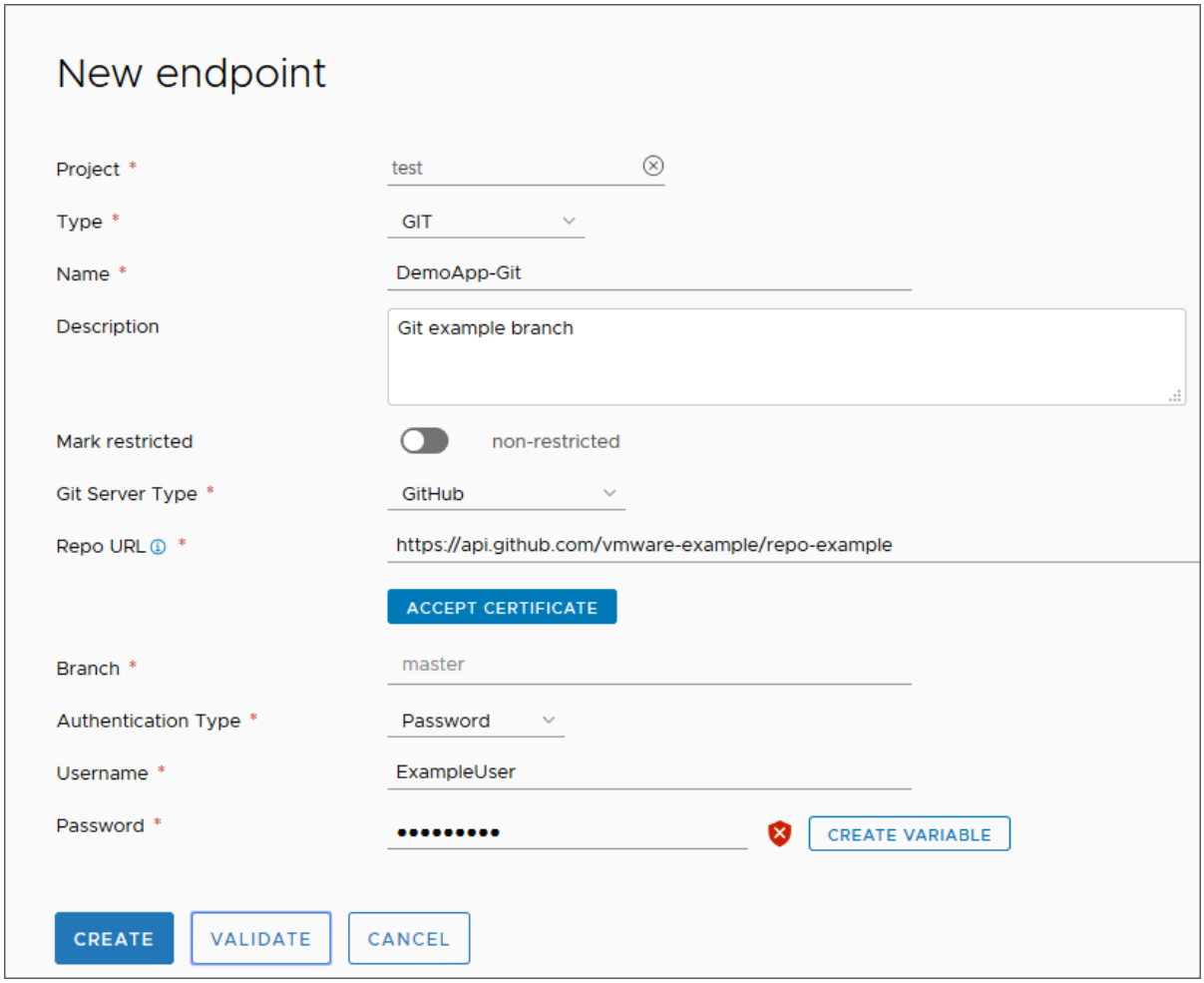
- 密码。要稍后创建 Webhook，必须为密码输入专用令牌。适用于 Git 的 Webhook 不支持使用基本身份验证创建的端点。

  使用机密变量可隐藏和加密敏感信息。可以对必须隐藏和加密以及限制在执行中使用的字符串、密码和 URL 使用受限制变量。例如，对密码或 URL 使用机密变量。可以在管道的任何类型的任务中使用机密变量和受限制变量。

- 专用令牌。此令牌特定于 Git，可提供对特定操作的访问权限。请参见 https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html。您还可以为专用令牌创建变量。

2 单击**验证**，并验证端点是否连接到 vRealize Automation Code Stream。

如果未连接，请更正所有错误，然后单击**创建**。



后续步骤

要了解更多信息，请参见其他节。请参见如何使用 vRealize Automation Code Stream 中的 Git 触发器运行管道。

如何将 vRealize Automation Code Stream 与 Gerrit 集成

vRealize Automation Code Stream 支持在 Gerrit 项目中进行代码审阅时触发管道。适用于 Gerrit 的触发器定义包含 Gerrit 项目以及必须对各种事件类型运行的管道。

适用于 Gerrit 的触发器会在将监控的 Gerrit 服务器中使用 Gerrit 侦听器。要在 vRealize Automation Code Stream 中定义 Gerrit 端点，需要选择项目，并输入 Gerrit 服务器的 URL。然后，在该服务器上创建 Gerrit 侦听器时指定该端点。

如果在启用了 FIPS 的 vRealize Automation 实例中使用 Gerrit 服务器作为 vRealize Automation Code Stream 端点，则必须验证 Gerrit 配置文件是否包含正确的消息身份验证密钥。如果 Gerrit 服务器配置文件不包含正确的消息身份验证密钥，则服务器将无法正确启动，并显示以下消息：`PrivateKey/PassPhrase is incorrect`

前提条件

- 确认您可以访问要连接到的 Gerrit 服务器。
- 验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。

步骤

1 定义 Gerrit 端点。

   a 单击**配置 > 端点**，然后单击**新建端点**。

   b 选择一个项目，并为端点类型选择 **Gerrit**。然后输入名称和描述。

   c 如果该端点是基础架构中的关键业务组件，请启用**标记为受限制**。

   d 输入 Gerrit 服务器的 URL。

   可以在 URL 中提供端口号，也可以将值留空以使用默认端口。

   e 输入 Gerrit 服务器的用户名和密码。

   如果密码必须加密，请单击**创建变量**，然后选择类型：

   - 机密。当具有任何角色的用户运行管道时，密码进行解析。
   - 受限制。当具有管理员角色的用户运行管道时，密码进行解析。

   对于值，输入必须安全的密码，例如 Jenkins 服务器的密码。

   f 对于私钥，输入用于安全访问 Gerrit 服务器的 SSH 密钥。

   此密钥是位于 `.ssh` 目录中的 RSA 私钥。

   g （可选） 如果密码短语与该私钥关联，请输入密码短语。

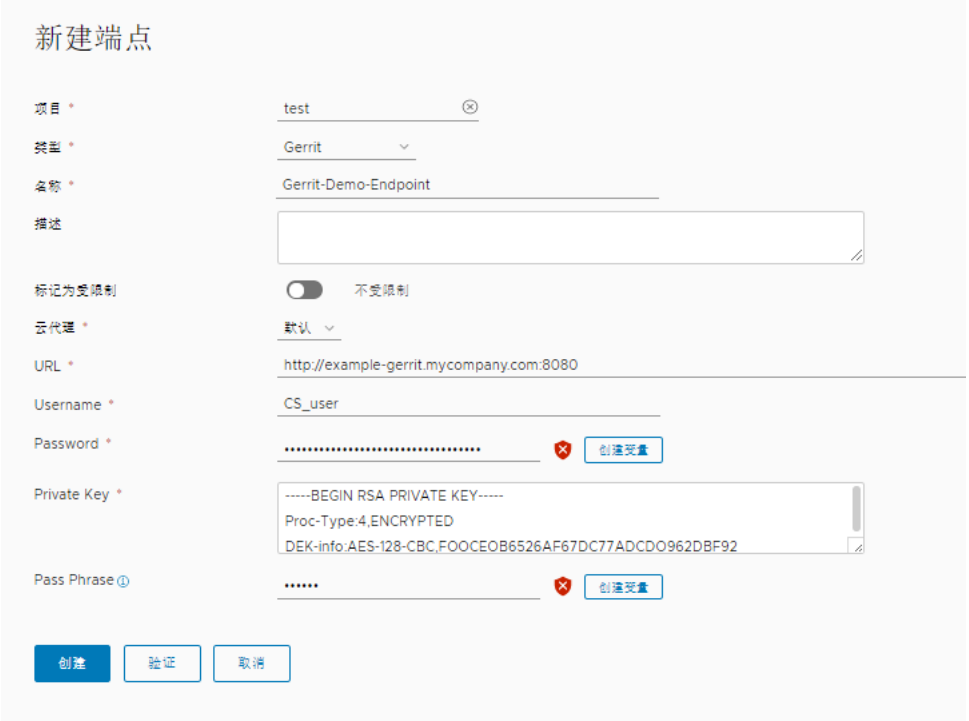
   要对密码短语进行加密，请单击**创建变量**，然后选择类型：

   - 机密。当具有任何角色的用户运行管道时，密码进行解析。
   - 受限制。当具有管理员角色的用户运行管道时，密码进行解析。

   对于值，必须输入安全的密码短语，例如 SSH 服务器的密码短语。

2 单击**验证**，然后验证 vRealize Automation Code Stream 中的 Gerrit 端点是否连接到 Gerrit 服务器。

如果未连接，请更正任何错误，然后再次单击**验证**。



3 单击**创建**。

4 验证 vRealize Automation 环境是否启用了 FIPS，或者让 Jenkins 作业使用 Jenkins URL 创建启用了 FIPS 的环境。

   a 要从命令行运行命令，请通过 SSH 连接到 vRealize Automation 8.x 设备，然后以 root 用户身份登录。例如，在端口 22、5480 或 443 上连接到完全限定域名 URL，例如 `https://cava-1-234-567.yourcompanyFQDN.com`。

   b 要检查 vRealize Automation 上的 FIPS，请运行命令 `vracli security fips`。

   c 验证命令是否返回 `FIPS mode: strict`。

5 如果 Gerrit 服务器是启用了 FIPS 的 vRealize Automation 实例中的端点，请确保 Gerrit 配置文件包含正确的消息身份验证 (MAC) 密钥。

   a 打开 Gerrit 并创建 SSH 密钥对。

   b 在 `'$site_path'/etc/gerrit.config` 中找到 Gerrit 服务器配置文件。

   c 验证 Gerrit 服务器配置文件是否包含一个或多个消息身份验证代码 (MAC) 密钥，`hmac-MD5` 除外。

   注 在 FIPS 模式下，`hmac-MD5` 不是受支持的 MAC 算法。为确保 Gerrit 服务器正常启动，Gerrit 服务器配置文件必须排除此算法。如果 Gerrit 服务器未正确启动，则会显示以下消息：
   `PrivateKey/PassPhrase is incorrect`

   以加号 (+) 开头的受支持消息身份验证代码 (MAC) 密钥名称处于启用状态。以连字符 (-) 开头的 MAC 密钥名称将从默认 MAC 列表中移除。默认情况下，这些受支持的 MAC 可在 vRealize Automation Code Stream 中用于 Gerrit 服务器：

   - `hmac-md5-96`
   - `hmac-sha1`
   - `hmac-sha1-96`
   - `hmac-sha2-256`
   - `hmac-sha2-512`

后续步骤

要了解更多信息，请参见其他节。请参见如何使用 vRealize Automation Code Stream 中的 Gerrit 触发器运行管道。

如何将 vRealize Automation Code Stream 与 vRealize Orchestrator 集成

vRealize Automation Code Stream 可以与 vRealize Orchestrator (`vRO`) 集成，以便通过运行 `vRO` 工作流来扩展其功能。vRealize Orchestrator 包括许多可与第三方工具集成的预定义工作流。这些工作流有助于自动执行和管理 DevOps 流程以及自动执行批量操作等。

例如，您可以在管道的 `vRO` 任务中使用工作流，以启用用户、移除用户、移动虚拟机以及与测试框架集成以在管道运行过程中测试代码等。可以在 code.vmware.com 中浏览 vRealize Orchestrator 工作流的代码示例。

使用 vRealize Orchestrator 工作流，管道可以在生成、测试和部署应用程序的过程中运行操作。您可以将预定义的工作流包含到管道中，也可以创建并使用自定义工作流。每个工作流都包含输入、任务和输出。

要在管道中运行 `vRO` 工作流，该工作流必须显示在包含到管道的 `vRO` 任务的可用工作流列表中。

管理员首先必须在 vRealize Orchestrator 中执行以下步骤，然后工作流才可以显示在管道的 vRO 任务中：

1 将 CODESTREAM 标记应用到 vRO 工作流。

2 将 vRO 工作流标记为全局工作流。

前提条件

- 确认您能够以管理员身份访问 vRealize Orchestrator 的内部部署实例。要获取帮助，请向管理员咨询并参见 vRealize Orchestrator 文档。
- 验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是其成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。
- 在 vRealize Automation Code Stream 中，创建一个管道并添加一个阶段。

步骤

1 以管理员的身份准备 vRealize Orchestrator 工作流以让管道运行。

   a 在 vRealize Orchestrator 中，查找要在管道中使用的工作流，例如用于启用用户的工作流。

   如果所需的工作流不存在，您可以创建该工作流。

   b 在搜索栏中，输入**标记工作流**以查找标记工作流。

   c 在标记工作流卡视图上，单击**运行**，随后将显示配置区域。

   d 在标记的工作流文本区域中，输入在 vRealize Automation Code Stream 管道中使用的工作流的名称，然后从列表中选择该工作流。

   e 在标记和值文本区域中，以大写字母形式输入 CODESTREAM。

   f 单击**全局标记**复选框。

   g 单击**运行**，将 CODESTREAM 标记附加到您要在 vRealize Automation Code Stream 管道中选择的工作流。

   h 在导航窗格中，单击**工作流**，并确认 CODESTREAM 标记显示在管道将运行的工作流卡视图上。

   登录到 vRealize Automation Code Stream 并向管道添加 vRO 任务后，标记的工作流会显示在工作流列表中。

2 在 vRealize Automation Code Stream 中，为 vRealize Orchestrator 实例创建端点。

   a 单击**端点 > 新建端点**。

   b 选择一个项目。

   c 输入相关的名称。

d 输入 vRealize Orchestrator 端点的 URL。

请使用以下格式：**https://host-n-01-234.eng.vmware.com:8281**

请勿使用以下格式：https://host-n-01-234.eng.vmware.com:8281/vco/api

vRealize Automation 设备中嵌入的 vRealize Orchestrator 实例的 URL 是设备的 FQDN，不含端口或路径。例如：**https://vra-appliance.yourdomain.local**

从 vRealize Automation 8.x 开始，对于外部 vRealize Orchestrator Appliance，设备的 FQDN 为 **https://vro-appliance.yourdomain.local**

如果在添加端点时出现问题，可能需要导入具有 SHA-256 证书指纹但移除了冒号的 YAML 配置。例如，**B0:01:A2:72...** 变为 **B001A272...**。示例 YAML 代码类似于：

```
```
---
project: Demo
kind: ENDPOINT
name: external-vro
description: ''
type: vro
properties:
  url: https://yourVROhost.yourdomain.local
  username: yourusername
  password: yourpassword
  fingerprint: <your_fingerprint>
```
```

对于 vRealize Automation 7.x 中包含的外部 vRealize Orchestrator Appliance，设备的 FQDN 为 **https://vro-appliance.yourdomain.local:8281/vco**

e 如果您输入的 URL 需要证书，单击**接受证书**。

f 输入 vRealize Orchestrator 服务器的用户名和密码。

如果使用非本地用户进行身份验证，则必须省略用户名的域部分。例如，要使用 **svc_vro@yourdomain.local** 进行身份验证，必须在**用户名**文本区域中输入 **svc_vro**。

3 准备管道以运行 vRO 任务。

a 向管道阶段添加 vRO 任务。

b 输入相关的名称。

c 在“工作流属性”区域，选择 vRealize Orchestrator 端点。

d 选择您在 vRealize Orchestrator 中标记为 CODESTREAM 的工作流。

如果选择您创建的自定义工作流，可能需要键入输入参数值。

e 对于**执行任务**，单击**基于条件**。

f 输入要在管道运行时应用的条件。

何时运行管道...	选择条件...
基于条件	仅在已定义的条件的评估结果为 true 时运行管道任务。如果该条件为 false，则跳过任务。 vRO 任务允许包含布尔表达式，该表达式使用以下操作数和运算符。 ■ 管道变量，例如 `${pipeline.variableName}`。输入变量时，仅可使用大括号。 ■ 任务输出变量，例如 `${Stage1.task1.machines[0].value.hostIp[0]}`。 ■ 默认管道绑定变量，例如 `${releasePipelineName}`。 ■ 不区分大小写的布尔值，例如 `true`、`false`、`'true'` 和 `'false'`。 ■ 不带引号的整数值或小数值。 ■ 带有单引号或双引号的字符串值，例如 `"test"` 和 `'test'`。 ■ 字符串和数值类型的值，例如 `== Equals` 和 `!= Not Equals`。 ■ 关系运算符，例如 `>`、`>=`、`<` 和 `<=`。 ■ 布尔逻辑，例如 `&&` 和 `\|\|`。 ■ 算术运算符，例如 `+`、`-`、`*` 和 `/`。 ■ 带有圆括号的嵌套表达式。 ■ 包含字面值 `ABCD` 的字符串的评估结果为 false，因此将跳过任务。 ■ 不支持一元运算符。 示例条件可以是 `${Stage1.task1.output} == “Passed” || ${pipeline.variableName} == 39`
始终	如果选择**始终**，管道将运行任务而不应用条件。

g 输入问候消息。

h 单击**验证任务**，并更正出现的任何错误。

4 保存、启用并运行管道。

5 管道运行之后，检查结果。

   a 单击**执行**。

   b 单击该管道。

   c 单击相关任务。

   d 检查结果、输入值和属性。

   您可以确定工作流执行 ID、对任务做出响应的人员、该人员做出响应的时间以及该人员添加的任何注释。

结果

恭喜！您已标记了要在 vRealize Automation Code Stream 中使用的 vRealize Orchestrator 工作流，并在 vRealize Automation Code Stream 管道中添加了 vRO 任务，以使其运行在 DevOps 环境中自动执行某个操作的工作流。

示例： vRO 任务输出格式

vRO 任务的输出格式类似于以下示例。

```
[{
                "name": "result",
                "type": "STRING",
                "description": "Result of workflow run.",
                "value": ""
},
{
                "name": "message",
                "type": "STRING",
                "description": "Message",
                "value": ""
}]
```

后续步骤

继续将其他 vRO 工作流任务包含到管道中，以便在开发环境、测试环境和生产环境中自动执行任务。

在 vRealize Automation Code Stream 中触发管道

7

您可以让 vRealize Automation Code Stream 在某些事件发生时触发管道。

例如：

- Docker 触发器可以在创建或更新新的工件时运行管道。
- 适用于 Git 的触发器可以在开发人员更新代码时触发管道。
- 适用于 Gerrit 的触发器可以在开发人员审阅代码时触发管道。
- `curl` 命令和 `wget` 命令可以指示 Jenkins 在构建完成后触发管道。

本章讨论了以下主题：

- 如何使用 vRealize Automation Code Stream 中的 Docker 触发器运行持续交付管道
- 如何使用 vRealize Automation Code Stream 中的 Git 触发器运行管道
- 如何使用 vRealize Automation Code Stream 中的 Gerrit 触发器运行管道

如何使用 vRealize Automation Code Stream 中的 Docker 触发器运行持续交付管道

作为 vRealize Automation Code Stream 管理员或开发人员，您可以在 vRealize Automation Code Stream 中使用 Docker 触发器。每当创建或更新生成工件时，Docker 触发器都会运行独立的持续交付 (CD) 管道。Docker 触发器将运行 CD 管道，而该管道会将新的或已更新的工件作为容器映像推送到 Docker Hub 存储库。CD 可以作为自动化内部版本的一部分运行。

例如，要通过 CD 管道持续部署已更新的容器映像，请使用 Docker 触发器。当您的容器映像签入 Docker 注册表时，Docker Hub 中的 Webhook 会通知 vRealize Automation Code Stream 该映像已更改。此通知会触发 CD 管道使用更新的容器映像运行，并将该映像上载到 Docker Hub 存储库。

要使用 Docker 触发器，您需要在 vRealize Automation Code Stream 中执行几个步骤。

表 7-1. 如何使用 Docker 触发器

执行的操作...	有关该操作的更多信息...
创建 Docker 注册表端点。	要使 vRealize Automation Code Stream 触发管道，您必须有 Docker 注册表端点。如果端点不存在，您可以选择在为 Docker 触发器添加 Webhook 时创建端点。 Docker 注册表端点包含 Docker Hub 存储库的 URL。
向管道添加可在管道运行时自动插入 Docker 参数的输入参数。	可以将 Docker 参数插入到管道。这些参数可以包括 Docker 事件所有者名称、映像、存储库名称、存储库命名空间和标记。 可以在 CD 管道中包含输入参数，Docker Webhook 会在管道触发之前将这些输入参数传递到管道。
创建 Docker Webhook。	在 vRealize Automation Code Stream 中创建 Docker Webhook 时，它还会在创建 Docker Hub 中创建相应的 Webhook。 vRealize Automation Code Stream 中的 Docker Webhook 通过包含在 Webhook 中的 URL 连接到 Docker Hub。 Webhook 与彼此通信，并在 Docker Hub 中创建或更新工件时触发管道。 如果在 vRealize Automation Code Stream 中更新或删除 Docker Webhook，则 Docker Hub 中的 Webhook 也会更新或删除。
在管道中添加并配置 Kubernetes 任务。	当在 Docker Hub 存储库中创建或更新工件时，管道将触发。然后，通过管道将工件部署到 Kubernetes 集群中的 Docker 主机。
将本地 YAML 定义包含到任务中。	应用于部署任务的 YAML 定义包括 Docker 容器映像。如果需要从专有存储库下载映像，则 YAML 文件必须包含具有 Docker 配置密钥的部分。请参见在使用智能管道模板之前在 vRealize Automation Code Stream 中计划 CICD 本机构建中的 CD 部分

在 Docker Hub 存储库中创建或更新工件时，Docker Hub 中的 Webhook 会通知 vRealize Automation Code Stream 中的 Webhook，而后者会触发管道。将执行以下操作：

1 Docker Hub 向 Webhook 中的 URL 发送 POST 请求。

2 vRealize Automation Code Stream 运行 Docker 触发器。

3 Docker 触发器启动 CD 管道。

4 CD 管道将工件推送到 Docker Hub 存储库。

5 vRealize Automation Code Stream 触发其 Docker Webhook，后者会运行 CD 管道以将工件部署到 Docker 主机。

在此示例中，您在 vRealize Automation Code Stream 中创建一个 Docker 端点和一个 Docker Webhook，以将应用程序部署到 Kubernetes 开发集群。步骤包括 Docker 发布到 Webhook 中的 URL 的负载的示例代码，它使用的 API 代码以及带有安全令牌的身份验证代码。

前提条件

- 确认您的 vRealize Automation Code Stream 实例中存在持续交付 (CD) 管道。还要确认该管道包含一个或多个用于部署应用程序的 Kubernetes 任务。请参见第 4 章 在 vRealize Automation Code Stream 中规划本地构建、集成和交付代码 。
- 验证您是否可以访问现有 Kubernetes 集群以便 CD 管道将应用程序部署到该集群进行开发。
- 验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是其成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。

步骤

1 创建 Docker 注册表端点。

   a 单击**端点**。

   b 单击**新建端点**。

   c 开始键入现有项目的名称。

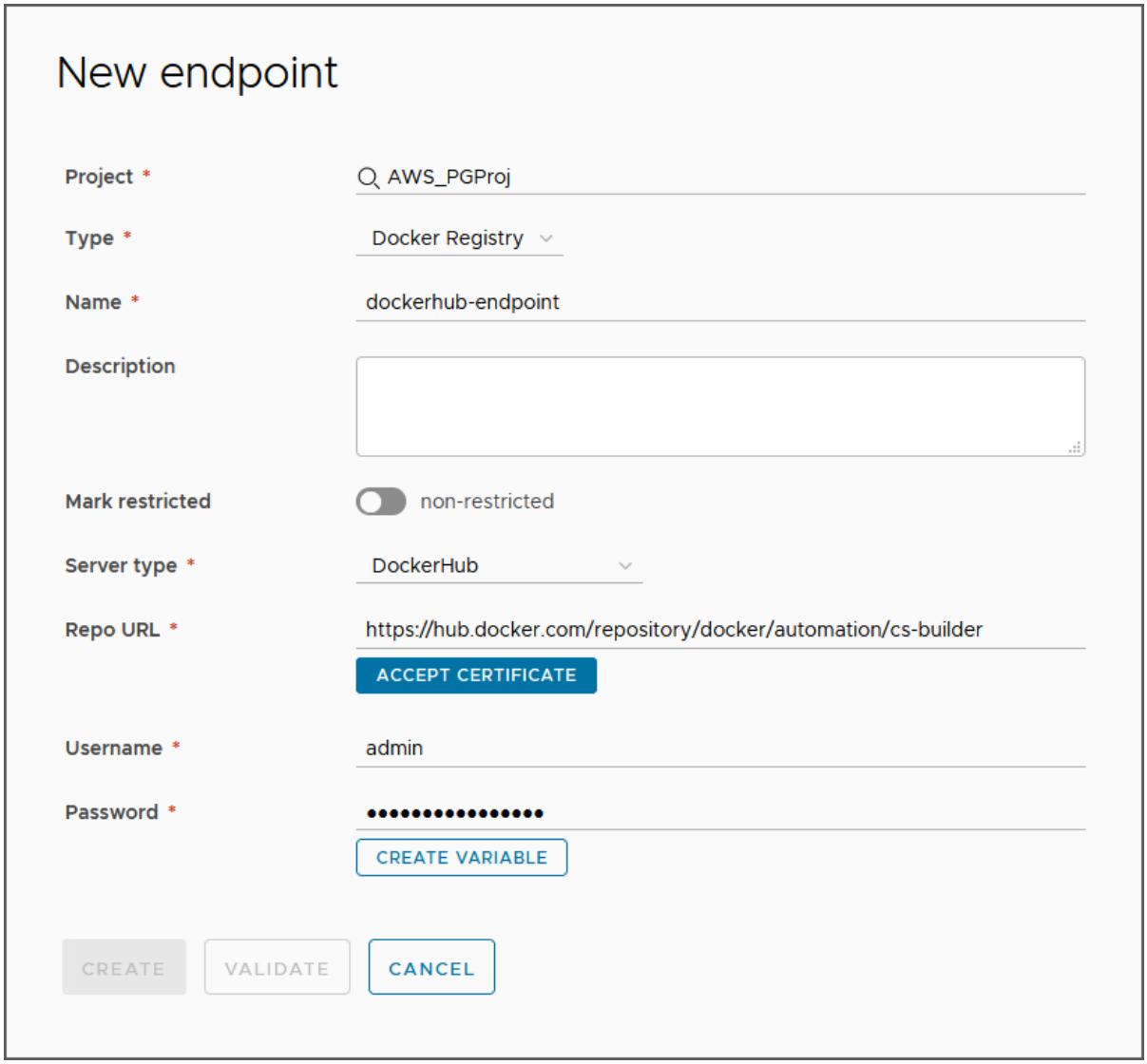
   d 选择 **Docker 注册表**作为类型。

   e 输入相关的名称。

   f 选择 **DockerHub** 作为服务器类型。

g 输入 Docker Hub 存储库的 URL。

h 输入可以访问存储库的名称和密码。



2 在 CD 管道中，设置输入属性以在管道运行时自动插入 Docker 参数。

3 创建 Docker Webhook。

a 单击**触发器 > Docker**。

b 单击**为 Docker 新建 Webhook**。

c 选择一个项目。

d 输入相关的名称。

e 选择您的 Docker 注册表端点。

如果端点尚未存在，请单击**创建端点**以创建端点。

f 选择 Webhook 要触发的插入了 Docker 参数的管道。请参见步骤 2。

如果为管道配置了自定义添加的输入参数，则“输入参数”列表将显示参数和值。您可以输入将通过触发器事件传递到管道的输入参数的值。或者，也可以将这些值留空，或使用默认值（如果已定义）。

有关输入选项卡上的参数的详细信息，请参见在手动添加任务之前在 vRealize Automation Code Stream 中规划 CICD 本地构建。

g 输入 API 令牌。

CSP API 令牌将对您进行身份验证，以与 vRealize Automation Code Stream 建立外部 API 连接。要获取 API 令牌，请执行以下操作：

1 单击**生成令牌**。

2 输入与您的用户名和密码关联的电子邮件地址，然后单击**生成**。

所生成令牌的有效期为六个月。该令牌也称为刷新令牌。

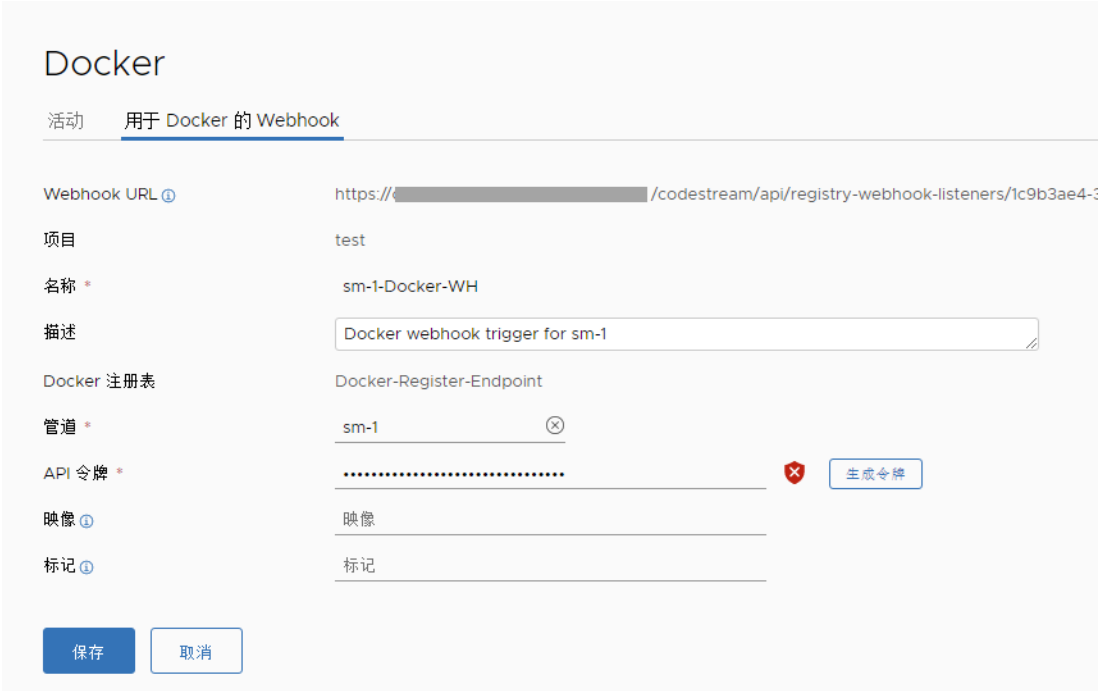
- 要将令牌保留为变量以供将来使用，请单击**创建变量**，输入变量的名称，然后单击**保存**。
- 要将令牌保留为文本值以供将来使用，请单击**复制**，然后将令牌粘贴到文本文件中以在本地保存。

您可以选择创建变量并将令牌存储在文本文件中，以供将来使用。

3 单击**关闭**。

h 输入内部版本映像。

i 输入标记。



j 单击**保存**。

此时将显示 Webhook 卡视图并启用 Docker Webhook。如果要虚拟推送到 Docker Hub 存储库，而不触发 Docker Webhook 和运行管道，请单击**禁用**。

4 在 CD 管道中，配置 Kubernetes 部署任务。

a 在 Kubernetes 任务属性中，选择您的 Kubernetes 开发集群。

b 选择**创建**操作。

c 为负载源选择**本地定义**。

d 然后选择本地 YAML 文件。

例如，Docker Hub 可能将以下本地 YAML 定义作为负载发布到 Webhook 中的 URL：

```
{
"callback_url": "https://registry.hub.docker.com/u/svendowideit/testhook/hook/
2141b5bi5i5b02bec211i4eeih0242eg11000a/",
"push_data": {
"images": [
"27d47432a69bca5f2700e4dff7de0388ed65f9d3fb1ec645e2bc24c223dc1cc3",
"51a9c7c1f8bb2fa19bcd09789a34e63f35abb80044bc10196e304f6634cc582c",
"..."
],
"pushed_at": 1.417566161e+09,
"pusher": "trustedbuilder",
"tag": "latest"
},
"repository": {
"comment_count": 0,
"date_created": 1.417494799e+09,
"description": "",
"dockerfile": "#\n# BUILD\u0009\u0009docker build -t svendowideit/apt-cacher .\n#
RUN\u0009\u0009docker run -d -p 3142:3142 -name apt-cacher-run apt-cacher\n#\n# and
then you can run containers with:\n# \u0009\u0009docker run -t -i -rm -e http_proxy
http://192.168.1.2:3142/ debian
bash\n#\nFROM\u0009\u0009ubuntu\n\n\nVOLUME\u0009\u0009[\/var/cache/apt-cacher-ng\]
\nRUN\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n\nEXPOSE
\u0009\u00093142\nCMD\u0009\u0009chmod 777 /var/cache/apt-cacher-ng ; /etc/init.d/apt-
cacher-ng start ; tail -f /var/log/apt-cacher-ng/*\n",
"full_description": "Docker Hub based automated build from a GitHub repo",
"is_official": false,
"is_private": true,
"is_trusted": true,
"name": "testhook",
"namespace": "svendowideit",
"owner": "svendowideit",
"repo_name": "svendowideit/testhook",
"repo_url": "https://registry.hub.docker.com/u/svendowideit/testhook/",
"star_count": 0,
"status": "Active"
}
}
```

用于在 Docker Hub 中创建 Webhook 的 API 采用以下格式：`https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/`

JSON 代码正文类似于：

```
{
"name": "demo_webhook",
"webhooks": [
{
"name": "demo_webhook",
```

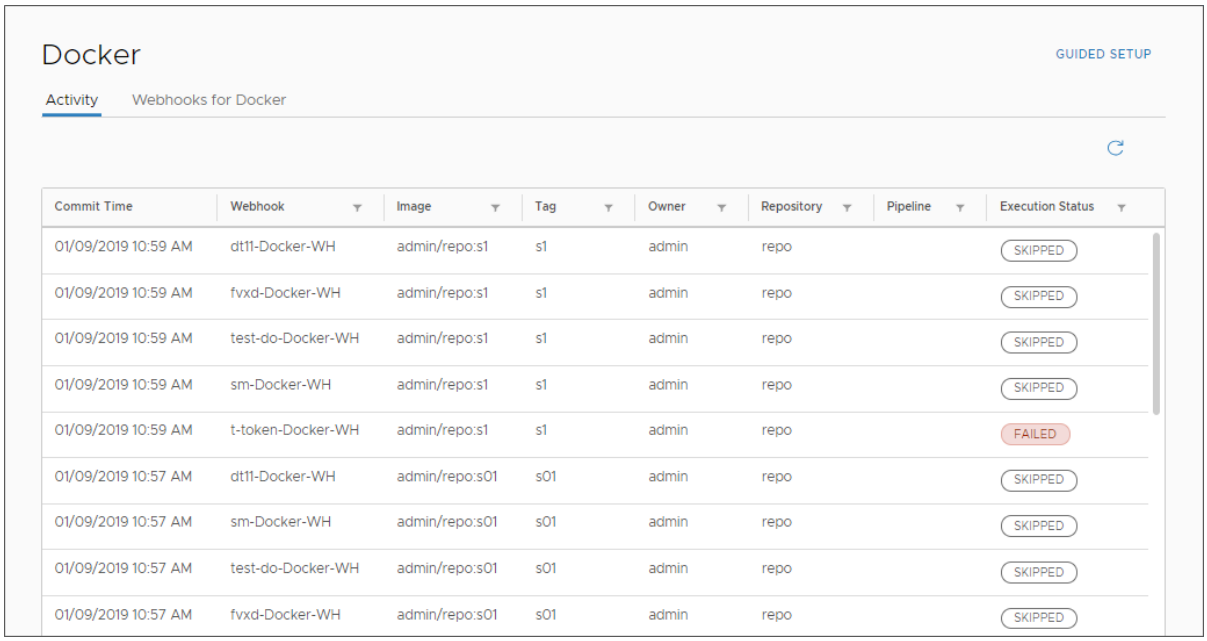
```
"hook_url": "http://www.google.com"
}
]
}
```

为了从 Docker Hub 服务器接收事件，在 vRealize Automation Code Stream 中创建的 Docker Webhook 的身份验证方案对 Webhook 使用允许列表身份验证机制和随机字符串令牌。它基于安全令牌筛选事件，安全令牌可以附加到 `hook_url`。

vRealize Automation Code Stream 可以使用已配置的安全令牌验证来自 Docker Hub 服务器的任何请求。例如：`hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""`

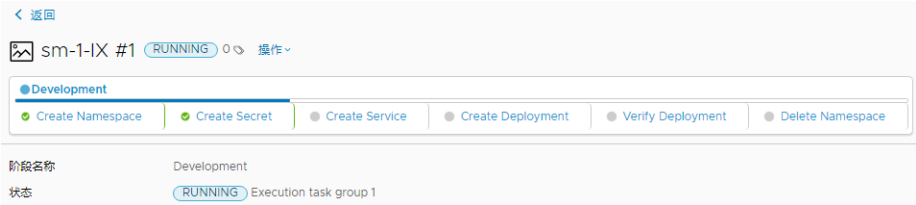
5 在 Docker Hub 存储库中创建 Docker 工件。或者更新现有工件。

6 要确认已触发触发器，并查看 Docker Webhook 上的活动，请单击**触发器 > Docker > 活动**。



7 单击**执行**，并在管道运行时观察管道。

8 单击运行阶段，并在管道运行过程中查看任务。



结果

恭喜！您设置了 Docker 触发器以持续运行 CD 管道。现在，您的管道可以将新的和更新的 Docker 工件上载到 Docker Hub 存储库。

后续步骤

验证新的或已更新的工件是否已部署到 Kubernetes 开发集群中的 Docker 主机。

如何使用 vRealize Automation Code Stream 中的 Git 触发器运行管道

作为 vRealize Automation Code Stream 管理员或开发人员，您可以使用 Git 触发器将 vRealize Automation Code Stream 与 Git 生命周期集成。您在 GitHub、GitLab 或 Bitbucket Enterprise 中进行代码更改时，事件将通过 Webhook 与 vRealize Automation Code Stream 通信，并触发管道。当 vRealize Automation Cloud Assembly 与 GitLab、GitHub 和 Bitbucket 内部部署企业版本可通过同一网络进行访问时，Webhook 可用于企业版本。

在 vRealize Automation Code Stream 中添加适用于 Git 的 Webhook 时，它还会在 GitHub、GitLab 或 Bitbucket 存储库中创建 Webhook。如果以后更新或删除 Webhook，该操作也会更新或删除 GitHub、GitLab 或 Bitbucket 中的 Webhook。

Webhook 定义必须在要监控的存储库分支上包括 Git 端点。要创建 Webhook， vRealize Automation Code Stream 将使用该 Git 端点。如果该端点不存在，您可以在添加 Webhook 时创建该端点。此示例假设您在 GitHub 中有预定义的 Git 端点。

注 您的 Git 端点必须使用专用令牌进行身份验证。如果您的 Git 端点使用密码进行基本身份验证，您将无法创建 Webhook。

可以通过使用同一 Git 端点并在 Webhook 配置页面中为分支名称提供不同的值，为不同的分支创建多个 Webhook。要为同一 Git 存储库中的其他分支创建其他 Webhook，可以在 Webhook 中提供分支名称，而无需针对多个分支多次克隆 Git 端点。此方法允许重用 Git 端点。如果 Git Webhook 中的分支与端点中的分支相同，则无需在 Git Webhook 页面中提供分支名称。

此示例显示了如何将 Git 触发器与 GitHub 存储库结合使用，但必备条件包括使用其他 Git 服务器类型时需要做的准备工作。

前提条件

- 验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是其成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。
- 确认要监控的 GitHub 分支上有 Git 端点。请参见如何将 vRealize Automation Code Stream 与 Git 集成。
- 确认您有权在 Git 存储库中创建 Webhook。
- 如果在 GitLab 中配置 Webhook，则更改 GitLab Enterprise 中的默认网络设置以启用出站请求，并允许创建本地 Webhook。

  注 仅 GitLab Enterprise 需要进行此更改。这些设置不适用于 GitHub 或 Bitbucket。

  a 以管理员身份登录到 GitLab Enterprise 实例。

  b 使用 URL（如 `http://{gitlab-server}/admin/application_settings/network`）访问网络设置。

  c 展开**出站请求**，然后单击：

    - 允许从 Web hook 和服务向本地网络发出请求。
    - 允许从系统 hook 向本地网络发出请求。
- 对于要触发的管道，确认已设置输入属性，以便在管道运行时插入 Git 参数。

  有关输入参数的信息，请参见在手动添加任务之前在 vRealize Automation Code Stream 中规划 CICD 本地构建。

步骤

1 在 vRealize Automation Code Stream 中，单击**触发器 > Git**。

2 单击**适用于 Git 的 Webhook** 选项卡，然后单击**新建适用于 Git 的 Webhook**。

  a 选择一个项目。

  b 为 Webhook 输入有意义的名称和说明。

c 选择为要监控的分支配置的 Git 端点。

创建 Webhook 时，Webhook 定义包含当前端点的详细信息。

- 如果稍后在端点中更改 Git 类型、Git 服务器类型或 Git 存储库 URL，则 Webhook 将无法再触发管道，因为它将尝试使用原始端点详细信息访问 Git 存储库。必须删除 Webhook，然后重新创建包含该端点的 Webhook。
- 如果稍后更改端点中的身份验证类型、用户名或专用令牌，则 Webhook 将继续工作。
- 如果使用的是 BitBucket 存储库，则存储库的 URL 必须采用以下格式之一：`https://api.bitbucket.org/{user}/{repo name}` 或 `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`。

注 如果之前创建了使用 Git 端点（使用密码进行基本身份验证）的 Webhook，则必须删除该 Webhook，然后重新定义具有 Git 端点（使用专用令牌进行身份验证）的 Webhook。

请参见如何将 vRealize Automation Code Stream 与 Git 集成。

d （可选） 输入希望 Webhook 监控的分支。

如果未指定分支，则 Webhook 将监控为 Git 端点配置的分支。

e （可选） 为 Webhook 生成密钥令牌。

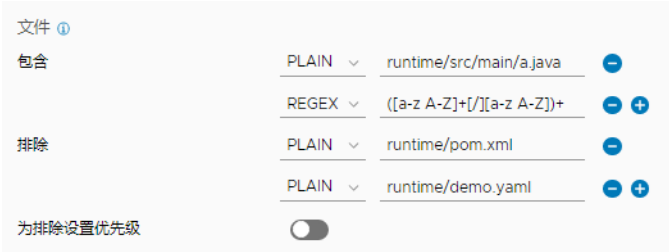
如果使用密钥令牌， vRealize Automation Code Stream 会为 Webhook 生成随机字符串令牌。随后，当 Webhook 收到 Git 事件数据时，它会将数据与密钥令牌一起发送。 vRealize Automation Code Stream 使用这些信息来确定调用是否来自预期的源，例如已配置的 GitHub 实例、存储库和分支。密钥令牌可提供额外的安全层，用于验证 Git 事件数据是否来自正确的源。

f （可选） 提供文件包含或文件排除作为触发器的条件。

- 文件包含。如果提交中的任何文件与包含路径或正则表达式中指定的文件匹配，将触发管道。如果指定正则表达式，则 vRealize Automation Code Stream 仅当更改集中的文件名与提供的表达式匹配时才触发管道。为单个存储库中的多个管道配置触发器时，正则表达式筛选器很有用。
- 文件排除。当提交中的所有文件与排除路径或正则表达式中指定的文件匹配时，不会触发管道。
- 排除优先。打开“排除优先”时，可确保即使提交中的任何文件与排除路径或正则表达式中指定的文件匹配也不触发管道。默认设置为“关闭”。

如果条件同时满足文件包含和文件排除，则不会触发管道。

在以下示例中，文件包含和文件排除都是触发器的条件。



- 对于文件包含，提交对 `runtime/src/main/a.java` 或任意 java 文件的任何更改都将触发事件配置中配置的管道。
- 对于文件排除，仅提交同时对这两个文件的更改不会触发事件配置中配置的管道。

g 对于 Git 事件，选择**推送**或**提取**请求。

h 输入 API 令牌。

CSP API 令牌将对您进行身份验证，以与 vRealize Automation Code Stream 建立外部 API 连接。要获取 API 令牌，请执行以下操作：

1 单击**生成令牌**。

2 输入与您的用户名和密码关联的电子邮件地址，然后单击**生成**。

所生成令牌的有效期为六个月。该令牌也称为刷新令牌。

- 要将令牌保留为变量以供将来使用，请单击**创建变量**，输入变量的名称，然后单击**保存**。
- 要将令牌保留为文本值以供将来使用，请单击**复制**，然后将令牌粘贴到文本文件中以在本地保存。

您可以选择创建变量并将令牌存储在文本文件中，以供将来使用。

3 单击**关闭**。

i 选择 Webhook 要触发的管道。

如果管道包括自定义添加的输入参数，则“输入参数”列表将显示参数和值。您可以输入通过触发器事件传递到管道的输入参数的值。或者，也可以将这些值留空，或使用默认值（如果已定义）。

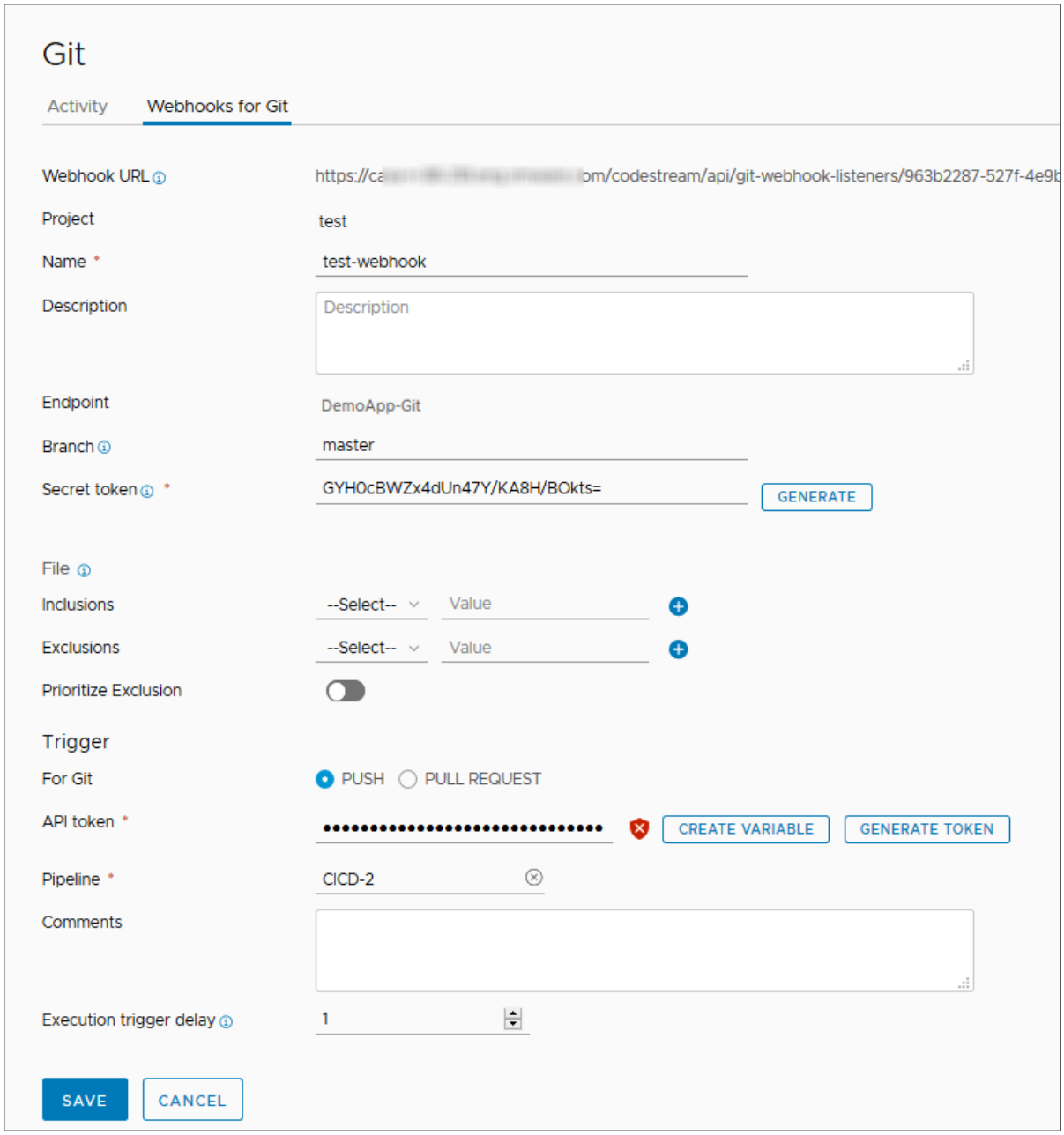
有关 Git 触发器的自动插入输入参数的信息，请参见必备条件。

j 单击**创建**。

Webhook 将显示为新卡。

3 单击该 Webhook 卡。

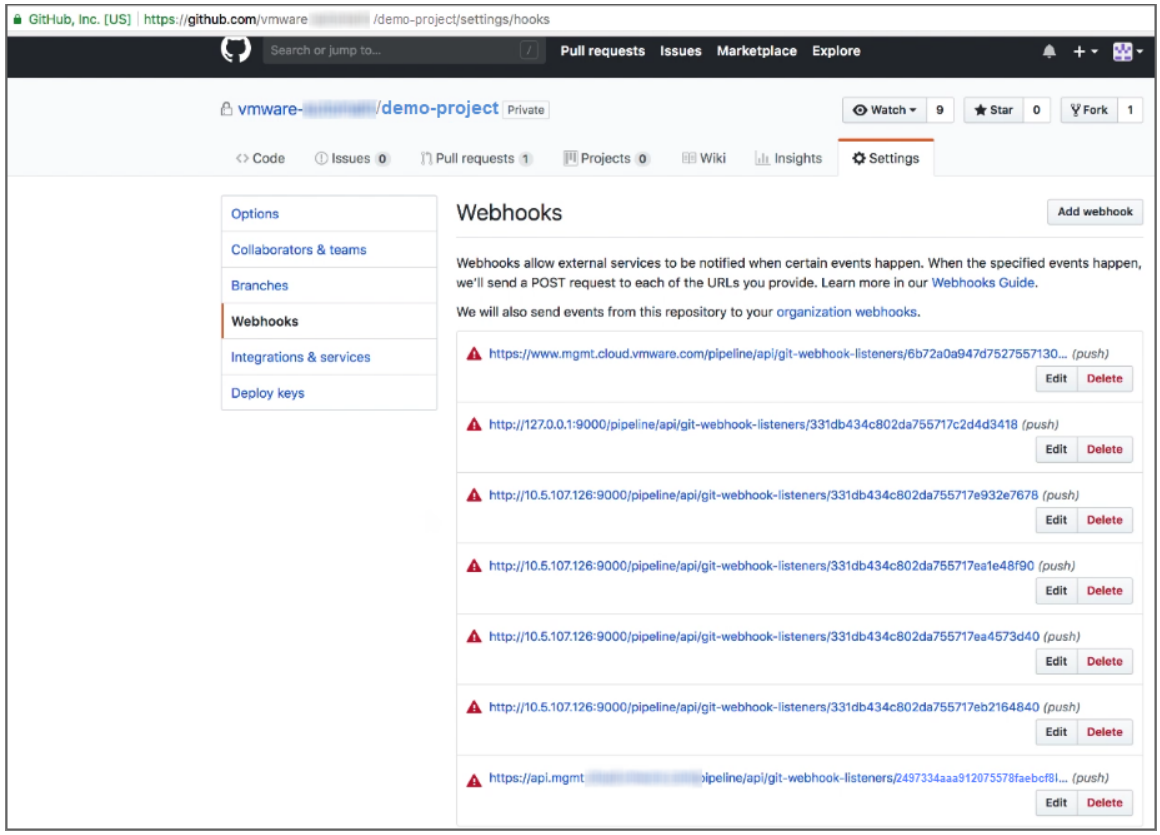
重新显示 Webhook 数据表单时，您会看到 Webhook URL 添加到表单顶部。Git Webhook 通过该 Webhook URL 连接到 GitHub 存储库。



4 在新的浏览器窗口中，打开通过 Webhook 连接的 GitHub 存储库。

a 要查看在 vRealize Automation Code Stream 中添加的 Webhook，请单击**设置**选项卡，然后选择 **Webhook**。

在 Webhook 列表的底部，您会看到相同的 Webhook URL。



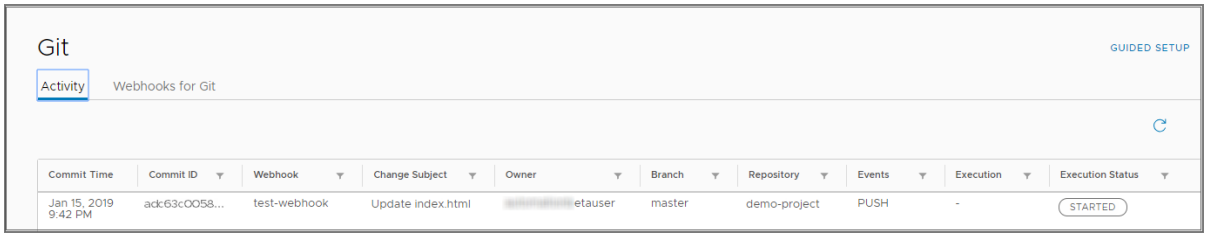
b 要更改代码，请单击**代码**选项卡，然后在分支上选择一个文件。编辑该文件后，提交更改。

c 要验证 Webhook URL 是否正常运行，请单击**设置**选项卡，然后再次选择 **Webhook**。

在 Webhook 列表的底部，Webhook URL 旁边会显示一个绿色勾选标记。

5 返回 vRealize Automation Code Stream 以查看 Git Webhook 上的活动。单击**触发器 > Git > 活动**。

在“执行状态”下，验证管道运行是否已启动。



6 单击**执行**，并在管道运行过程中跟踪管道。

要观察管道运行，可以按刷新按钮。

结果

恭喜！您已成功使用 Git 触发器！

如何使用 vRealize Automation Code Stream 中的 Gerrit 触发器运行管道

作为 vRealize Automation Code Stream 管理员或开发人员，您可以使用 Gerrit 触发器将 vRealize Automation Code Stream 与 Gerrit 代码审阅生命周期集成。在 Gerrit 项目上创建修补程序集、发布草稿、合并代码更改或直接在 Git 分支上推送更改时，事件将触发管道运行。

添加 Gerrit 触发器时，可以选择 Gerrit 侦听器、Gerrit 服务器上 Gerrit 项目，然后配置 Gerrit 事件。在以下示例中，先配置 Gerrit 侦听器，然后在三个不同管道上具有两个事件的 Gerrit 触发器中使用该侦听器。

前提条件

- 验证您是 vRealize Automation Code Stream 中项目的成员。如果您不是其成员，则让 vRealize Automation Code Stream 管理员将您添加为项目的成员。请参见如何在 vRealize Automation Code Stream 中添加项目。
- 确认在 vRealize Automation Code Stream 中配置了 Gerrit 端点。请参见如何将 vRealize Automation Code Stream 与 Gerrit 集成。
- 对于要触发的管道，确认您将管道的输入属性设置为 **Gerrit**，以允许管道在管道运行时将 Gerrit 参数作为输入接收。

有关输入参数的信息，请参见在手动添加任务之前在 vRealize Automation Code Stream 中规划 CICD 本地构建。

步骤

1 在 vRealize Automation Code Stream 中，单击**触发器 > Gerrit**。

2 （可选） 单击**侦听器**选项卡，然后单击**新建侦听器**。

注 如果已定义计划用于 Gerrit 触发器的 Gerrit 侦听器，则跳过此步骤。

a 选择一个项目。

b 输入 Gerrit 侦听器的名称。

c 选择 Gerrit 端点。

d 输入 API 令牌。

CSP API 令牌将对您进行身份验证，以与 vRealize Automation Code Stream 建立外部 API 连接。要获取 API 令牌，请执行以下操作：

1 单击**生成令牌**。

2 输入与您的用户名和密码关联的电子邮件地址，然后单击**生成**。

所生成令牌的有效期为六个月。该令牌也称为刷新令牌。

- 要将令牌保留为变量以供将来使用，请单击**创建变量**，输入变量的名称，然后单击**保存**。
- 要将令牌保留为文本值以供将来使用，请单击**复制**，然后将令牌粘贴到文本文件中以在本地保存。

您可以选择创建变量并将令牌存储在文本文件中，以供将来使用。

3 单击**关闭**。

如果创建变量，则 API 令牌将显示通过使用美元绑定输入的变量名称。如果复制令牌，则 API 令牌将显示经过屏蔽的令牌。

e 要验证令牌和端点详细信息，请单击**验证**。

您的令牌将在 90 天后过期。

f 单击**创建**。

g 在侦听器卡视图上，单击**连接**。

侦听器将开始监控 Gerrit 服务器上的所有活动，并侦听该服务器上已启用的任何触发器。要停止侦听该服务器上的触发器，请停用触发器。

注 要更新连接到侦听器的 Gerrit 端点，必须先断开侦听器的连接，然后再更新该端点。

- 单击**配置 > 触发器 > Gerrit**。
- 单击**侦听器**选项卡。
- 单击要更新且已连接到端点的侦听器上的**断开连接**。

3 单击**触发器**选项卡，然后单击**新建触发器**。

4 选择 Gerrit 服务器上的项目。

5 输入名称。

Gerrit 触发器名称必须唯一。

6 选择已配置的 Gerrit 侦听器。

通过使用 Gerrit 侦听器， vRealize Automation Code Stream 提供服务器上可用的 Gerrit 项目列表。

7 选择 Gerrit 服务器上的项目。

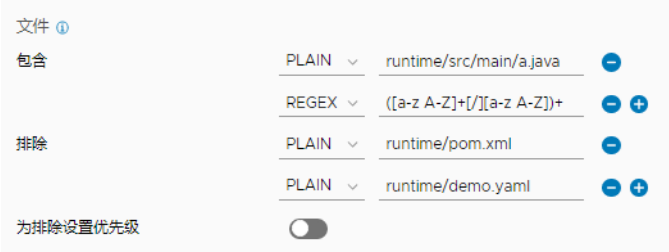
8 输入存储库中 Gerrit 侦听器将监控的分支。

9 （可选） 提供文件包含或文件排除作为触发器的条件。

- 提供文件包含以触发管道。当提交中的任何文件与包含路径或正则表达式中指定的文件匹配时，将触发管道。指定正则表达式时， vRealize Automation Code Stream 仅触发更改集中的文件名与提供的表达式匹配的管道。为单个存储库中的多个管道配置触发器时，正则表达式筛选器很有用。
- 提供文件排除以防止管道触发。当提交中的所有文件与排除路径或正则表达式中指定的文件匹配时，不会触发管道。
- **排除优先**打开时，可确保不会触发管道。即使提交中的任何文件与排除路径或正则表达式中指定的文件匹配，也不会触发管道。**排除优先**默认设置为关闭。

如果条件同时满足文件包含和文件排除，则不会触发管道。

在以下示例中，文件包含和文件排除都是触发器的条件。



- 对于文件包含，提交对 `runtime/src/main/a.java` 或任意 java 文件的任何更改都将触发事件配置中配置的管道。
- 对于文件排除，仅提交同时对这两个文件的更改不会触发事件配置中配置的管道。

10 单击**新建配置**。

a 对于 Gerrit 事件，选择**修补程序集已创建**、**草稿已发布**或**更改已合并**。或者，要绕过 Gerrit 直接推送到 Git，选择**直接 Git 推送**。

b 选择将触发的管道。

如果管道包括自定义添加的输入参数，则“输入参数”列表将显示参数和值。您可以输入将通过触发器事件传递到管道的输入参数的值。或者，也可以将这些值留空，或使用默认值。

注 如果定义了默认值：

- 为输入参数输入的任何值都将替代在管道模型中定义的默认值。
- 如果管道模型中的参数值发生更改，则触发器配置中的默认值不会更改。

有关 Gerrit 触发器的自动插入输入参数的信息，请参见必备条件。

c 对于**修补程序集已创建**、**草稿已发布**和**更改已合并**，默认情况下某些操作会与标签一起出现。可以更改标签或添加注释。然后，当管道运行时，标签或注释将作为对该管道**执行的操作**显示在**活动**选项卡上。

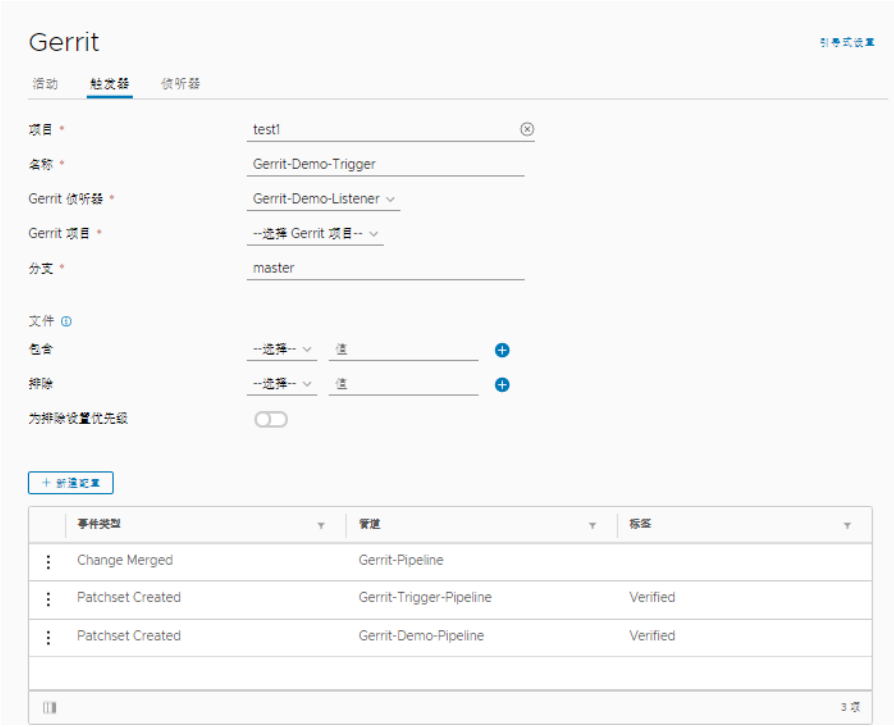
通过 Gerrit 事件配置，可以通过对成功注释或失败注释使用变量来输入注释。例如：`${var.success}` 和 `${var.failure}`。

d 单击**保存**。

要为多个管道添加多个触发器事件，请再次单击**新建配置**。

在以下示例中，您可以看到三个管道的事件：

- 如果 Gerrit 项目中发生**更改已合并**事件，则会触发名为 **Gerrit-Pipeline** 的管道。
- 如果 Gerrit 项目中发生**修补程序集已创建**事件，则会触发名为 **Gerrit-Trigger-Pipeline** 和 **Gerrit-Demo-Pipeline** 的管道。



11 单击**创建**。

Gerrit 触发器在**触发器**选项卡上显示为新的卡视图，并默认设置为**已禁用**。

12 在触发器卡中，单击**启用**。

启用触发器后，可以使用 Gerrit 侦听器，该侦听器将开始监控 Gerrit 项目分支上发生的事件。

要创建一个触发器（具有相同的文件包含条件或文件排除条件，但存储库不同于创建触发器时所包含的存储库），请在触发器卡视图上单击**操作 > 克隆**。然后，在克隆的触发器上，单击**打开**，然后更改参数。

结果

恭喜！您已成功配置在三个不同管道上具有两个事件的 Gerrit 触发器。

后续步骤

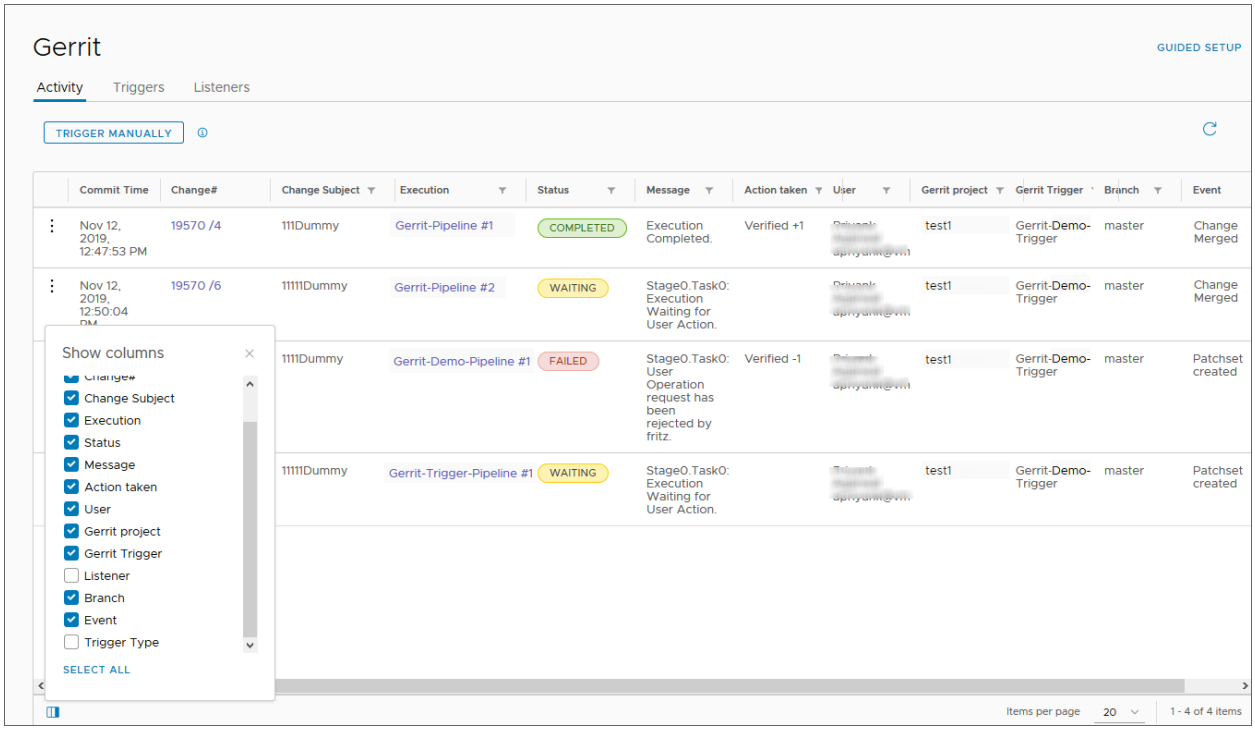
在 Gerrit 项目中提交代码更改之后，在 vRealize Automation Code Stream 的**活动**选项卡中查看 Gerrit 事件。验证活动列表是否包含与触发器配置中的每个管道执行对应的条目。

发生事件时，只能运行 Gerrit 触发器中与该特定事件类型相关的管道。在此示例中，如果创建修补程序集，只会运行 **Gerrit-Trigger-Pipeline** 和 **Gerrit-Demo-Pipeline**。

活动选项卡上列中的信息描述了每个 Gerrit 触发器事件。可以通过单击表下方的列图标选择显示的列。

- 如果触发器是直接 Git 推送，则**更改主题**和**执行**列为空。
- **Gerrit 触发器**列显示创建事件的触发器。
- 默认情况下，**侦听器**列处于关闭状态。选择该列后，该列将显示接收事件的 Gerrit 侦听器。单个侦听器可能显示与多个触发器关联。

- 默认情况下，**触发器类型**列处于关闭状态。选择该列后，该列将触发器类型显示为“自动”或“手动”。
- 其他列包括**提交时间**、**更改号码**、**状态**、**消息**、**执行的操作**、**用户**、**Gerrit 项目**、**分支**和**事件**。



要控制管道运行完成或失败的活动，请单击“活动”屏幕上任何条目左侧的三个点。

- 如果由于管道模型错误或其他问题而导致管道无法运行，请更正错误，然后选择**重新运行**以再次运行管道。
- 如果由于网络连接问题或其他问题而导致管道无法运行，请选择**继续**以重新启动同一管道执行，从而节省了运行时间。
- 使用**查看执行**可打开管道执行视图。请参见如何运行管道和查看结果。
- 使用**删除**从“活动”屏幕中删除条目。

如果 Gerrit 事件无法触发管道，可以单击**手动触发**，然后选择 Gerrit 触发器，输入“更改 ID”，并单击**运行**。

监控 vRealize Automation Code Stream 中的管道

8

作为 vRealize Automation Code Stream 管理员或开发人员，您需要在 vRealize Automation Code Stream 中深入了解管道的性能。您需要了解管道将代码从开发环境发布到测试环境再发布到生产环境这整个流程中的效率。

要深入了解，可以使用 vRealize Automation Code Stream 仪表板监控管道执行的趋势和结果。您可以使用默认管道仪表板监控单个管道，或者创建自定义仪表板以监控多个管道。

- 管道衡量指标包括平均时间等统计信息，可在管道仪表板上查看。
- 要查看多个管道的衡量指标，请使用自定义仪表板。

本章讨论了以下主题：

- vRealize Automation Code Stream 中管道仪表板显示的内容
- 如何在 vRealize Automation Code Stream 中使用自定义仪表板跟踪管道的关键绩效指标

vRealize Automation Code Stream 中管道仪表板显示的内容

管道仪表板是已运行的特定管道的结果视图，例如趋势、主要故障和成功的更改。vRealize Automation Code Stream 会在您创建管道时创建管道仪表板。

该仪表板包含显示管道执行结果的小组件。

管道“执行状态计数”小组件

可以查看管道在一段时间内的执行总数，并按状态分组：已完成、失败或已取消。要查看管道执行状态在较长时间或较短时间内的变化情况，请更改显示的持续时间。

管道“执行统计信息”小组件

管道执行统计信息包括一段时间内管道恢复、交付或失败的平均时间。

以下状态适用于所有管道执行：

- 已完成
- 失败
- 正在等待
- 正在运行

- 已取消
- 已排队
- 未启动
- 正在回滚
- 回滚已完成
- 回滚失败
- 已暂停

表 8-1. 测量平均时间

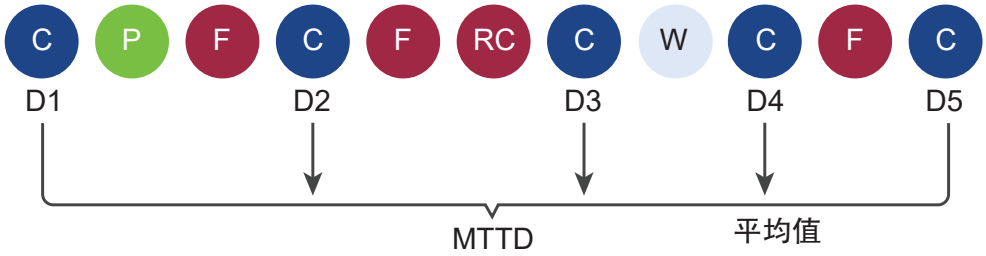
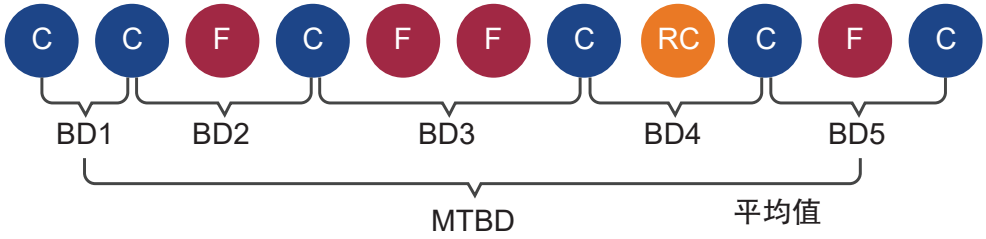
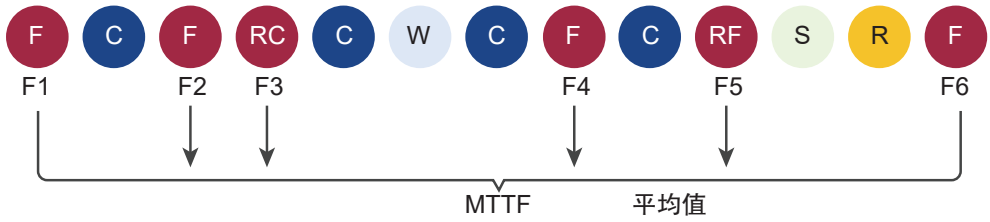
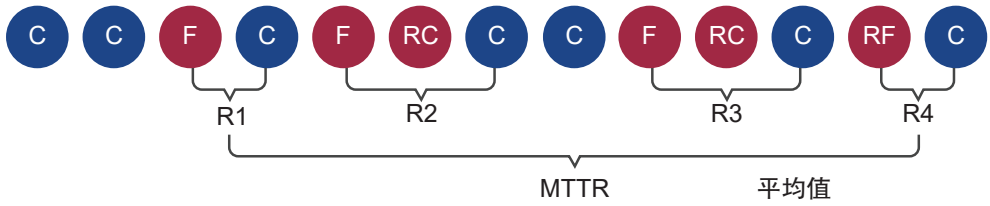
测量对象...	含义...
平均 CI	持续集成阶段所用的平均时间，按 CI 任务类型中的时间测量。
平均交付时间 (MTTD)	一段时间内所有已完成运行的平均持续时间。D1、D2 等是交付每个已完成运行的时间。 
平均交付间隔 (MTBD)	一段时间内两次成功交付之间的平均所用时间。两次连续已完成运行之间所用的时间是两次成功交付（如 BD1、BD2 等）之间的时间。MTBD 指示生产环境更新的频率。 

表 8-1. 测量平均时间 （续）

测量对象...	含义...
平均故障间隔 (MTTF)	一段时间内以失败、回滚已完成或回滚失败状态结束的运行的平均持续时间。F1、F2 等是运行以失败、回滚已完成或回滚失败结束的时间。 
平均恢复时间 (MTTR)	一段时间内从故障中恢复的平均时间。从故障中恢复的时间是指最终状态为失败、回滚已完成或回滚失败的运行与紧接着下一次状态为完成的成功运行之间所经过的时间。R1、R2 等是指每次从失败或回滚失败的运行恢复所需的时间。 

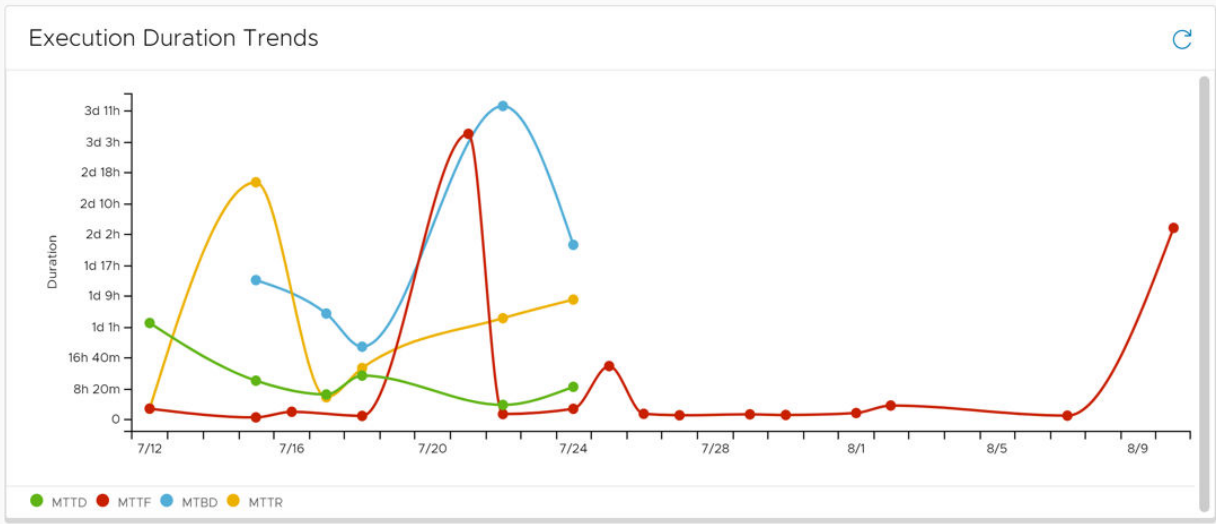
“前几个失败阶段”和“前几个失败任务”小组件

两个小组件显示管道中失败的前几个阶段和任务。每次测量都会针对每个管道和项目报告开发环境和开发后环境的失败数和百分比，计算每周或每月平均值。您可以查看前几个失败项，以便对发布自动化过程中的问题进行故障排除。

例如，您可以配置特定持续时间（如过去 7 天）的显示，并注意该时间段内的前几个失败任务。如果您在环境或管道中进行了更改并再次运行管道，然后检查较长持续时间（如过去 14 天）内的前几个失败任务，则前几个失败任务可能已更改。基于此结果，您将了解在发布自动化过程中所做的更改提高了管道执行的成功率。

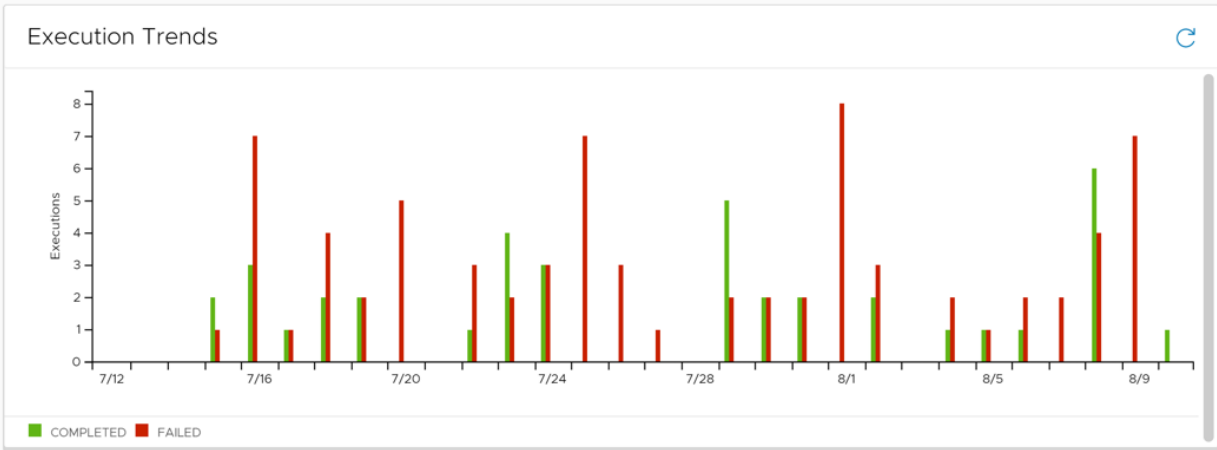
管道“执行持续时间趋势”小组件

管道执行持续时间趋势显示一段时间内的 MTTD、MTTF、MTBD 和 MTTR。



管道“执行趋势”小组件

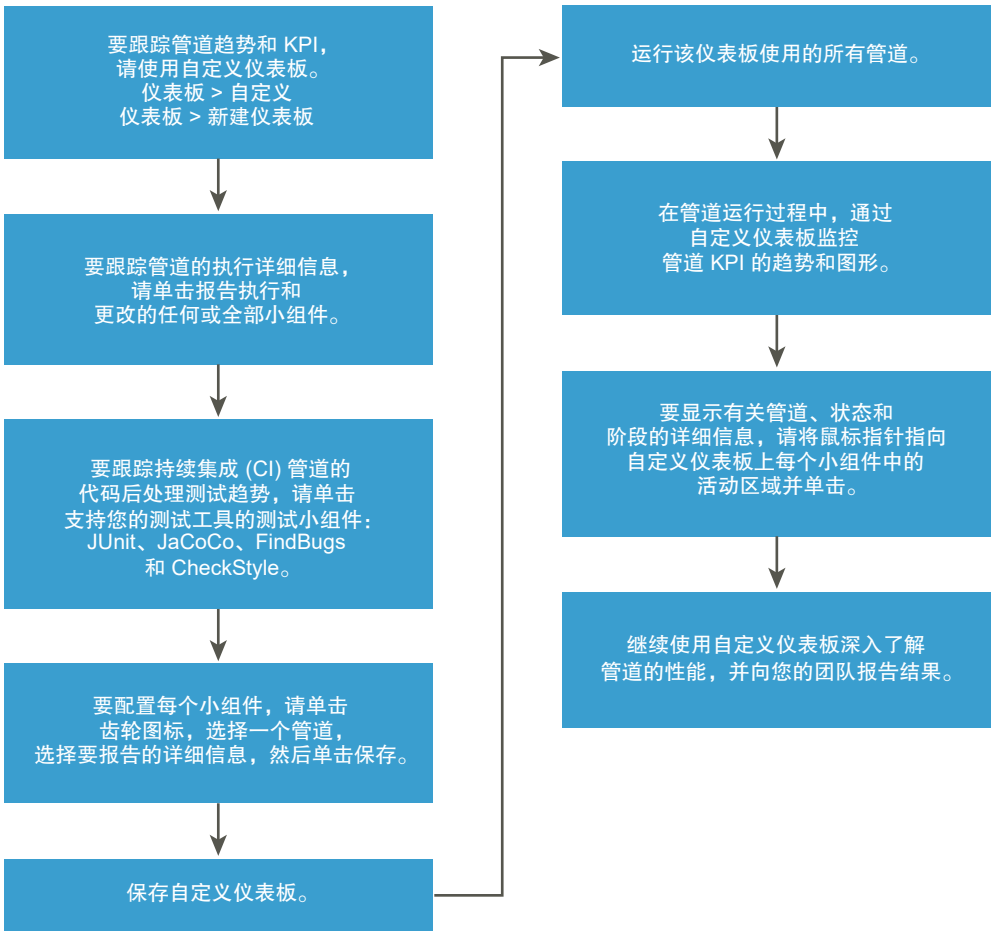
管道执行趋势显示一段时间内管道的每日运行总数，按状态分组。除当日之外，大多数每日汇总计数仅显示已完成和失败运行。



如何在 vRealize Automation Code Stream 中使用自定义仪表板跟踪管道的关键绩效指标

作为 vRealize Automation Code Stream 管理员或开发人员，您可以创建自定义仪表板以显示要查看的一个或多个已运行管道的结果。例如，您可以创建一个适用于整个项目的仪表板，其中包含从多个管道中收集的 KPI 和衡量指标。如果系统报告执行警告或故障，您可以使用仪表板对故障进行故障排除。

要使用自定义仪表板跟踪管道的趋势和关键绩效指标，您可以向仪表板添加小组件，并配置这些小组件以报告管道。



前提条件

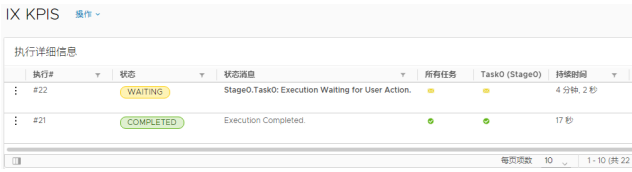
- 确认一个或多个管道存在。在用户界面中，单击**管道**。
- 对于要监控的管道，确认它们已成功运行。单击**执行**。

步骤

1 要创建自定义仪表板，请单击**仪表板 > 自定义仪表板 > 新建仪表板**。

2 要自定义仪表板以使其报告管道的特定趋势和关键绩效指标，请单击一个小组件。

例如，要显示有关管道状态、阶段、任务、运行持续时间以及运行该管道的用户的详细信息，请单击**执行持续时间**小组件。或者，对于持续集成 (CI) 管道，您可以使用适用于 JUnit、JaCoCo、FindBugs 和 CheckStyle 的小组件来跟踪趋势。



3 配置您添加的每个小组件。

   a 在小组件中，单击齿轮图标。

   b 选择一个管道，设置可用选项，并选择要显示的列。

   c 要保存小组件配置，请单击**保存**。

   d 要保存自定义仪表板，请单击**保存**，然后单击**关闭**。

4 要显示有关管道的更多信息，请单击小组件中的活动区域。

   例如，在**执行详细信息**小组件中，单击“状态”列中的条目可显示有关管道执行的更多信息。或者，在**最新的成功更改**小组件中，单击活动链接可显示管道阶段和任务的摘要。

结果

恭喜！您已创建一个自定义仪表板，该仪表板可用于跟踪管道的趋势和 KPI。

后续步骤

继续在 vRealize Automation Code Stream 中监控管道的性能并与经理和团队共享结果，以便持续改进应用程序发布流程。

了解有关 Code Stream 的更多信息

9

vRealize Automation Code Stream 管理员和开发人员可通过多种方法详细了解 vRealize Automation Code Stream 及其功能。

您可以使用本文档了解有关管道及其执行、如何添加端点、如何添加项目等的更多信息。

了解角色提供的相应权限。了解如何使用受限制资源，以及指定管道需要批准。请参见如何管理 vRealize Automation Code Stream 中的用户访问和批准。

通过发现特定作业或组件在管道、执行或端点中的位置来了解搜索所起的作用。

本章讨论了以下主题：

- 什么是 vRealize Automation Code Stream 中的搜索
- 供 vRealize Automation Code Stream 管理员和开发人员使用的更多资源

什么是 vRealize Automation Code Stream 中的搜索

可以使用搜索来查找特定项或其他组件所在的位置。例如，您可能想要搜索已激活或已停用的管道，因为已停用的管道无法运行。

可以搜索哪些内容

可以在以下位置搜索：

- 项目
- 端点
- 管道
- 执行
- 管道仪表板和自定义仪表板
- Gerrit 触发器和服务器
- Git Webhook
- Docker Webhook

可以在以下位置执行基于列的筛选器搜索：

- 用户操作

- 变量
- Gerrit、Git 和 Docker 触发器活动

可以在每个触发器的**活动**页面中执行基于网格的筛选器搜索。

搜索的工作原理是什么

搜索条件因所在的页面而异。每个页面具有不同的搜索条件。

搜索位置	用于执行搜索的条件
管道仪表板	项目、名称、说明、标记和链接
自定义仪表板	项目、名称、说明和链接（仪表板上的项的 UUID）
执行	名称、注释、原因、标记、索引、状态、项目、显示、执行者、由我执行、链接（执行 UUID）、输入参数、输出参数或使用以下格式的状态消息：`<key>:<value>`
管道	名称、说明、状态、标记、创建者、由我创建、更新者、由我更新和项目
项目	名称、说明
端点	名称、说明、类型、更新者和项目
Gerrit 触发器	名称、状态和项目
Gerrit 服务器	名称、服务器 URL 和项目
Git Webhook	名称、服务器类型、存储库、分支和项目

其中：

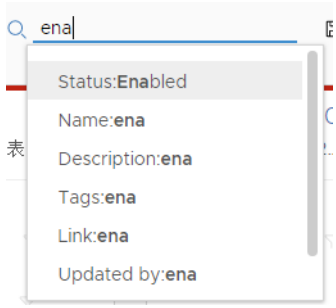
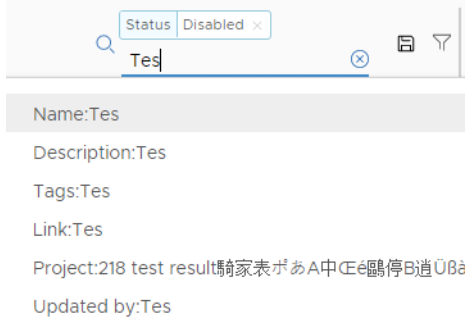
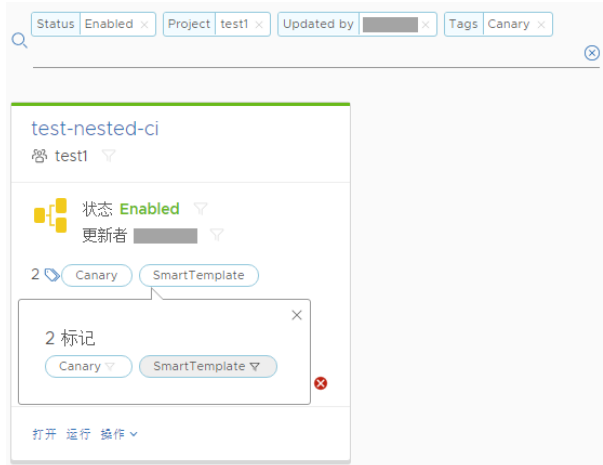
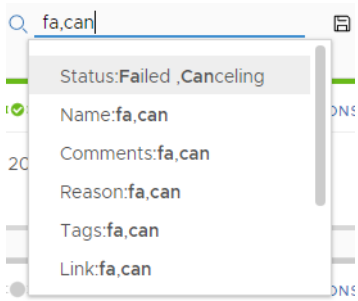
- 链接是管道、执行或仪表板上的小组件的 UUID。
- 输入参数、输出参数、状态消息表示形式和示例包括：
  - 表示形式：`input.<inputKey>:<inputValue>`

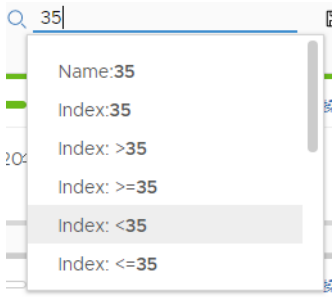
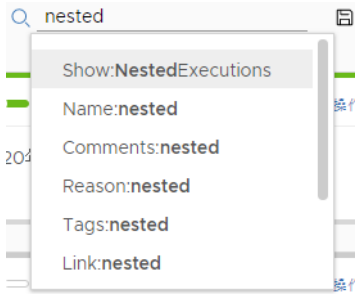
    示例：**`input.GERRIT_CHANGE_OWNER_EMAIL:joe_user`**
  - 表示形式：`output.<outputKey>:<outputValue>`

    示例：**`output.BuildNo:29`**
  - 表示形式：`statusMessage:<value>`

    示例：**`statusMessage:Execution failed`**
- 状态取决于搜索页面。
  - 对于执行，可能的值包括：已完成、失败、回滚失败或已取消。
  - 对于管道，可能状态值包括：已启用、已禁用或已发布。
  - 对于触发器，可能状态值包括：已启用或已禁用。
- 由我执行、由我创建或由我更新中的我是指已登录的用户。

搜索显示在每个有效页面的右上方。开始在搜索框中键入内容时， vRealize Automation Code Stream 将获知页面的上下文并提供搜索选项建议。

可以使用的搜索方法	如何输入
键入搜索参数的一部分。 例如，要添加列出所有已启用管道的状态筛选器，请键入 **ena**。	
要减少找到的项数，请添加筛选器。 例如，键入 **Tes** 可以添加名称筛选器。该筛选器作为 AND 条件与现有**状态: 禁用**筛选器结合使用，以便仅显示已禁用且名称包含 **Tes** 的管道。 添加另一个筛选器时，将显示其余选项：**名称**、**描述**、**标记**、**链接**、**项目**和**更新者**。	
要减少显示的项数，请单击管道或管道执行属性上的筛选器图标。 ■ 对于管道，**状态**、**标记**、**项目**和**更新者**各有一个筛选器图标。 ■ 对于执行，**标记**、**执行者**和**状态消息**各有一个筛选器图标。 例如，在管道卡视图上，单击 **SmartTemplate** 标记所对应的筛选器图标，以将该筛选器添加到以下项所对应的现有筛选器中：**状态: Enabled**、**项目: test**、**更新者: user** 和**标记: Canary**。	
使用逗号分隔符以包含处于两种执行状态的所有项。 例如，键入 **fa,can** 可以创建一个作为 OR 条件工作的状态筛选器，以列出所有失败的或已取消的执行。	

可以使用的搜索方法	如何输入
键入一个数字以包含某个索引范围内的所有项。 例如，键入 **35** 并选择 **<** 可以列出索引编号小于 35 的所有执行。	
建模为任务的管道将成为嵌套执行，默认情况下不会与所有执行一起列出。 要显示嵌套执行，请键入 **nested** 并选择**显示**筛选器。	

如何保存喜爱的搜索

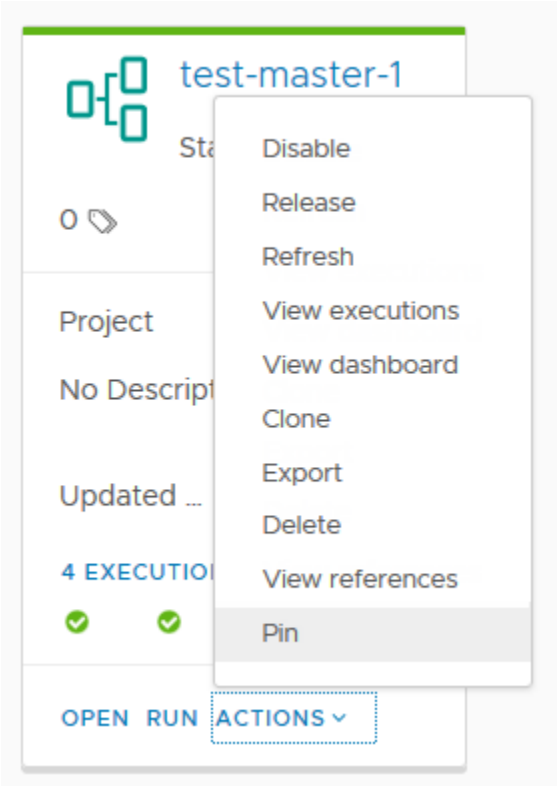
通过单击搜索区域旁边的磁盘图标，可以保存喜爱的搜索以在每个页面中使用。

- 通过键入搜索参数并单击图标为搜索指定名称（例如 **my enabled**），保存搜索。
- 保存搜索后，单击图标即可访问该搜索。还可以选择**管理**来重命名和删除搜索或在保存的搜索列表中移动搜索。

搜索与用户名关联，并且仅在搜索适用于的页面中显示。例如，如果您在管道页面中保存了适用于**状态: enabled** 筛选器且名为 **my enabled** 的搜索，则即使**状态: enabled** 搜索对触发器有效，**my enabled** 搜索在 Gerrit 触发器页面中也不可用。

是否可以保存喜爱的管道

如果您有喜爱的管道或仪表板，则可以将其固定，以使其始终显示在管道页面或仪表板页面的顶部。在管道卡视图中，单击**操作 > 加为看板项**。



供 vRealize Automation Code Stream 管理员和开发人员使用的更多资源

作为 vRealize Automation Code Stream 管理员或开发人员，您可以了解有关 vRealize Automation Code Stream 的更多信息。

表 9-1. 供管理员使用的更多资源

要了解...	参见以下资源...
管理员可以使用 vRealize Automation Code Stream 的其他方式： ■ 配置管道以自动测试和发布云本机应用程序。 ■ 在生产前的整个测试过程中自动执行并测试开发人员源代码。 ■ 为开发人员配置管道，以便开发人员在将更改提交到主分支之前对更改进行测试。 ■ 跟踪管道关键衡量指标。	vRealize Automation Code Stream ■ vRealize Automation 文档 ■ vRealize Automation 产品网站 VMware 练习 ■ 使用 vRealize Automation 社区。 ■ 使用 VMware 学习区域。 ■ 搜索 VMware 博客。 ■ 尝试 VMware 练习实验室。

表 9-2. 供开发人员使用的更多资源

要了解...	参见以下资源...
开发人员可以使用 vRealize Automation Code Stream 的其他方式： ■ 使用公共和专用注册表映像为新的应用程序或服务构建环境。 ■ 设置开发环境，以便您可以从最新的稳定构建创建分支。 ■ 使用最新的代码更改和工件来更新开发环境。 ■ 对照其他从属服务的最新的稳定构建来测试未提交的代码更改。 ■ 当提交给主 CICD 管道的更改使其他服务中断时，会收到通知。	vRealize Automation Code Stream ■ vRealize Automation 文档 ■ vRealize Automation 产品网站 VMware 练习 ■ 使用 vRealize Automation 社区。 ■ 使用 VMware 学习区域。 ■ 搜索 VMware 博客。 ■ 尝试 VMware 练习实验室。