

# Uso e gerenciamento do vRealize Automation Code Stream

14 DE DEZEMBRO DE 2022  
vRealize Automation 8.7

Você pode encontrar a documentação técnica mais atualizada no site da VMware, em:

<https://docs.vmware.com/br/>

**VMware, Inc.**  
3401 Hillview Ave.  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

**VMware Brasil**  
Rua Surubim, 504 4º andar CEP 04571-050  
Cidade Monções  
São Paulo  
SÃO PAULO: 04571-050  
Brasil  
Tel: +55 11 55097200  
Fax: + 55. 11. 5509-7224  
[www.vmware.com/br](http://www.vmware.com/br)

Copyright © 2022 VMware, Inc. Todos os direitos reservados. [Informações sobre direitos autorais e marca registrada.](#)

# Conteúdo

- 1 O que é o Code Stream e como ele funciona 5**
- 2 Configuração para modelar meu processo de liberação 10**
  - Como adicionar um projeto 15
  - Como gerenciar o acesso do usuário e as aprovações 16
  - O que são operações do usuário e aprovações 25
- 3 Como criar e usar pipelines 27**
  - Como executar um pipeline e ver os resultados 30
  - Que tipos de tarefas estão disponíveis 35
  - Como usar associações de variáveis em pipelines 41
  - Como usar associações de variáveis em uma tarefa de condição para executar ou parar um pipeline 51
  - Que variáveis e expressões eu posso usar ao associar tarefas de pipeline 54
  - Como enviar notificações sobre meu pipeline 72
  - Como criar um tíquete do Jira quando uma tarefa de pipeline falhar 74
  - Como reverter minha implantação 77
- 4 Como planejar compilar, integrar e entregar seu código de forma nativa 84**
  - Como configurar o espaço de trabalho do pipeline 84
  - Como planejar uma compilação nativa de CICD antes de usar o modelo de pipeline inteligente 88
  - Como planejar uma compilação nativa de CI antes de usar o modelo de pipeline inteligente 95
  - Como planejar uma compilação nativa de CD antes de usar o modelo de pipeline inteligente 96
  - Como planejar uma compilação nativa de CICD antes de adicionar tarefas manualmente 98
  - Como planejar uma reversão 104
- 5 Tutoriais 107**
  - Como integrar continuamente o código do meu repositório GitHub ou GitLab ao pipeline 108
  - Como automatizar a liberação de um aplicativo implantado a partir de um modelo de nuvem YAML 113
  - Como automatizar a liberação de um aplicativo para um cluster do Kubernetes 121
  - Como implantar meu aplicativo na implantação Azul-Verde 129
  - Como integrar minhas próprias ferramentas de compilação, teste e implantação 134
  - Como usar as propriedades de recursos de uma tarefa de modelo de nuvem na minha próxima tarefa 146
  - Como usar uma REST API para integração com outros aplicativos 150
  - Como alavancar o pipeline como código 155

## **6 Conectando-se a endpoints 161**

- O que são endpoints 161
- Como integrar ao Jenkins 164
- Como integrar ao Git 171
- Como integrar ao Gerrit 173
- Como integrar ao vRealize Orchestrator 177

## **7 Como disparar pipelines 183**

- Como usar o gatilho do Docker para executar um pipeline de entrega contínua 183
- Como usar o gatilho Git para executar um pipeline 192
- Como usar o gatilho Gerrit para executar um pipeline 200

## **8 Como monitorar pipelines 209**

- O que o painel do pipeline está mostrando 209
- Como usar painéis personalizados para rastrear indicadores-chave de desempenho 212

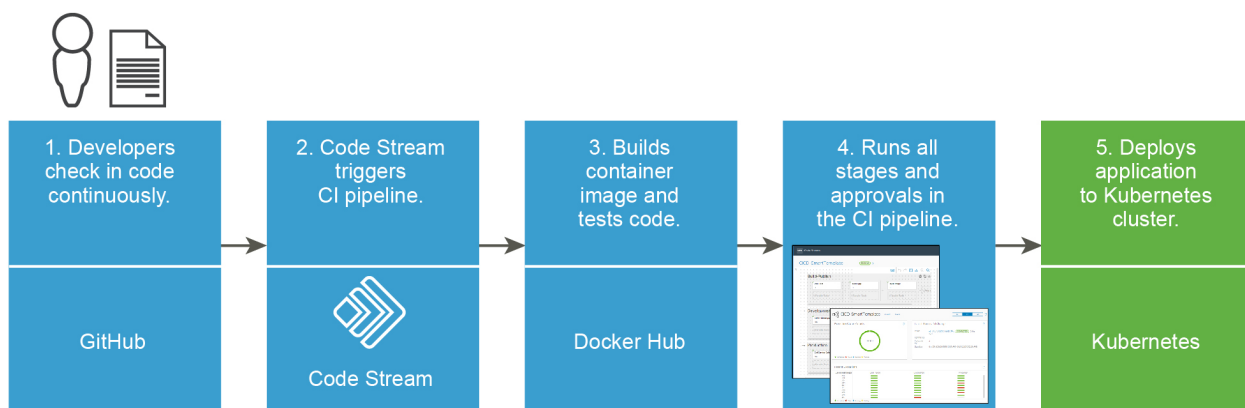
## **9 Saiba mais 215**

- O que é pesquisa 215
- Mais recursos para administradores e desenvolvedores 221

# O que é o Code Stream e como ele funciona

1

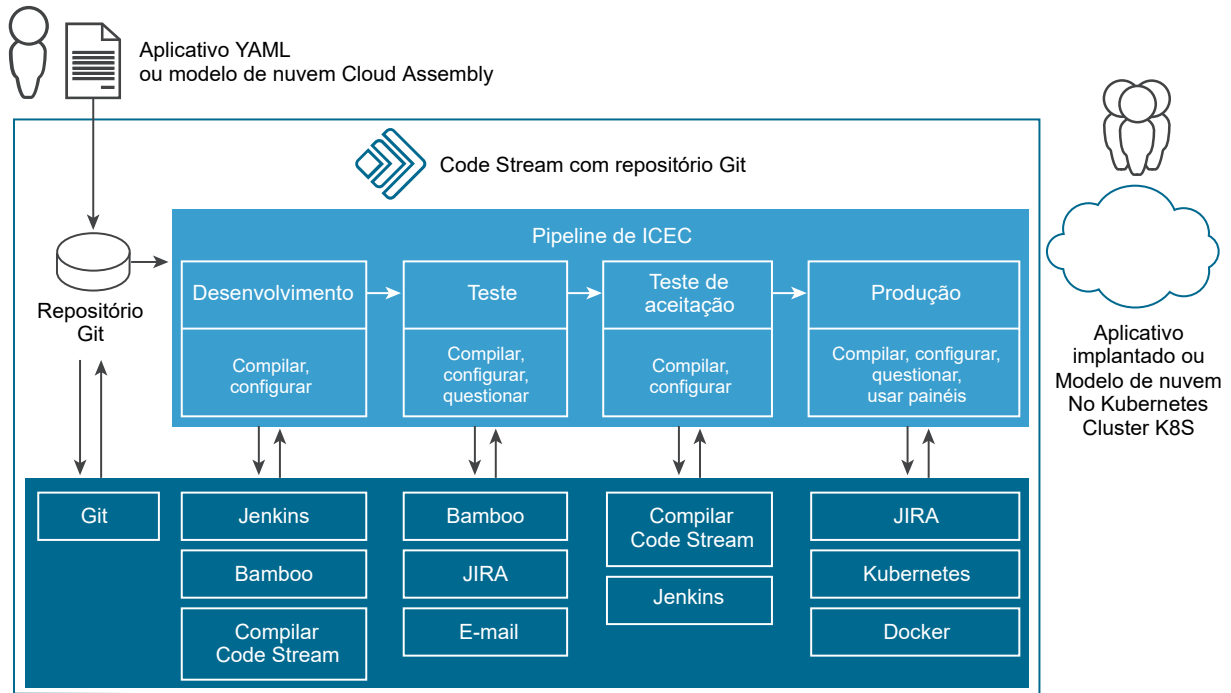
O vRealize Automation Code Stream™ é uma ferramenta de integração e entrega contínua (CI/CD). Ao criar pipelines que modelam o processo de liberação de software no seu ciclo de vida de DevOps, você constrói a infraestrutura de código que entrega seu software de maneira rápida e contínua.



Ao usar o Code Stream para entregar seu software, integre duas das partes mais importantes de seu ciclo de vida do DevOps: seu processo de liberação e suas ferramentas de desenvolvedor. Após a configuração inicial, que integra o Code Stream às suas ferramentas de desenvolvimento existentes, os pipelines automatizam todo o ciclo de vida de DevOps.

A partir do vRealize Automation 8.2, blueprints se chamam VMware Cloud Templates.

Você cria um pipeline que compila, testa e libera seu software. O Code Stream usa esse pipeline para conduzir seu software desde o repositório do código-fonte, passando pela fase de testes, até a produção.



Você pode saber mais sobre como planejar a integração contínua e pipelines de entrega contínua em [Capítulo 4 Planejamento para compilar, integrar e entregar seu código de forma nativa no Code Stream](#).

## Como os administradores do Code Stream usam o Code Stream

Como administrador, você cria endpoints e garante que as instâncias de trabalho estejam disponíveis para desenvolvedores. É possível criar, disparar e gerenciar pipelines e muito mais. Você tem a função *Administrator*, conforme descrito em [Como gerenciar o acesso do usuário e as aprovações no Code Stream](#).

Tabela 1-1. Como os administradores do Code Stream oferecem suporte aos desenvolvedores

Para oferecer suporte aos desenvolvedores...	Veja o que é possível fazer...
Fornecer e gerenciar ambientes.	<p>Criar ambientes para os desenvolvedores testarem e implantarem seu código.</p> <ul style="list-style-type: none"> <li>■ Rastrear o status e enviar notificações por e-mail.</li> <li>■ Manter seus desenvolvedores produtivos garantindo que seus ambientes funcionem continuamente.</li> </ul> <p>Para saber mais, consulte <a href="#">Mais recursos para administradores e desenvolvedores do Code Stream</a>.</p> <p>Consulte também <a href="#">Capítulo 5 Tutoriais para usar o Code Stream</a>.</p>
Forneça endpoints.	Certifique-se de que os desenvolvedores tenham instâncias de trabalho de endpoints que possam se conectar aos pipelines.
Fornecer integrações a outros serviços.	<p>Certifique-se de que as integrações a outros serviços estejam funcionando.</p> <p>Para saber mais, consulte a <a href="#">documentação do VMware Cloud Services</a>.</p>

**Tabela 1-1. Como os administradores do Code Stream oferecem suporte aos desenvolvedores (continuação)**

Para oferecer suporte aos desenvolvedores...	Veja o que é possível fazer...
Criar pipelines.	<p>Crie pipelines que modelem os processos de liberação.</p> <p>Para saber mais, consulte <a href="#">Capítulo 3 Criando e usando pipelines no Code Stream</a>.</p>
Disparar pipelines.	<p>Certifique-se de que os pipelines sejam executados quando eventos ocorrerem.</p> <ul style="list-style-type: none"> <li>■ Para disparar um pipeline autônomo de entrega contínua (CD), sempre que um artefato de compilação for criado ou atualizado, use o gatilho do Docker.</li> <li>■ Para disparar um pipeline quando um desenvolvedor confirmar alterações no seu código, use o gatilho Git.</li> <li>■ Para disparar um pipeline quando os desenvolvedores revisarem o código, fizerem uma mesclagem e mais, use o gatilho Gerrit.</li> <li>■ Para executar um pipeline autônomo de entrega contínua (CD), sempre que um artefato de compilação for criado ou atualizado, use o gatilho do Docker.</li> </ul> <p>Para saber mais, consulte <a href="#">Capítulo 7 Disparando pipelines no Code Stream</a>.</p>
Gerenciar pipelines e aprovações.	<p>Mantenha-se atualizado sobre pipelines.</p> <ul style="list-style-type: none"> <li>■ Visualize o status dos pipelines e veja quem os executou.</li> <li>■ Exiba as aprovações em execuções do pipeline e gerencie as aprovações para execuções do pipeline ativas e inativas.</li> </ul> <p>Para saber mais, consulte <a href="#">O que são operações do usuário e aprovações no Code Stream</a>.</p> <p>Consulte também <a href="#">Como usar painéis personalizados para rastrear indicadores-chave de desempenho para o meu pipeline no Code Stream</a>.</p>
Monitorar os ambientes de desenvolvedor.	<p>Crie painéis personalizados que monitoram o status do pipeline, suas tendências, métricas e indicadores principais. Use esses painéis personalizados para monitorar pipelines que são aprovados ou reprovados em ambientes de desenvolvimento. Você também pode identificar e relatar recursos subutilizados e liberar recursos.</p> <p>Consulte também:</p> <ul style="list-style-type: none"> <li>■ Há quanto tempo um pipeline foi executado antes de ser bem-sucedido.</li> <li>■ Há quanto tempo um pipeline aguardou a aprovação e notificar o usuário que deve aprová-lo.</li> <li>■ Estágios e tarefas que falham com mais frequência.</li> <li>■ Estágios e tarefas que levam mais tempo para serem executados.</li> <li>■ Liberar as equipes de desenvolvimento têm em andamento.</li> <li>■ Aplicativos que foram bem-sucedidos ao serem implantados e liberados.</li> </ul> <p>Para saber mais, consulte <a href="#">Capítulo 8 Monitorando pipelines no Code Stream</a>.</p>
Solucionar problemas.	<p>Solucionar problemas e resolver falhas de pipeline em ambientes de desenvolvimento.</p> <ul style="list-style-type: none"> <li>■ Identifique e resolva problemas em ambientes de integração contínua e entrega contínua (CICD).</li> <li>■ Use painéis de pipeline e crie painéis personalizados para ver mais. Consulte <a href="#">Capítulo 8 Monitorando pipelines no Code Stream</a>.</li> </ul> <p>Consulte também <a href="#">Capítulo 2 Como configurar o Code Stream para modelar meu processo de liberação</a>.</p>

O Code Stream faz parte do VMware Cloud Services.

- Use o Cloud Assembly para implantar modelos de nuvem.

- Use o Service Broker para obter modelos de nuvem do catálogo.

Para conhecer outras coisas que você pode fazer, consulte a [Documentação do VMware vRealize Automation](#).

## Como os desenvolvedores usam o Code Stream

Como desenvolvedor, você usa o Code Stream para compilar e executar pipelines e monitorar a atividade do pipeline nos painéis. Você tem a função `User`, conforme descrito em [Como gerenciar o acesso do usuário e as aprovações no Code Stream](#).

Depois de executar um pipeline, você desejará saber:

- Se o seu código foi bem-sucedido em todos os estágios do pipeline. Para descobrir, observe os resultados nas execuções de pipeline.
- O que fazer se o pipeline falhar e o que causou a falha. Para descobrir, observe os principais erros nos painéis do pipeline.

Tabela 1-2. Desenvolvedores que usam o Code Stream

Para integrar e liberar seu código	Faça o seguinte
Compilar pipelines.	Testar e implantar seu código. Atualizar seu código quando um pipeline falhar.
Conectar seu pipeline a endpoints.	Conectar as tarefas do seu pipeline a endpoints, como um repositório GitHub.
Executar pipelines.	Adicione uma tarefa de aprovação da operação do usuário para que outro usuário possa aprovar seu pipeline em pontos específicos.
Exibir painéis.	Exiba os resultados no painel do pipeline. É possível visualizar tendências, histórico, falhas e muito mais.

Para obter mais informações sobre como começar, consulte [Introdução ao VMware Code Stream](#).

## Encontre conteúdo de documentação adicional no painel Suporte no produto

Se você não encontrar as informações necessárias aqui, poderá obter mais ajuda no produto. 

- Clique e leia as sinalizações e as dicas de ferramentas na interface do usuário do para obter as informações específicas do contexto de que você precisa e exatamente quando precisar delas.
- Abra o painel de suporte no produto e leia os tópicos que aparecem para a página da interface do usuário ativa. Você também pode pesquisar no painel para obter respostas às perguntas.

Mais sobre webhooks



Você pode criar vários webhooks para ramificações diferentes usando o mesmo endpoint Git e fornecendo diferentes valores para o nome da ramificação na página de configuração do webhook. Para criar outro webhook para outra ramificação no mesmo repositório Git, não é necessário clonar o endpoint Git várias vezes para várias ramificações. Em vez disso, forneça o nome da ramificação no webhook, o que permite reutilizar o endpoint Git. Se a ramificação no webhook Git for a mesma que a ramificação no endpoint, você não precisará fornecer o nome da ramificação na página do webhook Git.

# Como configurar o Code Stream para modelar meu processo de liberação

## 2

Para modelar seu processo de liberação, crie um pipeline que represente os estágios, as tarefas e as aprovações que você normalmente usa para liberar seu software. Em seguida, o Code Stream automatiza o processo que compila, testa, aprova e implanta seu código.

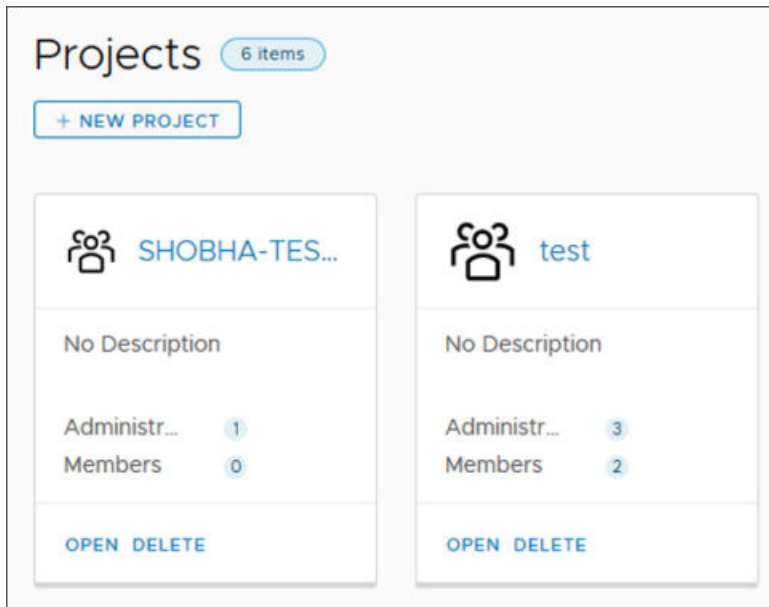
Agora que você tem tudo para modelar seu processo de liberação de software, veja como fazer isso no Code Stream.

### Pré-requisitos

- Verifique se já existem endpoints disponíveis. No Code Stream, clique em **Endpoints**.
- Saiba mais sobre as maneiras nativas de criar e implantar seu código. Consulte [Capítulo 4 Planejamento para compilar, integrar e entregar seu código de forma nativa no Code Stream](#).
- Determine se alguns dos recursos que serão usados no seu pipeline devem ser marcados como restritos. Consulte [Como gerenciar o acesso do usuário e as aprovações no Code Stream](#).
- Se você tiver a função de usuário ou expectador em vez da função de administrador, determine quem é o administrador da sua instância do Code Stream.

### Procedimentos

- 1 Examine os projetos disponíveis no Code Stream e selecione um que seja adequado para você.
  - Se nenhum projeto estiver visível, peça a um administrador do Code Stream para criar um projeto e torná-lo um membro desse projeto. Consulte [Como adicionar um projeto no Code Stream](#).
  - Se você não for membro de nenhum dos projetos listados, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto.



- Adicione novos endpoints necessários para o pipeline.

Por exemplo, você pode precisar de Git, Jenkins, Compilação de Code Stream, Kubernetes e Jira.

- Crie variáveis para que você possa reutilizar valores em suas tarefas de pipeline.

Para restringir os recursos usados nos seus pipelines, como uma máquina host, use variáveis restritas. Você pode impedir o pipeline de continuar a ser executado até que outro usuário o aprove explicitamente.

Os administradores podem criar variáveis secretas e variáveis restritas. Os usuários podem criar variáveis secretas.

Você pode reutilizar uma variável quantas vezes desejar em vários pipelines. Por exemplo, uma variável que define uma máquina host pode ser `HostIPAddress`. Para usar a variável em uma tarefa de pipeline, insira `${var.HostIPAddress}`.

The screenshot shows the 'Variables' page with 3 items. The table below represents the data shown:

	Project	Name	Type	Value
⋮	Code Stream	Test	Regular	123
⋮	Code Stream	Test-Restricted	Restricted	*****
⋮	Code Stream	Test-Global-name	Secret	*****

- 4 Se você for administrador, marque quaisquer endpoints e variáveis que sejam vitais para seus negócios como recursos restritos.

Quando um usuário que não é administrador tenta executar um pipeline que inclui um recurso restrito, o pipeline é interrompido na tarefa que usa esse recurso restrito. Em seguida, um administrador deve retomar o pipeline.

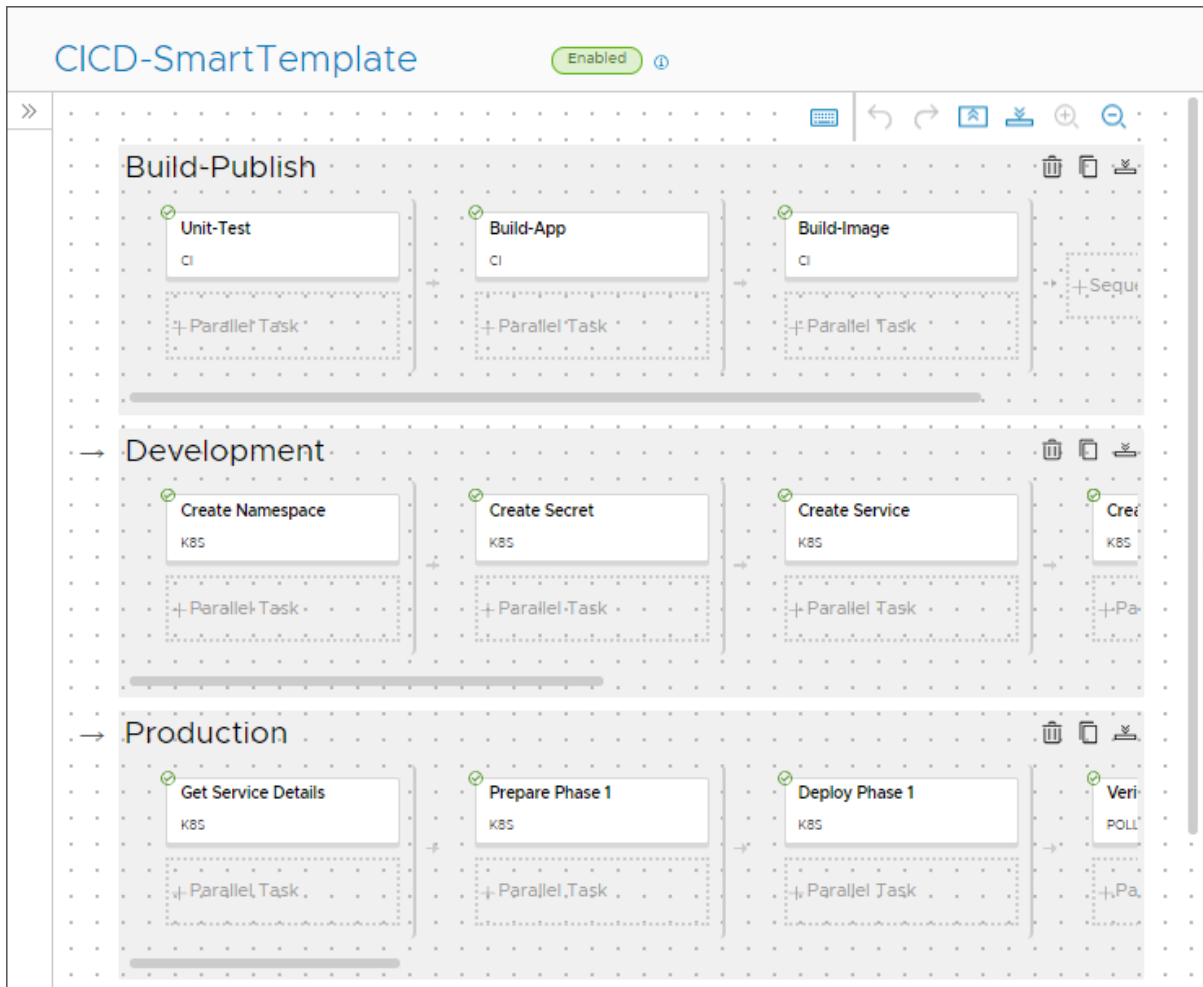
- 5 Planeje a estratégia de compilação para seu pipeline nativo de CICD, CI ou CD.

Antes de criar um pipeline que integre continuamente (CI) e implante continuamente (CD) seu código, planeje sua estratégia de compilação. O plano de compilação ajuda a determinar do que o Code Stream precisa para poder compilar, integrar, testar e implantar seu código de forma nativa.

<b>Como criar uma compilação nativa do Code Stream</b>	<b>Resultados nesta estratégia de compilação</b>
<b>Use um dos modelos de pipeline inteligentes.</b>	<ul style="list-style-type: none"> <li>■ Compila todos os estágios e tarefas para você.</li> <li>■ Clona o repositório de origem.</li> <li>■ Compila e testa seu código.</li> <li>■ Armazena seu código em contêineres para implantação.</li> <li>■ Preenche as etapas da tarefa de pipeline com base nas suas seleções.</li> </ul>
<b>Adicionar estágios e tarefas manualmente.</b>	Você adiciona estágios, adiciona tarefas e insere as informações que os preenchem.

- 6 Crie seu pipeline usando um modelo de pipeline inteligente ou adicione estágios e tarefas manualmente ao pipeline.

Em seguida, marque todos os recursos como restritos. Adicione aprovações onde necessário. Aplique qualquer variável regular, restrita ou secreta. Adicione quaisquer associações entre tarefas.



7 Valide, ative e execute o pipeline.

8 Exibir as execuções do pipeline.

**Executions** 280 items

NEW EXECUTION Search

- Demo-Jenkins... #95** **COMPLETED** Stages: ACTIONS
  - By kr on 09/11/2018 10:32 AM
  - Execution Completed.
  - Input : 8df0d9a1d365299f2...
  - Output : NA
- Demo-Jenkins... #94** **COMPLETED** Stages: ACTIONS
  - By kr on 09/11/2018 9:17 AM
  - Execution Completed.
  - Input : 6d82d079a8b8921a9...
  - Output : NA
- Demo-CICD-S... #51** **COMPLETED** Stages: ACTIONS
  - By dk on 09/11/2018 7:13 AM
  - Execution Completed.
  - Input : NA
  - Output : NA
- Demo-CICD-S... #50** **FAILED** Stages: ACTIONS
  - By dk on 09/11/2018 5:51 AM
  - Execution failed on task 'Production.Deploy Phase 1'. deployments...
  - Input : NA
  - Output : NA

9 Para rastrear o status e os principais indicadores de desempenho (KPIs), use os painéis de pipeline e crie painéis personalizados.

**CICD-SmartTemplate** CLONE BACK 1D 7D 14D

**Execution Status Counts**

Total: 1

● Completed ● Failed ● Running ● Waiting

**Latest Successful Change**

When CICD-SmartTemplate #46 **COMPLETED** a day ago

Comments -

Executed by d

Duration 6m 37s (09/06/2018 10:21 AM - 09/06/2018 10:29 AM)

**Recent Executions**

Execution#/Stages	Build-Publish	Development	Production
#46			
#45			
#44			
#43			
#42			
#41			
#40			
#39			
#38			
#37			

● Completed ● Failed ● Running ● Waiting

## Resultados

Você criou um pipeline que pode ser usado no projeto selecionado.

Você também pode exportar o YAML do pipeline e importá-lo e reutilizá-lo em outros projetos.

## Próximo passo

Saiba mais sobre os casos de uso que você pode querer aplicar no seu ambiente. Consulte [Capítulo 5 Tutoriais para usar o Code Stream](#).

# Como adicionar um projeto no Code Stream

Você cria um projeto e adiciona administradores e membros a ele. Os membros do projeto podem usar recursos como criar um pipeline e adicionar um endpoint. Para criar, excluir ou atualizar um projeto para uma equipe de desenvolvimento, você deve ser administrador do Code Stream.

Um projeto deve existir para que você possa criar um pipeline. Ao criar um pipeline, você seleciona um projeto que agrupa todas as informações do pipeline. As definições de endpoints e variáveis também dependem de um projeto existente.

## Pré-requisitos

- Verifique se você tem a função de administrador do Code Stream. Consulte [O que são funções no Code Stream](#).

Se você não tiver a função de administrador do Code Stream, mas for um administrador no Cloud Assembly, poderá usar a interface de usuário do Cloud Assembly para criar, atualizar ou excluir projetos. Consulte [Como adicionar um projeto para minha equipe de desenvolvimento do Cloud Assembly](#).

- Se você estiver adicionando grupos do Active Directory a projetos, verifique se configurou esses para a sua organização. Consulte [Como habilitar grupos do Active Directory no vRealize Automation para projetos](#). Se os grupos não forem sincronizados, eles não ficarão disponíveis quando você tentar adicioná-los a um projeto.

## Procedimentos

- 1 Selecione **Projetos** e clique em **Novo projeto**.
- 2 Digite o nome do projeto.
- 3 Clique em **Criar**.
- 4 Selecione o cartão para o projeto recém-criado e clique em **Abrir**.
- 5 Clique na guia **Usuários** e adicione usuários e atribua funções.
  - O administrador do projeto pode adicionar membros.
  - O membro do projeto com uma função de serviço pode usar serviços.
  - O expectador do projeto pode ver projetos, mas não pode criá-los, atualizá-los ou excluí-los.

Para obter mais informações sobre funções de projeto, consulte [Como gerenciar o acesso do usuário e as aprovações no Code Stream](#).

## 6 Clique em **Salvar**.

### Próximo passo

Adicione endpoints e pipelines que usam o projeto. Consulte [Capítulo 6 Como conectar o Code Stream aos endpoints](#) e [Capítulo 3 Criando e usando pipelines no Code Stream](#).

Depois de criar um pipeline, o nome do projeto que agrupa todas as informações do pipeline aparece nos cartões do pipeline e nos cartões de execução do pipeline.

## Como gerenciar o acesso do usuário e as aprovações no Code Stream

O Code Stream fornece várias maneiras de garantir que os usuários tenham a autorização e o consentimento apropriados para trabalhar com pipelines que liberam seus aplicativos de software.

Cada membro em uma equipe tem uma função atribuída, que fornece permissões específicas nos pipelines, endpoints e painéis, e a capacidade de marcar recursos como restritos.

As operações e aprovações dos usuários permitem que você controle quando um pipeline é executado e quando ele deve ser interrompido para uma aprovação. Sua função determina se é possível retomar um pipeline e executar pipelines que incluem variáveis ou endpoints restritos.

Use variáveis secretas para ocultar e criptografar informações confidenciais. Use a variável restrita para strings, senhas e URLs que devem ser ocultas e criptografadas e para restringir o uso em execuções. Por exemplo, use uma variável secreta para uma senha ou URL. Você pode usar variáveis secretas e restritas em qualquer tipo de tarefa no seu pipeline.

## O que são funções no Code Stream

Dependendo da sua função no Code Stream, é possível realizar determinadas ações e acessar determinadas áreas. Por exemplo, sua função pode permitir a criação, atualização e execução de pipelines. Ou, pode permitir apenas a exibição de pipelines.

Todas as ações, exceto as restritas significa que essa função tem permissão para executar ações de criação, leitura, atualização e exclusão em entidades, exceto para variáveis restritas e endpoints.



Tabela 2-1. Permissões de acesso em nível de Serviço e Projeto no Code Stream

Níveis de acesso	Funções do Code Stream				
	Administrador do Code Stream	Desenvolvedor do Code Stream	Executor do Code Stream	Espectador do Code Stream	Usuário do Code Stream
Acesso em nível de serviço do Code Stream	Todas as ações	Todas as ações, exceto as restritas	Ações de execução	Somente leitura	Nenhum
Acesso em nível de projeto: Administrador do Projeto	Todas as ações	Todas as ações	Todas as ações	Todas as ações	Todas as ações
Acesso em nível de projeto: Membro do Projeto	Todas as ações	Todas as ações, exceto as restritas	Todas as ações, exceto as restritas	Todas as ações, exceto as restritas	Todas as ações, exceto as restritas
Acesso em nível de projeto: Espectador do Projeto	Todas as ações	Todas as ações, exceto as restritas	Ações de execução	Somente leitura	Somente leitura

Os usuários que têm a função de administrador de projeto podem realizar todas as ações em projetos em que eles são um administrador de projeto.

Um administrador de projeto poderá criar, ler, atualizar e excluir pipelines, variáveis, endpoints, dashboards, gatilhos e iniciar um pipeline que inclua endpoints ou variáveis restritas se esses recursos estiverem no projeto em que o usuário é um Administrador de projeto.

Os usuários com a função de Visualizador de Serviços podem ver todas as informações disponíveis ao administrador. Eles não podem realizar nenhuma ação, a menos que um administrador os torne um administrador de projeto ou membro do projeto. Se o usuário for afiliado a um projeto, ele terá as permissões relacionadas à função. A função de visualizador de projeto não abrange as permissões da mesma forma que a função de administrador ou membro. Essa função é de somente leitura em todos os projetos.

Se você tiver permissões de leitura em um projeto, ainda poderá ver recursos restritos.

- Para ver endpoints restritos, que exibem um ícone de cadeado no cartão do endpoint, clique em **Configurar > Endpoints**.
- Para ver variáveis restritas e secretas, que exibem RESTRITO ou SECRETO na coluna **Tipo**, clique em **Configurar > Variáveis**.

Tabela 2-2. Recursos de funções de serviço do Code Stream

Contexto da interface do usuário	Recursos	Função Administrador do Code Stream	Função Desenvolvedor do Code Stream	Função Executor do Code Stream	Função Expectador do Code Stream	Função Usuário do Code Stream
<b>Pipelines</b>						
	Exibir pipelines	Sim	Sim	Sim	Sim	
	Criar pipelines	Sim	Sim			
	Executar pipelines	Sim	Sim	Sim		
	Executar pipelines que incluem variáveis ou endpoints restritos	Sim				
	Atualizar pipelines	Sim	Sim			
	Excluir pipelines	Sim	Sim			
<b>Execução de Pipeline</b>						
	Exibir execuções de pipeline	Sim	Sim	Sim	Sim	
	Retomar, pausar e cancelar execuções de pipeline	Sim	Sim	Sim		
	Retomar pipelines que pararam para aprovação em recursos restritos	Sim				
<b>Integrações Personalizadas</b>						
	Criar integrações personalizadas	Sim	Sim			
	Ler integrações personalizadas	Sim	Sim	Sim	Sim	

Tabela 2-2. Recursos de funções de serviço do Code Stream (continuação)

Contexto da interface do usuário	Recursos	Função Administrador do Code Stream	Função Desenvolvedor do Code Stream	Função Executor do Code Stream	Função Expectador do Code Stream	Função Usuário do Code Stream
	Atualizar integrações personalizadas	Sim	Sim			
<b>Endpoints</b>						
	Exibir execuções	Sim	Sim	Sim	Sim	
	Criar execuções	Sim	Sim			
	Atualizar execuções	Sim	Sim			
	Excluir execuções	Sim	Sim			
<b>Marcar os recursos como restritos</b>						
	Marcar uma variável ou um endpoint como restrito	Sim				
<b>Painéis</b>						
	Exibir painéis	Sim	Sim	Sim	Sim	
	Criar painéis	Sim	Sim			
	Atualizar painéis	Sim	Sim			
	Excluir painéis	Sim	Sim			

## Funções e permissões personalizadas no Code Stream

Você pode criar funções personalizadas no Cloud Assembly que estendem privilégios para usuários que trabalham com pipelines. Ao criar uma função personalizada para pipelines do Code Stream, você seleciona uma ou mais permissões de **Pipeline**.

Selecione o número mínimo de permissões de **Pipeline** necessárias para usuários que receberão essa função personalizada.

Quando um usuário é atribuído a um projeto e recebe uma função nesse projeto, e também recebe uma função personalizada que inclui uma ou mais permissões de **Pipeline**, ele pode executar todas as ações permitidas por essas permissões. Por exemplo, ele pode criar variáveis restritas, gerenciar pipelines restritos, criar e gerenciar integrações personalizadas e muito mais.

Tabela 2-3. Permissões de pipeline que você pode atribuir a funções personalizadas

Permissão de Pipeline	Administrador do Code Stream	Desenvolvedor do Code Stream	Executor do Code Stream	Espectador do Code Stream	Usuário do Code Stream	Administrador do projeto	Membro do projeto	Expectador de projeto
Gerenciar Pipelines	Sim	Sim				Sim	Sim	
Gerenciar Pipelines Restritos	Sim					Sim		
Gerenciar integrações personalizadas	Sim	Sim						
Executar Pipelines	Sim	Sim	Sim			Sim	Sim	
Executar Pipelines Restritos	Sim					Sim		
Gerenciar execuções	Sim					Sim		
Leitura. Essa permissão não está visível.	Sim	Sim	Sim	Sim		Sim	Sim	Sim

Tabela 2-4. Como você pode usar permissões de Pipeline com funções personalizadas

Permissão	O que é possível fazer
Gerenciar Pipelines	<ul style="list-style-type: none"> <li>■ Criar, atualizar, excluir e clonar pipelines.</li> <li>■ Lançar e cancelar o lançamento de pipelines para o VMware Service Broker.</li> <li>■ Criar, atualizar e excluir endpoints.</li> <li>■ Criar, atualizar e excluir variáveis regulares e secretas.</li> <li>■ Criar, clonar, atualizar e excluir um ouvinte Gerrit.</li> <li>■ Conectar e desconectar um ouvinte Gerrit.</li> <li>■ Criar, clonar, atualizar, excluir um gatilho Gerrit.</li> <li>■ Criar, atualizar e excluir um webhook Git.</li> <li>■ Criar, atualizar e excluir um webhook Docker.</li> <li>■ Usar modelos de pipeline inteligentes para criar pipelines.</li> <li>■ Importar pipelines do YAML e exportá-los para o YAML.</li> <li>■ Criar, atualizar e excluir painéis personalizados.</li> <li>■ Pode ler todas as integrações personalizadas.</li> <li>■ Pode ler todos os endpoints e variáveis restritos, mas não pode visualizar seus valores.</li> </ul>
Gerenciar Pipelines Restritos	<ul style="list-style-type: none"> <li>■ Criar, atualizar e excluir endpoints.</li> <li>■ Marcar endpoints como restritos, atualizar endpoints restritos e excluí-los.</li> <li>■ Criar, atualizar e excluir variáveis regulares e secretas.</li> <li>■ Criar, atualizar e excluir variáveis restritas.</li> <li>■ Todas as permissões que você pode fazer com Gerenciar Pipelines.</li> </ul>
Gerenciar integrações personalizadas	<ul style="list-style-type: none"> <li>■ Criar e atualizar integrações personalizadas.</li> <li>■ Controlar versões e lançamentos de integrações personalizadas.</li> <li>■ Excluir e substituir versões de integração personalizadas.</li> <li>■ Excluir Integrações personalizadas.</li> </ul>
Executar Pipelines	<ul style="list-style-type: none"> <li>■ Executar pipelines.</li> <li>■ Pausar, retomar e cancelar execuções de pipeline.</li> <li>■ Repetir a execução de pipelines.</li> <li>■ Reiniciar, repetir a execução e acionar manualmente um evento de gatilho Gerrit.</li> <li>■ Pode aprovar uma operação de usuário e pode fazer aprovações em lote de operações de usuários.</li> </ul>

**Tabela 2-4. Como você pode usar permissões de Pipeline com funções personalizadas (continuação)**

Permissão	O que é possível fazer
Executar Pipelines Restritos	<ul style="list-style-type: none"> <li>■ Executar pipelines.</li> <li>■ Pausar, retomar, cancelar e excluir execuções de pipeline.</li> <li>■ Repetir a execução de pipelines.</li> <li>■ Sincronizar uma execução de pipeline em execução.</li> <li>■ Forçar a exclusão de uma execução de pipeline em execução.</li> <li>■ Reiniciar, repetir a execução, excluir e acionar manualmente um evento de gatilho Gerrit.</li> <li>■ Resolver itens restritos e continuar a execução do pipeline.</li> <li>■ Alternar o contexto de usuário e continuar a execução do pipeline após a aprovação de uma tarefa de Operação do Usuário.</li> <li>■ Todas as permissões que você pode fazer com Executar Pipelines.</li> </ul>
Gerenciar execuções	<ul style="list-style-type: none"> <li>■ Executar pipelines.</li> <li>■ Pausar, retomar, cancelar e excluir execuções de pipeline.</li> <li>■ Repetir a execução de pipelines.</li> <li>■ Reiniciar, repetir a execução, excluir e acionar manualmente um evento de gatilho Gerrit.</li> <li>■ Todas as permissões que você pode fazer com Executar Pipelines.</li> </ul>

Funções personalizadas podem incluir combinações de permissões. Essas permissões são organizadas em grupos de recursos que permitem aos usuários gerenciar ou executar pipelines, com e sem recursos restritos. Essas permissões representam todos os recursos que cada função pode desempenhar no Code Stream.

Por exemplo, se você criar uma função personalizada e incluir a permissão chamada **Gerenciar Pipelines Restritos**, os usuários que tiverem a função de Desenvolvedor do Code Stream poderão:

- Criar, atualizar e excluir endpoints.
- Marcar endpoints como restritos, atualizar endpoints restritos e excluí-los.
- Criar, atualizar e excluir variáveis regulares e secretas.
- Criar, atualizar e excluir variáveis restritas.

**Tabela 2-5. Exemplos de combinações de permissões de Pipeline em funções personalizadas**

Número de permissões atribuídas à função personalizada	Exemplos de permissões combinadas	Como usar essa combinação
Permissão única	Executar Pipelines	
Duas permissões	Gerenciar Pipelines e Executar Pipelines	
Três permissões	Gerenciar Pipelines e Execute Pipelines e Executar Pipelines Restritos	

**Tabela 2-5. Exemplos de combinações de permissões de Pipeline em funções personalizadas (continuação)**

<b>Número de permissões atribuídas à função personalizada</b>	<b>Exemplos de permissões combinadas</b>	<b>Como usar essa combinação</b>
	<b>Gerenciar Pipelines e Gerenciar Integrações Personalizadas e Executar Pipelines Restritos</b>	Essa combinação pode se aplicar a uma função de Desenvolvedor do Code Stream, mas está limitada aos projetos dos quais o usuário é membro.
	<b>Gerenciar Pipelines e Gerenciar Integrações Personalizadas e Gerenciar Execuções</b>	Essa combinação pode se aplicar a um Administrador do Code Stream, mas está limitado aos projetos dos quais o usuário é membro.
	<b>Gerenciar Pipelines, Gerenciar Pipelines Restritos e Gerenciar Integrações Personalizadas</b>	Com essa combinação, um usuário tem permissões totais e pode criar e excluir qualquer coisa no Code Stream.

## Com a função de administrador

Como administrador, você pode criar integrações, endpoints, variáveis, gatilhos, pipelines e dashboards personalizados.

Projetos permitem que pipelines acessem recursos de infraestrutura. Os administradores criam projetos para que os usuários possam agrupar pipelines, endpoints e painéis juntos. Em seguida, os usuários selecionam o projeto em seus pipelines. Cada projeto inclui um administrador e usuários com funções atribuídas.

Com a função de Administrador, é possível marcar endpoints e variáveis como recursos restritos e executar pipelines que usam recursos restritos. Se um usuário não administrativo executar o pipeline que inclui um endpoint ou variável restrita, o pipeline será interrompido na tarefa em que a variável restrita é usada, e um administrador deve retomar esse pipeline.

Como administrador, você também pode solicitar que os pipelines sejam publicados no vRealize Automation Service Broker.

## Com a função de desenvolvedor

É possível trabalhar com pipelines, como administrador, exceto trabalhar com variáveis ou endpoints restritos.

Se você executar um pipeline que usa endpoints ou variáveis restritas, o pipeline apenas será executado até a tarefa que usa o recurso restrito. Em seguida, ele será interrompido, e um administrador do Code Stream ou administrador de projeto deverá retomar o pipeline.

## Se você tiver a função Usuário

Você pode acessar o Code Stream, mas não tem nenhum dos privilégios que as outras funções fornecem.

## Se você tiver a função Expectador

Você pode ver os mesmos recursos visíveis para um administrador, como pipelines, endpoints, execuções de pipeline, dashboards, integrações personalizadas e gatilhos, mas não pode criá-los, atualizá-los ou excluí-los. Para realizar ações, a função de Espectador também deve receber a função de administrador de projeto ou membro do projeto.

Os usuários que têm a função de Espectador podem ver projetos. Eles também podem ver endpoints restritos e variáveis restritas, mas não podem ver as informações detalhadas sobre eles.

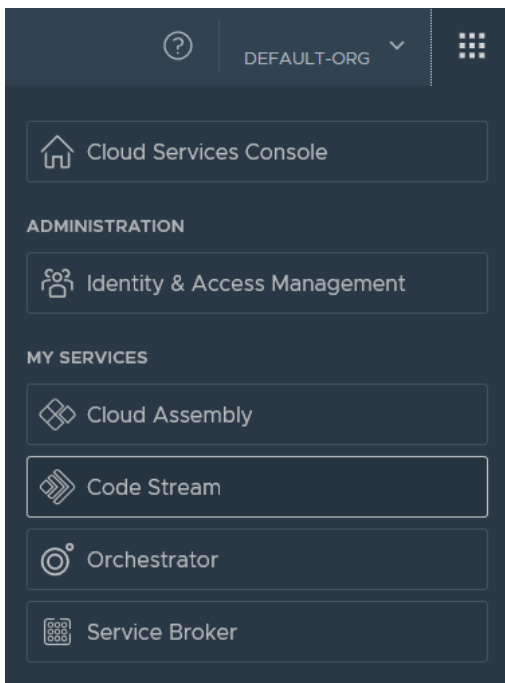
## Se você tiver a função Executor

Você poderá executar pipelines e realizar ações em tarefas de operações do usuário. Também poderá retomar, pausar e cancelar execuções de pipeline. No entanto, não poderá modificar pipelines.

## Como atribuir e atualizar funções

Para atribuir e atualizar funções de outros usuários, é necessário ser um administrador.

- 1 Para ver os usuários ativos e suas funções, no vRealize Automation, clique nos nove pontos no canto superior direito.
- 2 Clique em **Gerenciamento de Identidades e Acessos**.



- 3 Para exibir nomes de usuário e funções, clique em **Usuários Ativos**.





- 4 Para adicionar funções para um usuário ou alterar suas funções, clique na caixa de seleção ao lado do nome do usuário e depois clique em **Editar Funções**.
- 5 Ao adicionar ou alterar funções de usuário, você também pode adicionar acesso a serviços.
- 6 Para salvar as alterações, clique em **Salvar**.

## O que são operações do usuário e aprovações no Code Stream

A área Operações do Usuário exibe as execuções de pipeline que precisam de aprovação. O aprovador necessário pode aprovar ou rejeitar a execução do pipeline.

Ao criar um pipeline, talvez seja necessário adicionar uma aprovação a um pipeline se:

- Um membro da equipe precisar revisar seu código.
- Outro usuário precisar confirmar um artefato de compilação.
- Você deve garantir que todos os testes estejam concluídos.
- Uma tarefa usa um recurso que um administrador marcou como restrito, e essa tarefa requer aprovação.
- O pipeline liberará o software para produção.

Para determinar se uma tarefa de pipeline deve ser aprovada, o aprovador necessário deve ter permissão e experiência.

Ao adicionar uma tarefa de Operação do Usuário, você pode definir o tempo limite de expiração em dias, horas ou minutos. Por exemplo, talvez seja necessário que o usuário responsável aprove o pipeline em 30 minutos. Se ele não o aprovar o pipeline em 30 minutos, este falhará conforme esperado.

Se você habilitar o envio de notificações por e-mail, a tarefa Operação do Usuário enviará notificações apenas para aprovadores que possuem endereços de e-mail completos e não para nomes de aprovadores que não estejam em um formato de e-mail.

Depois que o usuário necessário aprovar a tarefa:

- A execução do pipeline pendente poderá continuar.
- Quando o pipeline continuar, todas as solicitações pendentes anteriores para aprovação dessa mesma tarefa de operação do usuário serão canceladas.

**User Operations** GUIDED SETUP

Active Items   Inactive Items

✓ APPROVE   ✗ REJECT

<input type="checkbox"/>	Index#	Execution	Summary	Requested By	Request Date	Approvers
<input type="checkbox"/>	c07b12	Demo2-Jenkins-K8s#7	Testing	fritz	Nov 13, 2019, 11:32:31 AM	f...om
<input type="checkbox"/>	a0a990	Demo2-Jenkins-K8s#6	Testing	fritz	Nov 11, 2019, 1:34:11 PM	k...om, f...m
<input checked="" type="checkbox"/>	<b>User Operation #8f1728</b> <div> <div>Request Details</div> <div> <div>Execution</div> <div>Demo-Jenkins-K8s #5</div> </div> <div> <div>Summary</div> <div>Testing</div> </div> <div> <div>Approvers</div> <div>k...om, f...om</div> </div> <div> <div>Requested By</div> <div>fritz</div> </div> <div> <div>Requested On</div> <div>Nov 11, 2019, 1:22:21 PM</div> </div> <div> <div>Expires On</div> <div>Nov 14, 2019, 1:22:21 PM</div> </div> </div>					

1   Items per page 20   1 - 7 of 7 items

Na área Operações do Usuário, os itens a serem aprovados ou rejeitados aparecem como itens ativos ou inativos. Cada item é mapeado para uma tarefa de operação do usuário em um pipeline.

- **Itens Ativos** aguardam o aprovador que deve revisar a tarefa e aprová-la ou rejeitá-la. Se você for um usuário que está na lista de aprovadores, poderá expandir a linha de operação do usuário e clicar em **Aceitar** ou **Rejeitar**.
- **Itens Inativos** foram aprovados ou rejeitados. Se um usuário tiver rejeitado a operação do usuário ou se a aprovação na tarefa tiver atingido o tempo limite, ela não poderá mais ser aprovada.

O Índice# é uma cadeia de caracteres alfanumérica exclusiva de seis caracteres que você pode usar como filtro para procurar uma determinada aprovação.

Aprovações de pipeline também aparecem na área **Execuções**.

- Os pipelines que estão aguardando aprovação indicam seu status como aguardando.
- Outros status são: em fila, concluído e com falha.
- Se o pipeline estiver em estado de espera, o aprovador necessário deverá aprovar a tarefa de pipeline.

# Criando e usando pipelines no Code Stream

## 3

Você pode usar o vRealize Automation Code Stream para modelar o processo de compilação, teste e implantação. Com o vRealize Automation Code Stream, configure a infraestrutura que oferece suporte ao seu ciclo de liberação e crie pipelines que modelam suas atividades de liberação de software. O vRealize Automation Code Stream entrega seu software desde o código de desenvolvimento, passando pela fase de testes, e o implanta nas suas instâncias de produção.

Cada pipeline inclui estágios e tarefas. Estágios representam suas fases de desenvolvimento, e tarefas executam as ações necessárias que entregam seu aplicativo de software ao longo dos estágios.

## O que são pipelines no vRealize Automation Code Stream

Um pipeline é uma integração contínua e um modelo de entrega contínua do seu processo de lançamento de software. Ele libera seu software desde o código-fonte, passando pela fase de testes até a produção. Ele inclui uma sequência de estágios que incluem tarefas que representam as atividades em seu ciclo de lançamento de software. Seu aplicativo de software flui de um estágio para o outro por meio do pipeline.

Adicione endpoints para que as tarefas no pipeline possam se conectar a fontes de dados, repositórios ou sistemas de notificação.

## Como criar pipelines

Você pode criar um pipeline começando com uma tela em branco, usando um modelo de pipeline inteligente ou importando o código YAML.

- Use a tela em branco. Para obter um exemplo, consulte [Planejando uma compilação nativa de CI/CD no Code Stream antes de adicionar tarefas manualmente](#).
- Use um modelo de pipeline inteligente. Para obter um exemplo, consulte [Capítulo 4 Planejamento para compilar, integrar e entregar seu código de forma nativa no Code Stream](#).
- Importe o código YAML. Clique em **Pipelines > Importar**. Na caixa de diálogo **Importar**, selecione o arquivo YAML ou insira o código YAML e clique em **Importar**.

Ao usar a tela em branco para criar um pipeline, você adiciona estágios, tarefas e aprovações. O pipeline automatiza o processo que cria, testa, implanta e libera seu aplicativo. As tarefas em cada estágio executam ações que compilam, testam e liberam seu código em cada estágio.

Tabela 3-1. Exemplo de estágios e usos de pipelines

Estágio de exemplo	Exemplos do que você pode fazer
Desenvolvimento	<p>Em um estágio de desenvolvimento, você pode provisionar uma máquina, recuperar um artefato, adicionar uma tarefa de construção que cria um host do Docker para integração contínua do seu código e muito mais.</p> <p>Por exemplo:</p> <ul style="list-style-type: none"> <li>■ Para planejar e criar uma compilação de integração contínua (CI), que entrega seu código usando o recurso de compilação nativo no vRealize Automation Code Stream, consulte <a href="#">Como planejar uma compilação nativa de integração contínua no Code Stream antes de usar o modelo de pipeline inteligente</a>.</li> </ul>
Teste	<p>Em um estágio de teste, é possível adicionar uma tarefa de Jenkins para testar o aplicativo de software e incluir ferramentas de teste pós-processamento, como JUnit, JaCoCo e muito mais.</p> <p>Por exemplo:</p> <ul style="list-style-type: none"> <li>■ Integre o vRealize Automation Code Stream ao Jenkins e execute um trabalho do Jenkins no seu pipeline, que compila e testa seu código-fonte. Consulte <a href="#">Como integrar o Code Stream ao Jenkins</a>.</li> <li>■ Crie scripts personalizados que estendem a capacidade do vRealize Automation Code Stream de se integrar às suas próprias ferramentas de compilação, teste e implantação. Consulte <a href="#">Como integrar minhas próprias ferramentas de compilação, teste e implantação com o Code Stream</a>.</li> <li>■ Rastreie tendências no pós-processamento para um pipeline de integração contínua (CI). Consulte <a href="#">Como usar painéis personalizados para rastrear indicadores-chave de desempenho para o meu pipeline no Code Stream</a>.</li> </ul>
Produção	<p>Em um estágio de produção, você pode integrar um modelo de nuvem no Cloud Assembly que provisiona sua infraestrutura, implanta seu software em um cluster Kubernetes e muito mais.</p> <p>Por exemplo:</p> <ul style="list-style-type: none"> <li>■ Para ver os estágios de exemplo para desenvolvimento e produção, que podem implantar seu aplicativo de software no seu próprio modelo de implantação Azul-Verde, consulte <a href="#">Como implantar meu aplicativo no Code Stream na implantação Azul-Verde</a>.</li> <li>■ Para integrar um modelo de nuvem ao seu pipeline, consulte <a href="#">Como automatizar a liberação de um aplicativo implantado a partir de um modelo de nuvem YAML no Code Stream</a>. Você também pode adicionar uma tarefa de implantação que executa um script para implantar o aplicativo.</li> <li>■ Para automatizar a implantação dos seus aplicativos de software em um cluster do Kubernetes, consulte <a href="#">Como automatizar a liberação de um aplicativo no Code Stream para um cluster do Kubernetes</a>.</li> <li>■ Para integrar o código ao seu pipeline e implantar a imagem de compilação, consulte <a href="#">Como integrar continuamente o código do meu repositório do GitHub ou GitLab ao pipeline no Code Stream</a>.</li> </ul>

Você pode exportar seu pipeline como um arquivo YAML. Clique em **Pipelines**, clique em um cartão de pipeline e, em seguida, clique em **Ações > Exportar**.

## Aprovando pipelines

Você pode obter uma aprovação de outro membro da equipe em pontos específicos no pipeline.

- Para exigir a aprovação em um pipeline, incluindo uma tarefa de operação do usuário em um pipeline, consulte [Como executar um pipeline e ver os resultados](#). Essa tarefa envia uma

notificação por e-mail ao usuário que deve revisá-la. O revisor deve aprovar ou rejeitar a aprovação antes que o pipeline possa continuar sendo executado. Se a tarefa Operação do Usuário tiver um tempo limite de expiração definido em dias, horas ou minutos, o usuário necessário deverá aprovar o pipeline antes da expiração da tarefa. Caso contrário, o pipeline falhará conforme esperado.

- Em qualquer estágio de um pipeline, se uma tarefa ou estágio falhar, será possível especificar que o vRealize Automation Code Stream crie um tíquete Jira. Consulte [Como criar um tíquete do Jira no Code Stream quando uma tarefa de pipeline falhar](#).

## Como disparar pipelines

Pipelines podem ser acionados quando os desenvolvedores verificam seu código no repositório, ou revisam o código, ou quando ele identifica um artefato de construção novo ou atualizado.

- Para integrar o vRealize Automation Code Stream ao ciclo de vida do Git e disparar um pipeline quando os desenvolvedores atualizarem seu código, use o gatilho do Git. Consulte [Como usar o gatilho Git no Code Stream para executar um pipeline](#).
- Para integrar o vRealize Automation Code Stream ao ciclo de vida de revisão de código do Gerrit e disparar um pipeline em revisões de código, use o gatilho do Gerrit. Consulte [Como usar o gatilho Gerrit no Code Stream para executar um pipeline](#).
- Para disparar um pipeline quando um artefato de compilação do Docker for criado ou atualizado, use o gatilho do Docker. Consulte [Como usar o gatilho do Docker no Code Stream para executar um pipeline de entrega contínua](#).

Para obter mais informações sobre os gatilhos para os quais o vRealize Automation Code Stream oferece suporte, consulte [Capítulo 7 Disparando pipelines no Code Stream](#).

Este capítulo inclui os seguintes tópicos:

- [Como executar um pipeline e ver os resultados](#)
- [Que tipos de tarefas estão disponíveis no Code Stream](#)
- [Como usar associações de variáveis em pipelines do Code Stream](#)
- [Como usar associações de variáveis em uma tarefa de condição para executar ou parar um pipeline no Code Stream](#)
- [Que variáveis e expressões eu posso usar ao associar tarefas de pipeline no Code Stream](#)
- [Como enviar notificações sobre meu pipeline no Code Stream](#)
- [Como criar um tíquete do Jira no Code Stream quando uma tarefa de pipeline falhar](#)
- [Como faço para reverter minha implantação no Code Stream](#)

## Como executar um pipeline e ver os resultados

Você pode executar um pipeline a partir do cartão de pipeline, no modo de edição de pipeline e a partir da execução do pipeline. Você também pode usar os gatilhos disponíveis para que o Code Stream execute um pipeline quando determinados eventos ocorrerem.

Quando todos os estágios e tarefas do pipeline forem válidos, ele estará pronto para ser liberado, executado ou disparado.

Para executar ou disparar seu pipeline usando o Code Stream, você pode ativar e executar o pipeline do cartão de pipeline ou enquanto estiver no pipeline. Em seguida, você pode visualizar a execução do pipeline para confirmar se o pipeline criou, testou e implantou seu código.

Quando uma execução de pipeline estiver em andamento, você poderá excluir a execução se for um administrador ou um usuário não administrador.

- **Administrador:** para excluir a execução de um pipeline quando estiver em execução, clique em **Execuções**. Na execução a ser excluída, clique em **Ações > Excluir**.
- **Usuário não administrador:** para excluir uma execução de pipeline em execução, clique em **Execuções** e em **Alt Shift d**.

Quando uma execução de pipeline estiver em andamento e parecer travada, um administrador poderá atualizá-la na página de Execuções ou na página de Detalhes da execução.

- **Página Execuções:** clique em **Execuções**. Na execução a ser atualizada, clique em **Ações > Sincronizar**.
- **Página Detalhes da execução:** clique em **Execuções**, no link para ver os detalhes da execução e em **Ações > Sincronizar**.

Para executar um pipeline quando ocorrerem eventos específicos, use os gatilhos.

- O gatilho Git poderá executar um pipeline quando os desenvolvedores atualizarem o código.
- O gatilho Gerrit poderá executar um pipeline quando ocorrerem revisões de código.
- O gatilho Docker poderá executar um pipeline quando um artefato for criado em um registro de Docker.
- Os comandos `curl` ou `wget` podem fazer com que o Jenkins execute um pipeline após a conclusão de uma compilação do Jenkins.

Para obter mais informações sobre como usar os gatilhos, consulte [Capítulo 7 Disparando pipelines no Code Stream](#).

O procedimento a seguir mostra como executar um pipeline do cartão de pipeline, visualizar execuções, ver os detalhes da execução e usar as ações. Ele também mostra como liberar um pipeline para que você possa adicioná-lo ao vRealize Automation Service Broker.

### Pré-requisitos

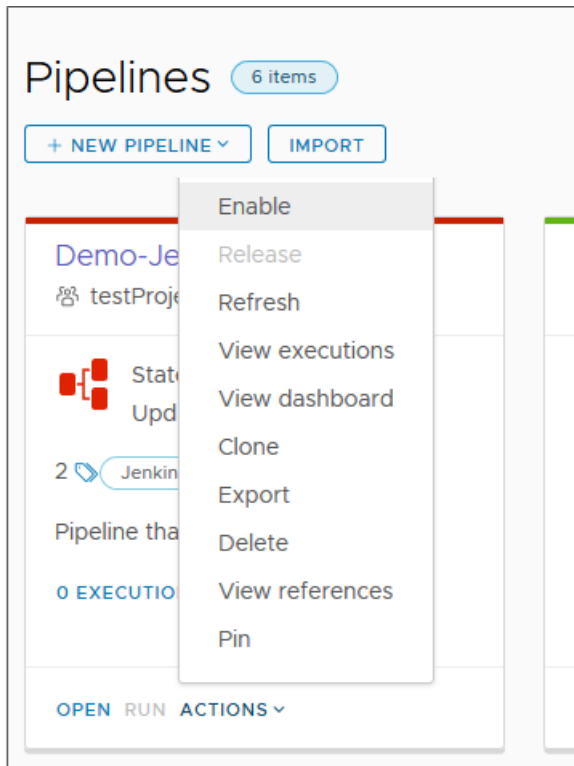
- Verifique se um ou mais pipelines foram criados. Consulte os exemplos em [Capítulo 5 Tutoriais para usar o Code Stream](#).

## Procedimentos

### 1 Habilite seu pipeline.

Para executar ou liberar um pipeline, você deve habilitá-lo primeiro.

- a Clique em **Pipelines**.
- b No seu cartão de pipeline, clique em **Ações > Ativar**.



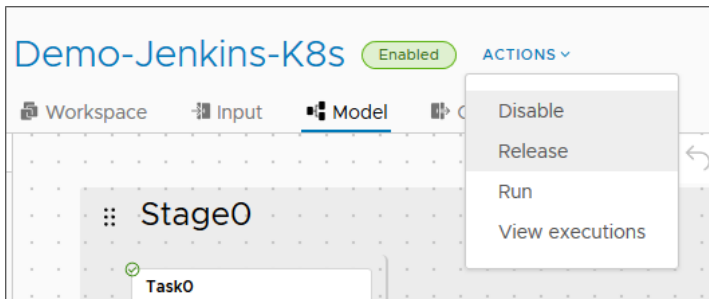
Você também pode habilitar seu pipeline enquanto estiver nele. Se o seu pipeline já estiver habilitado, o comando **Executar** estará ativo, e o menu **Ações** exibirá **Desativar**.

## 2 (Opcional) Liberte seu pipeline.

Se quiser disponibilizar o pipeline como um item de catálogo no vRealize Automation Service Broker, deverá liberá-lo no Code Stream.

- a Clique em **Pipelines**.
- b No seu cartão de pipeline, clique em **Ações > Liberar**.

Você também pode liberar seu pipeline enquanto estiver nele.



Depois de liberar o pipeline, abra o Service Broker para adicionar o pipeline como um item de catálogo e executá-lo. Consulte [Adicionar pipelines do Code Stream ao catálogo do Service Broker](#).

**Observação** Se o pipeline exigir mais de 120 minutos para ser executado, forneça um tempo de execução aproximado como um valor de tempo limite da solicitação. Para definir ou revisar o tempo limite da solicitação para um projeto, abra o Service Broker como administrador e selecione **Infraestrutura > Projetos**. Clique no nome do projeto e depois em **Provisionando**.

Se o valor de tempo limite da solicitação não estiver definido, uma execução que exige mais de 120 minutos para ser executada aparece como falha com um erro de solicitação de tempo limite de retorno de chamada. No entanto, a execução do pipeline não é afetada.

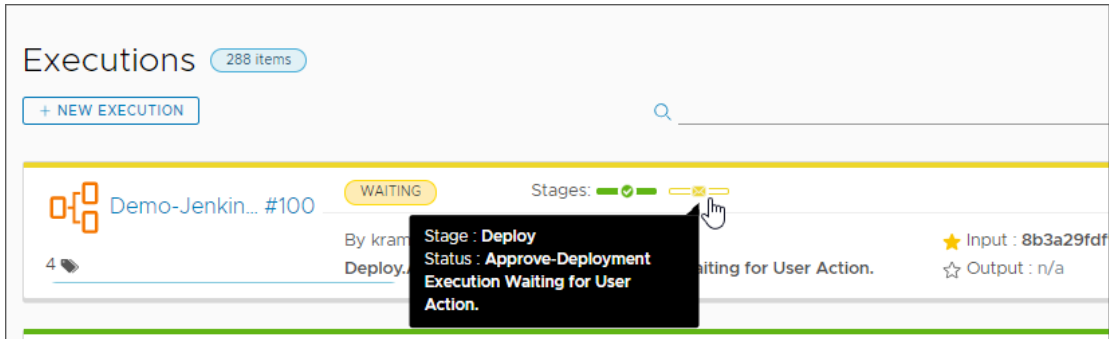
- 3 No cartão de pipeline, clique em **Executar**.
- 4 Para exibir o pipeline enquanto ele é executado, clique em **Execuções**.

O pipeline executa cada estágio em sequência, e a execução do pipeline exibe um ícone de status para cada estágio. Se o pipeline incluir uma tarefa de operação do usuário, um usuário deverá aprovar a tarefa para que o pipeline continue a ser executado. Quando uma tarefa de operação do usuário é usada, o pipeline para de ser executado e aguarda o usuário necessário aprovar a tarefa.

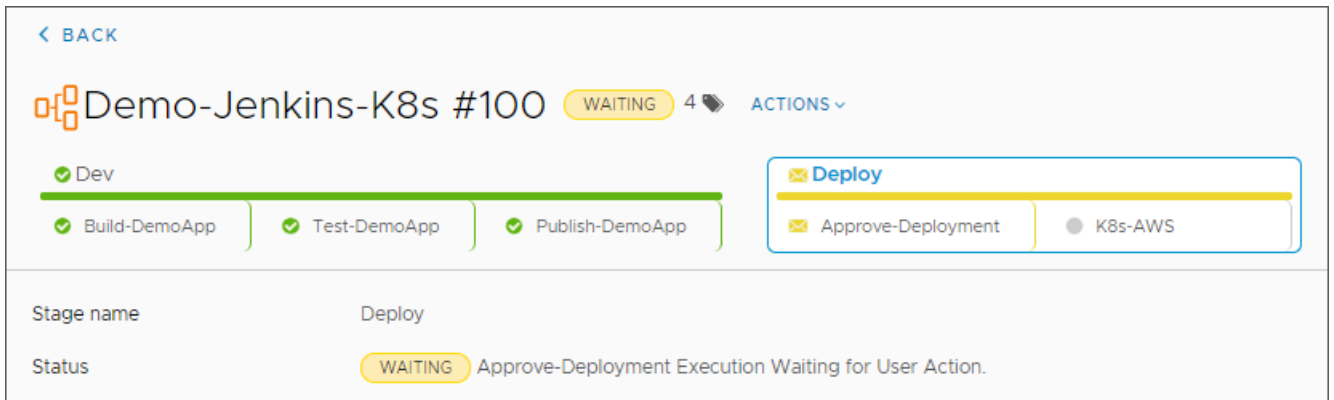
Por exemplo, você pode usar a tarefa de operação do usuário para aprovar a implantação do código em um ambiente de produção.

Se a tarefa Operação do Usuário tiver um tempo limite de expiração definido em dias, horas ou minutos, o usuário necessário deverá aprovar o pipeline antes da expiração da tarefa. Caso contrário, o pipeline falhará conforme esperado.





- 5 Para ver o estágio do pipeline que está aguardando a aprovação do usuário, clique no ícone de status do estágio.

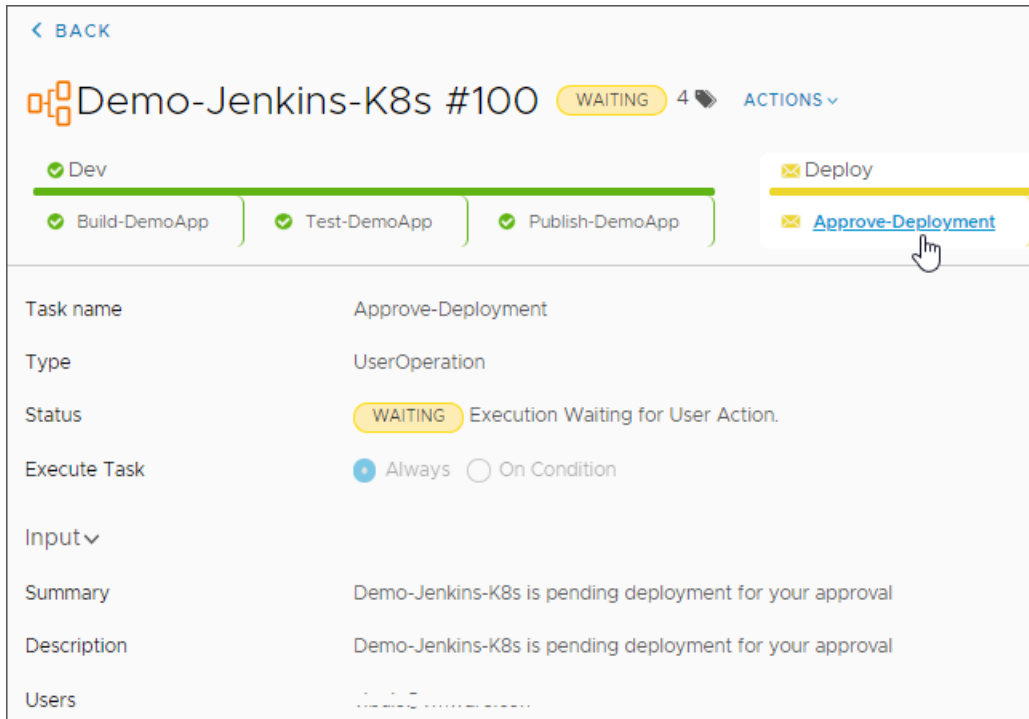


- 6 Para ver os detalhes da tarefa, clique na tarefa.

Após o usuário necessário aprovar a tarefa, um usuário que tiver a função apropriada deverá retomar o pipeline. Para conhecer as funções necessárias, consulte [Como gerenciar o acesso do usuário e as aprovações no Code Stream](#).

Se uma execução falhar, você deverá fazer a triagem e corrigir a causa da falha. Em seguida, vá até a execução e clique em **Ações > Executar novamente**.

Você poderá retomar as execuções de pipelines primários e execuções aninhadas.



- 7 Na execução do pipeline, você pode clicar em **Ações** para visualizar o pipeline e selecionar uma ação, como **Pausar**, **Cancelar** e muito mais. Quando uma execução de pipeline estiver em andamento, se você for um administrador, poderá excluir ou sincronizar a execução do pipeline. Se você for um usuário não administrador, poderá excluir um pipeline em execução.
- 8 Para navegar facilmente entre as execuções e ver os detalhes de uma tarefa, clique em **Execuções** e clique em uma execução de pipeline. Em seguida, clique na guia na parte superior e selecione a execução do pipeline.



## Resultados

Parabéns! Você executou um pipeline, examinou a execução do pipeline e visualizou uma tarefa de operação do usuário que exigiu aprovação para que o pipeline continuasse a ser executado. Você também usou o menu **Ações** na execução do pipeline para retornar ao modelo de pipeline, para poder fazer quaisquer alterações necessárias.

## Próximo passo

Para saber mais sobre como usar o Code Stream para automatizar o ciclo de liberação do software, consulte [Capítulo 5 Tutoriais para usar o Code Stream](#).

## Que tipos de tarefas estão disponíveis no Code Stream

Ao configurar o pipeline, você adiciona tipos específicos de tarefas que o pipeline executa para as ações necessárias. Cada tipo de tarefa integra-se a outro aplicativo e ativa seu pipeline à medida que ele cria, testa e entrega seus aplicativos.

Para executar seu pipeline, se você deve puxar artefatos de um repositório para a implantação, executar um script remoto ou exigir a aprovação de um membro da equipe em uma operação do usuário, o Code Stream tem o tipo de tarefa ideal para você.

O Code Stream permite o cancelamento de uma execução de pipeline em vários tipos de tarefas. Quando você clica em **Cancelar** em uma execução de pipeline, a tarefa, o estágio ou o pipeline inteiro entram no estado de cancelamento e cancela a execução do pipeline.

O Code Stream permite que você cancele a execução do pipeline em uma tarefa, um estágio ou em todo o pipeline ao usar estas tarefas:

- Jenkins
- SSH
- PowerShell
- Operação do Usuário
- Pipeline
- Modelo de nuvem
- vRO
- PESQUISA

O Code Stream não propaga o comportamento de cancelamento a sistemas de terceiros para estas tarefas: CI, Integração Personalizada ou Kubernetes. O Code Stream marca a tarefa como cancelada e interrompe imediatamente a busca do status sem esperar que a tarefa seja concluída. A tarefa pode ser concluída ou falhar no sistema de terceiros, mas será interrompida imediatamente no Code Stream quando você clicar em **Cancelar**.

Antes de usar uma tarefa no seu pipeline, verifique se o endpoint correspondente está disponível.

Tabela 3-2. Obtenha uma aprovação ou defina um ponto de decisão

Tipo de tarefa	O que ele faz	Exemplos e detalhes
<b>Operação do Usuário</b>	Uma tarefa de Operação do Usuário permite uma aprovação necessária que controla quando um pipeline é executado e deve parar para uma aprovação.	Consulte <a href="#">Como executar um pipeline e ver os resultados</a> , e <a href="#">Como gerenciar o acesso do usuário e as aprovações no Code Stream</a> .
<b>Condição</b>	Adiciona um ponto de decisão, que determina se o pipeline deve continuar a ser executado ou para, com base nas expressões de condição. Quando a condição for verdadeira, o pipeline executará tarefas sucessivas. Quando for falso, o pipeline para.	Consulte <a href="#">Como usar associações de variáveis em uma tarefa de condição para executar ou parar um pipeline no Code Stream</a> .

Tabela 3-3. Automatize a integração contínua e a implantação

Tipo de tarefa	O que ele faz	Exemplos e detalhes
Modelo de nuvem	Implanta um modelo de nuvem de automação a partir do GitHub e provisiona um aplicativo, automatiza a integração contínua e entrega contínua (CICD) desse modelo de nuvem para sua implantação.	<p>Consulte <a href="#">Como automatizar a liberação de um aplicativo implantado a partir de um modelo de nuvem YAML no Code Stream</a>.</p> <p>Os parâmetros do modelo de nuvem aparecem depois que você seleciona <b>Criar</b> ou <b>Atualizar</b> pela primeira vez e depois seleciona <b>Modelo de Nuvem</b> e <b>Versão</b>. Você pode adicionar estes elementos, que acomodam associações de variáveis, às áreas de texto de entrada na tarefa de modelo de nuvem:</p> <ul style="list-style-type: none"> <li>■ Inteiro</li> <li>■ Cadeia de caracteres de enumeração</li> <li>■ Boolean</li> <li>■ Variável de matriz</li> </ul> <p>Ao usar associação de variáveis na entrada, esteja ciente dessas exceções. Para enumerações, você deve selecionar um valor de enumeração de um conjunto fixo. Para Valores boolean, você deve inserir o valor na área de texto de entrada.</p> <p>O parâmetro de modelo de nuvem aparece na tarefa de modelo de nuvem quando um modelo de nuvem no Cloud Assembly inclui variáveis de entrada. Por exemplo, se um modelo de nuvem tiver um tipo de entrada de <code>Integer</code>, você poderá inserir o número inteiro diretamente ou como uma variável usando a associação de variáveis.</p>
IC	<p>A tarefa de CI permite a integração contínua do seu código no pipeline, extraindo uma imagem de compilação do Docker de um endpoint de registro e implantando-a em um cluster Kubernetes.</p> <p>A tarefa de CI exibe 100 linhas do log como saída e exibe 500 linhas quando você baixa os logs.</p> <p>As tarefas de CI exigem portas efêmeras de 32768 a 61000.</p>	Consulte <a href="#">Como planejar uma compilação nativa de CICD no Code Stream antes de usar o modelo de pipeline inteligente</a> .
Personalizado	A tarefa Personalizada integra o Code Stream com as suas próprias ferramentas de compilação, teste e implantação.	Consulte <a href="#">Como integrar minhas próprias ferramentas de compilação, teste e implantação com o Code Stream</a> .

Tabela 3-3. Automatize a integração contínua e a implantação (continuação)

Tipo de tarefa	O que ele faz	Exemplos e detalhes
Kubernetes	Automatize a implantação de seus aplicativos de software para clusters do Kubernetes no AWS.	Consulte <a href="#">Como automatizar a liberação de um aplicativo no Code Stream para um cluster do Kubernetes</a> .
Pipeline	Aninha um pipeline em um pipeline principal. Quando um pipeline é aninhado, ele se comporta como uma tarefa no pipeline principal. Na guia Tarefa do pipeline principal, é possível navegar facilmente até o pipeline aninhado clicando no link dele. O pipeline aninhado é aberto em uma nova guia do navegador.	Para encontrar pipelines aninhados em <b>Execuções</b> , digite <b>nested</b> na área de pesquisa.

Tabela 3-4. Integre aplicativos de desenvolvimento, teste e implantação

Tipo de tarefa...	O que ele faz...	Exemplos e detalhes...
Bamboo	Interage com um servidor de integração contínua (CI) Bamboo, que continuamente compila, testa e integra o software na preparação para implantação, além de gerar códigos de disparo quando os desenvolvedores confirmam as alterações. Ele expõe os locais de artefato que a compilação de Bamboo produz para que a tarefa possa produzir os parâmetros de outras tarefas a serem usados para compilação e implantação.	Conecte-se a um endpoint do servidor Bamboo e inicie um plano de compilação de Bamboo a partir de seu pipeline.
Jenkins	Dispara os trabalhos do Jenkins que compilam e testam o código-fonte, executa casos de teste e pode usar scripts personalizados.	Consulte <a href="#">Como integrar o Code Stream ao Jenkins</a> .
TFS	Permite que você conecte seu pipeline ao Team Foundation Server para gerenciar e invocar projetos de compilação, incluindo trabalhos configurados que compilam e testam seu código.	Para versões do Team Foundation Server compatíveis com o Code Stream, consulte <a href="#">O que são endpoints no Code Stream</a> .
vRO	Estende o recurso do Code Stream executando fluxos de trabalho predefinidos ou personalizados no vRealize Orchestrator. O Code Stream oferece suporte para autenticação básica e autenticação baseada em token para o vRealize Orchestrator. O Code Stream usa o token de API para autenticar e validar o cluster do vRealize Orchestrator. Com a autenticação baseada em token, o Code Stream oferece suporte a endpoints do vRealize Orchestrator que usam um Proxy de Extensibilidade de Nuvem. Como resultado, no Code Stream, você pode disparar fluxos de trabalho com um endpoint do vRealize Orchestrator que usa o Proxy de Extensibilidade de Nuvem.	Consulte <a href="#">Como integrar o Code Stream ao vRealize Orchestrator</a> .

Tabela 3-5. Integre outros aplicativos por meio de uma API

Tipo de tarefa...	O que ele faz...	Exemplos e detalhes...
REST	Integra o Code Stream a outros aplicativos que usam uma REST API para que você possa desenvolver e entregar continuamente os aplicativos de software que interagem entre si.	Consulte <a href="#">Como usar uma REST API para integrar o Code Stream a outros aplicativos</a> .
Sondagem	<p>Invoca uma REST API e a sonda até que a tarefa do pipeline atenda aos critérios de saída e seja concluída.</p> <p>Um administrador do Code Stream pode definir a contagem de sondagens como um máximo de 10.000. O intervalo de sondagem deve ser maior que ou igual a 60 segundos.</p> <p>Quando você marca a caixa de seleção <b>Continuar em caso de falha</b>, se a contagem ou o intervalo exceder esses valores, a tarefa de sondagem continuará a ser executada.</p> <p><code>POLL Iteration Count</code>: aparece na execução do pipeline e exibe quantas vezes a tarefa POLL solicitou uma resposta da URL. Por exemplo, se a entrada POLL for 65, e 4 for o número de vezes reais que a solicitação POLL foi executada, a contagem de iterações na saída de execução do pipeline exibirá 4 (de 65).</p>	Consulte <a href="#">Como usar uma REST API para integrar o Code Stream a outros aplicativos</a> .

Tabela 3-6. Execute scripts remotos e definidos pelo usuário

Tipo de tarefa	O que ele faz	Exemplos e detalhes
PowerShell	<p>Com a tarefa do PowerShell, o Code Stream pode executar comandos de script em um host remoto. Por exemplo, um script pode automatizar tarefas de teste e executar tipos administrativos de comandos.</p> <p>O script pode ser remoto ou definido pelo usuário. Ela pode se conectar via HTTP ou HTTPS e pode usar TLS.</p> <p>O host do Windows deve ter o serviço winrm configurado e winrm deve ter MaxShellsPerUser e MaxMemoryPerShellMB configurados.</p> <p>Para executar uma tarefa do PowerShell, é necessário ter uma sessão ativa para o host Remoto do Windows.</p> <p><b>Comprimento da linha de comando do PowerShell</b></p> <p>Se você inserir um comando base64 do PowerShell, deverá calcular o comprimento geral desse comando.</p> <p>O pipeline do Code Stream codifica e agrupa um comando base64 do PowerShell em outro comando, o que aumenta o comprimento geral do comando.</p> <p>O comprimento máximo permitido para um comando winrm do PowerShell é 8192 bytes. O limite de comprimento do comando é menor para a tarefa do PowerShell quando ela é codificada e agrupada. Como resultado, você deve calcular o comprimento do comando antes de inserir o comando do PowerShell.</p> <p>O limite de comprimento do comando para a tarefa Code Stream do PowerShell depende do comprimento codificado em base64 do comando original. O comprimento do comando é calculado da seguinte forma.</p> <pre>3 * (length of original command / 4) - (numberOfPaddingCharacters) + 77 (Length of Write-output command)</pre> <p>O comprimento do comando para Code Stream deve ser menor que o limite máximo de 8192.</p>	<p>Quando você configura MaxShellsPerUser e MaxMemoryPerShellMB:</p> <ul style="list-style-type: none"> <li>■ O valor aceitável para MaxShellsPerUser é 500 para 50 pipelines simultâneos, com 5 tarefas de PowerShell para cada pipeline. Para definir o valor, execute: winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'</li> <li>■ O valor de memória aceitável para MaxMemoryPerShellMB é 2048. Para definir o valor, execute: winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'</li> </ul> <p>O script escreve a saída em um arquivo de resposta que outro pipeline pode consumir.</p>
SSH	<p>A tarefa de SSH permite que a tarefa de script do shell Bash execute comandos de script em um host remoto. Por exemplo, um script pode automatizar tarefas de teste e executar tipos administrativos de comandos.</p> <p>O script pode ser remoto ou definido pelo usuário. Ela pode se conectar por HTTP ou HTTPS e requer uma chave privada ou senha.</p>	<p>O script pode ser remoto ou definido pelo usuário. Por exemplo, um script pode ser semelhante a:</p> <pre>message="Hello World" echo \$message</pre> <p>O script escreve a saída em um arquivo de resposta que outro pipeline pode consumir.</p>



Tabela 3-6. Execute scripts remotos e definidos pelo usuário (continuação)

Tipo de tarefa	O que ele faz	Exemplos e detalhes
	<p>O serviço SSH deve ser configurado no host do Linux, e a configuração de SSHD de <code>MaxSessions</code> deve ser definida para 50.</p> <p>Se você executar várias tarefas de SSH simultaneamente, aumente <code>MaxSessions</code> e <code>MaxOpenSessions</code> no host SSH. Não use sua instância vRealize Automation como o host SSH se precisar modificar as definições de configuração <code>MaxSessions</code> e <code>MaxOpenSessions</code>.</p>	

## Como usar associações de variáveis em pipelines do Code Stream

A associação de uma tarefa de pipeline significa que você cria uma dependência para a tarefa quando esse pipeline é executado. É possível criar uma associação para uma tarefa de pipeline de várias maneiras. Você pode associar uma tarefa a outra, associá-la a uma variável e uma expressão ou associá-la a uma condição.

## Como aplicar associações de cifrão a variáveis de modelo de nuvem em uma tarefa de modelo de nuvem

É possível aplicar associações de cifrão a variáveis de modelo de nuvem em uma tarefa de modelo de nuvem de pipeline do Code Stream. A maneira como você modifica as variáveis no Code Stream depende da codificação das propriedades da variável no modelo de nuvem.

Se você tiver que usar associações com cifrão em uma tarefa de modelo de nuvem, mas a versão atual do modelo de nuvem que você está usando na tarefa de modelo de nuvem não o permitir, modifique o modelo de nuvem no Cloud Assembly e implante uma nova versão. Em seguida, use a nova versão do modelo de nuvem na sua tarefa de modelo de nuvem e adicione as associações de cifrão onde necessário.

Para aplicar associações com cifrão aos tipos de propriedades fornecidos pelo modelo de nuvem do Cloud Assembly, você deve ter as permissões corretas.

- Você deve ter a mesma função que a pessoa que criou a implantação do modelo de nuvem no Cloud Assembly.
- A pessoa que modela o pipeline e a pessoa que executa o pipeline podem ser dois usuários diferentes e pode ter funções diferentes.
- Se um desenvolvedor tiver a função Executor do Code Stream e modelar o pipeline, o desenvolvedor também deverá ter a mesma função do Cloud Assembly que a pessoa que implantou o modelo de nuvem. Por exemplo, a função necessária pode ser administrador do Cloud Assembly.

- Somente a pessoa que modela o pipeline pode criá-lo e criar a implantação, pois ela tem a devida permissão.

Para usar um token de API na tarefa de modelo de nuvem:

- A pessoa que modelar o pipeline poderá fornecer um token de API para outro usuário que tiver a função Executor do Code Stream. Dessa forma, quando o Executor executar o pipeline, ele usará o token de API e as credenciais que são criadas por esse token.
- Quando um usuário inserir o token de API na tarefa de modelo de nuvem, esse token criará as credenciais exigidas pelo pipeline.
- Para criptografar o valor do token de API, clique em **Criar Variável**.
- Se você não criar uma variável para o token de API e usá-lo na tarefa de modelo de nuvem, o valor desse token de API aparecerá em texto sem formatação.

Para aplicar associações de cifra a variáveis de modelo de nuvem em uma tarefa de modelo de nuvem, siga estas etapas.

Comece com um modelo de nuvem que tenha propriedades de variáveis de entrada definidas, como `integerVar`, `stringVar`, `flavorVar`, `BooleanVar`, `objectVar` e `arrayVar`. Você pode encontrar as propriedades da imagem definidas na seção `resources`. As propriedades no código do modelo de nuvem podem ser semelhantes ao seguinte:

```
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}'
```

Você pode usar variáveis de cifrão (\$) para `image` e `flavor`. Por exemplo:

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

Para usar um modelo de nuvem em um pipeline do Code Stream e adicionar associações com cifrão a ele, siga estas etapas.

- 1 No Code Stream, clique em **Pipelines > Tela em Branco**.
- 2 Adicione uma tarefa de **Modelo de nuvem** ao pipeline.
- 3 Na tarefa de modelo de nuvem, para **Origem do modelo de nuvem**, selecione **Modelos de nuvem do Cloud Assembly**, insira o nome do modelo de nuvem e selecione a versão do modelo de nuvem.
- 4 Observe que você pode inserir um token de API, que fornece credenciais para o pipeline. Para criar uma variável que criptografa o token de API na tarefa do modelo de nuvem, clique em **Criar Variável**.
- 5 Na tabela de **Parâmetros e Valores** exibida, observe os valores do parâmetro. O valor padrão para **Tipo** é **Pequeno** e o valor padrão para **Imagem** é **ubuntu**.
- 6 Digamos que você precise alterar o modelo de nuvem no Cloud Assembly. Por exemplo:
  - a Defina `flavor` para que ela use uma propriedade do tipo `array`. O Cloud Assembly permite valores separados por vírgula para **Tipo** quando o tipo é **Matriz**.
  - b Clique em **Implantar**.
  - c Na página **Tipo de Implantação**, insira um nome de implantação e selecione a versão do modelo de nuvem.
  - d Na página **Entradas de Implantação**, você pode definir um ou mais valores para **Tipo**.
  - e Observe que entradas de Implantação incluem todas as variáveis definidas no seu código de modelo de nuvem e aparecem conforme definido no código do modelo de nuvem. Por exemplo: `Integer Var`, `String Var`, `Flavor Var`, `Boolean Var`, `Object Var` e `Array Var`. `String Var` e `Flavor Var` são valores de cadeia de caracteres, enquanto `Boolean Var` é uma caixa de seleção.
  - f Clique em **Implantar**.

- 7 No Code Stream, selecione a nova versão do modelo de nuvem e insira os valores na tabela **Parâmetros e Valores**. Modelos de nuvem oferecem suporte para os seguintes tipos de parâmetros, que permitem associações do Code Stream usando variáveis de cifrão. Existem pequenas diferenças entre a interface do usuário da tarefa de modelo de nuvem do Code Stream e a interface do usuário do modelo de nuvem do Cloud Assembly. Dependendo da codificação de um modelo de nuvem no Cloud Assembly, talvez não seja permitido inserir valores na tarefa de modelo de nuvem no Code Stream.

- a Para **flavorVar**, se o modelo de nuvem tiver definido o tipo como cadeia de caracteres ou matriz, insira uma cadeia de caracteres ou uma matriz de valores separados por vírgula. Uma matriz de exemplo é semelhante a **test, test**.
- b Para **BooleanVar**, no menu suspenso, selecione **true** ou **false**. Ou, para usar uma associação de variável, insira **\$** e selecione uma associação de variável na

Parameter	Value
stringVar	raj
integerVar	1
flavorVar	medium
BooleanVar	\$
objectVar	var
arrayVar	input

Output Parameter: name, description, Stage0, status, deploymentCreate, deployment, deploymentCreate

- c Para **objectVar**, insira o valor entre chaves e aspas neste formato: **{"bkix": "bkix": }**.
  - d O **objectVar** será transmitido ao modelo de nuvem e poderá ser usado de várias maneiras, dependendo do modelo de nuvem. Ele permite um formato de cadeia de caracteres para um objeto JSON e pode adicionar pares de chave/valor como valores separados por vírgula na tabela de chaves/valores. Você pode inserir texto sem formatação para um objeto JSON ou um par de chave/valor como um formato de cadeia de caracteres normal para o JSON.
  - e Para **arrayVar**, insira o valor de entrada separado por vírgula como uma matriz no seguinte formato: **["1", "2"]**.
- 8 No pipeline, é possível associar um parâmetro de entrada a uma matriz.
- a Clique na guia **Entrada**.
  - b Insira um nome para a entrada. Por exemplo, **arrayInput**.
  - c Na tabela **Parâmetros e Valores**, clique em **arrayVar** e insira **\${input.arrayInput}**.

- d Depois de salvar o pipeline e habilitá-lo, quando o pipeline for executado, você deverá fornecer um valor de entrada de matriz. Por exemplo, insira `["1", "2"]` e clique em **Executar**.

Agora, você aprendeu a usar associações de variáveis de cifrão (\$) em um modelo de nuvem em uma tarefa de modelo de nuvem de pipeline do Code Stream.

## Como transmitir um parâmetro a um pipeline quando ele é executado

Você pode adicionar parâmetros de entrada ao pipeline para que o Code Stream os transmita ao pipeline. Em seguida, quando o pipeline for executado, um usuário deverá inserir o valor para o parâmetro de entrada. Quando você adiciona parâmetros de saída ao pipeline, as tarefas de pipeline podem usar o valor de saída de uma tarefa. O Code Stream oferece suporte ao uso de parâmetros de várias maneiras que dão suporte às suas próprias necessidades de pipeline.

Por exemplo, para solicitar que um usuário insira a URL do servidor Git quando um pipeline com uma tarefa REST for executada, é possível associar a tarefa REST a uma URL do servidor Git.

Para criar a associação de variáveis, adicione uma variável de associação de URL à tarefa REST. Quando o pipeline for executado e atingir a tarefa REST, um usuário deverá inserir a URL do servidor Git. Veja como criar a associação:

- 1 No pipeline, clique em a guia **Entrada**.
- 2 Para definir o parâmetro, para **Parâmetros de injeção automática** clique em **Git**.  
A lista de parâmetros Git é exibida e inclui **GIT\_SERVER\_URL**. Se você tiver que usar um valor padrão para a URL do servidor Git, edite esse parâmetro.
- 3 Clique em **Modelo** e clique na tarefa REST.
- 4 Na guia **Tarefa**, na área **URL**, insira \$ e, em seguida, selecione **entrada** e **GIT\_SERVER\_URL**.

The screenshot shows the configuration window for a task named 'Task3'. The interface includes tabs for 'Task :Task3', 'Notifications', and 'Rollback', along with a 'VALIDATE TASK' button. The configuration fields are as follows:

- Task name:** Task3
- Type:** REST
- Continue on failure:** ☐
- Execute task:** ☒ Always ☐ On condition
- REST Request:**
  - Action:** GET
  - URL:** \${input.GIT\_SERVER\_URL} (A dropdown menu is open showing a list of variables: GIT\_BRANCH\_NAME, GIT\_CHANGE\_SUBJECT, GIT\_COMMIT\_ID, GIT\_EVENT\_DESCRIPTION, GIT\_EVENT\_OWNER\_NAME, GIT\_EVENT\_TIMESTAMP, GIT\_REPO\_NAME, and GIT\_SERVER\_URL, which is highlighted.)
  - Agent endpoint:**
  - Headers:**
- Output Parameters:** status, responseHeaders, responseBody, responseJson, responseCode

A entrada é semelhante a: `${input.GIT_SERVER_URL}`

- Para verificar a integridade da associação de variáveis da tarefa, clique em **Validar Tarefa**.

O Code Stream indica que a tarefa foi validada com sucesso.

- Quando o pipeline executar a tarefa REST, um usuário deverá inserir a URL do servidor Git. Caso contrário, a tarefa não terminará de ser executada.

## Como associar duas tarefas de pipeline criando parâmetros de entrada e saída

Ao associar tarefas ao mesmo tempo, adicione uma variável de associação à configuração de entrada da tarefa de recebimento. Em seguida, quando o pipeline for executado, um usuário deverá substituir a variável de associação pela entrada obrigatória.

Para associar tarefas de pipeline, você usa a variável de sinal de cifrão (\$) nos parâmetros de entrada e de saída. Este exemplo mostra como.

Digamos que o pipeline precise chamar uma URL em uma tarefa REST e a saída de uma resposta. Para chamar a URL e gerar a resposta, você inclui parâmetros de entrada e saída na sua tarefa REST. Você também precisa de um usuário que possa aprovar a tarefa e incluir uma tarefa de Operações do Usuário para outro usuário que possa aprová-la quando o pipeline for executado. Esse exemplo mostra como usar expressões nos parâmetros de entrada e saída e fazer com que o pipeline aguarde a aprovação na tarefa.

- 1 No pipeline, clique em a guia **Entrada**.

The screenshot shows the configuration for a task named 'rest-ix-1'. The 'Input' tab is active, displaying 'Input Parameters'. Under 'Auto inject parameters', the 'None' radio button is selected. Below this, there is an 'ADD' button and a button labeled 'ADD/REMOVE INJECTED PARAMETERS'. A table lists the injected parameters:

Starred	Name	Value	Description
<input type="checkbox"/>	URL	{Stage0.Task3.input.http://www.docs.vmware.com}	Docs URL

- 2 Deixe **Parâmetros de injeção automática** como **Nenhum**.
- 3 Clique em **Adicionar** e insira o nome do parâmetro, o valor e a descrição e clique em **OK**. Por exemplo:
  - a Insira um nome de URL.
  - b Insira o valor: {Stage0.Task3.input.http://www.docs.vmware.com}
  - c Insira uma descrição.
- 4 Clique na guia **Saída**, clique em **Adicionar** e insira o nome e o mapeamento de parâmetros de saída.

### Add Pipeline Output Parameter

Name \*

Reference \$ \*

responseHeaders

responseBody

responseJson

responseCode

- a Insira um nome de parâmetro de saída exclusivo.
- b Clique na área **Referência** e insira \$.
- c Insira o mapeamento de saída da tarefa selecionando as opções à medida que elas são exibidas. Selecione **Stage0**, **Task3**, **output** e **responseCode**. Em seguida, clique em **OK**.

rest-ix-1
Enabled
ACTIONS ▾

Workspace
Input
Model
Output

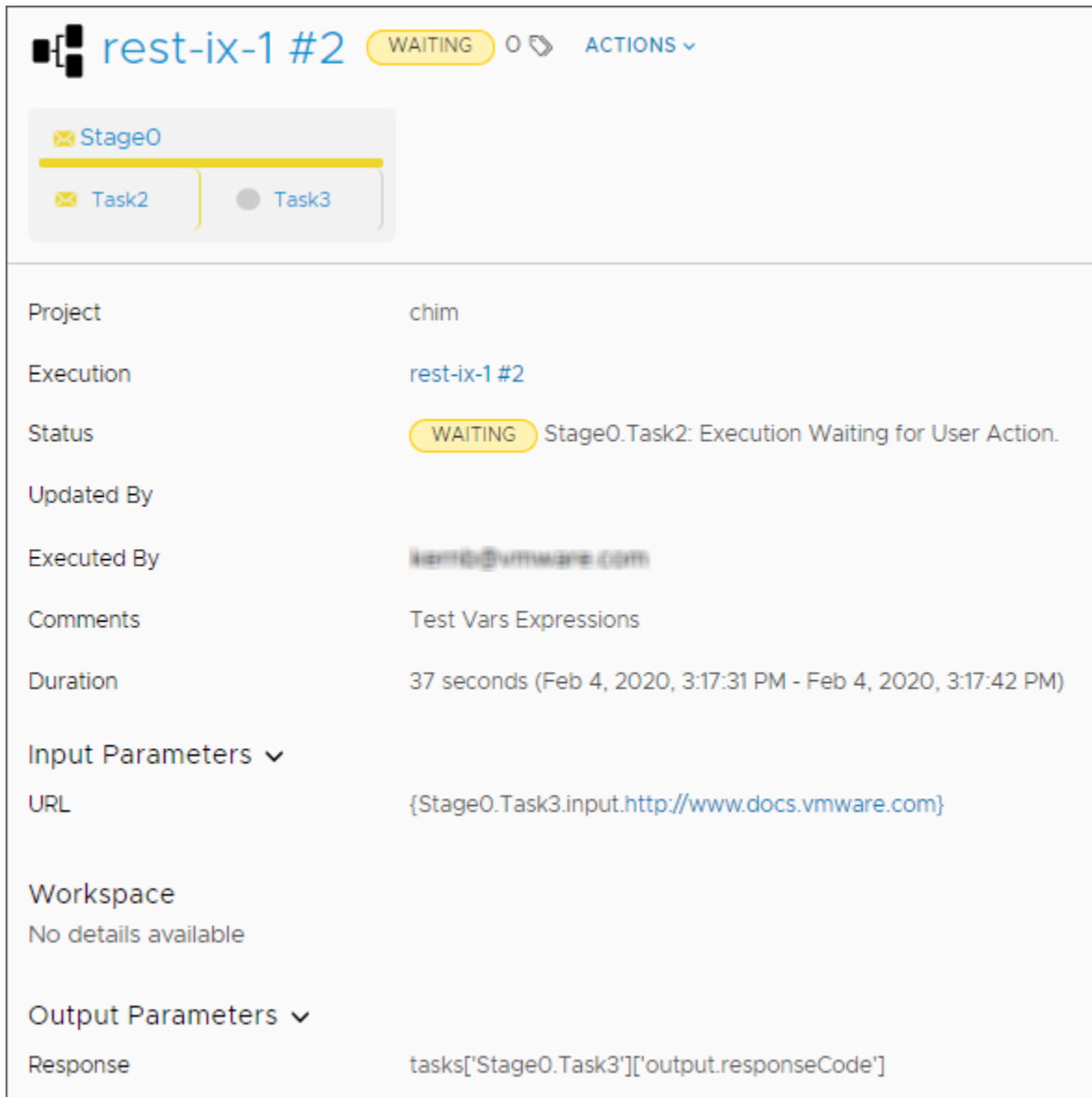
#### Output Parameters ⓘ

ADD

Starred ⓘ	Name ▾	Reference
⋮ ☆	RESTResponse	\${Stage0.Task3.output.responseCode}

- 5 Salve seu pipeline.
- 6 No menu **Ações**, clique em **Executar**.
- 7 Clique em **Ações > Exibir execuções**.
- 8 Clique na execução do pipeline e examine os parâmetros de entrada e de saída que você definiu.



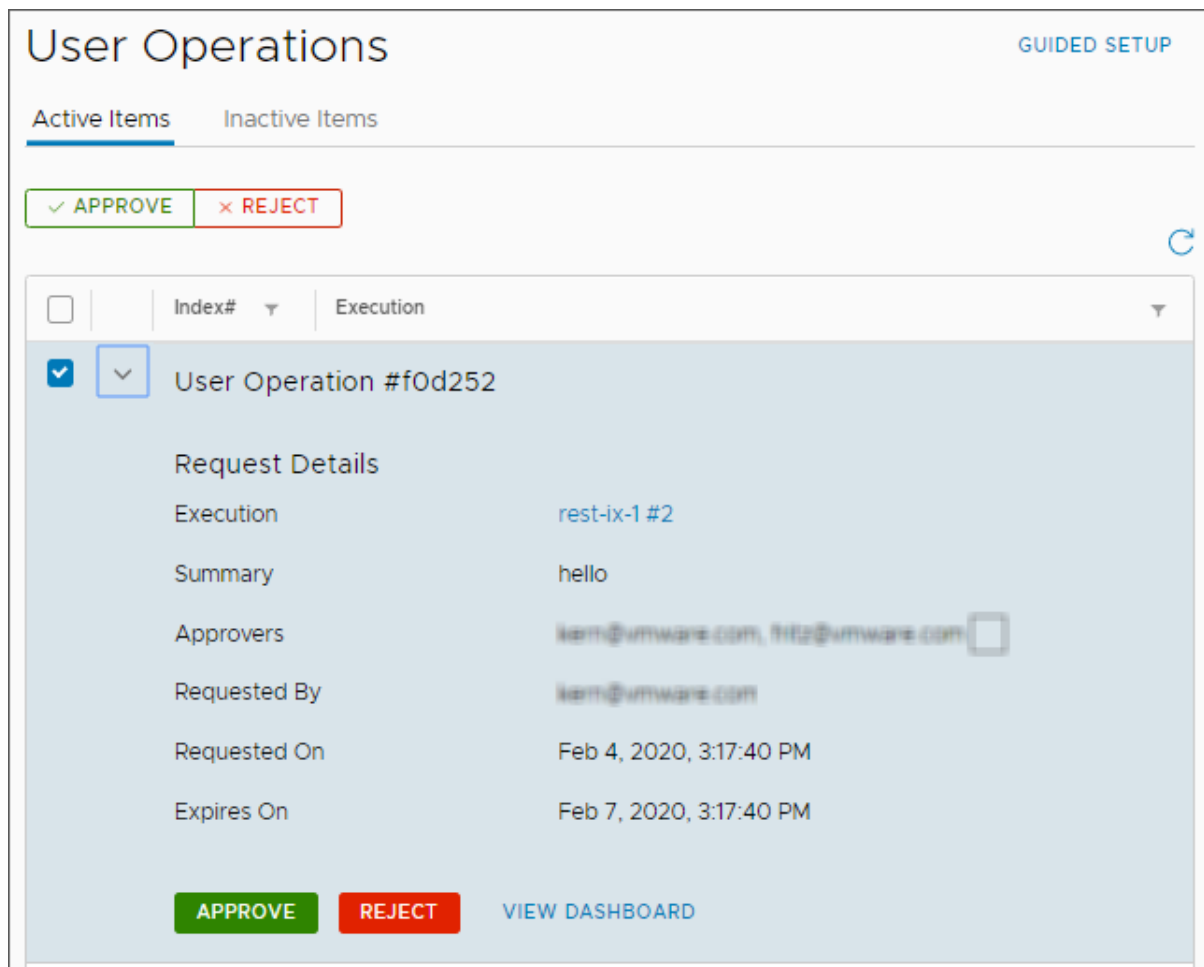

rest-ix-1 #2
WAITING
0
ACTIONS ▾

Stage0

Task2
Task3

Project	chim
Execution	rest-ix-1 #2
Status	<span>WAITING</span> Stage0.Task2: Execution Waiting for User Action.
Updated By	
Executed By	user@vmware.com
Comments	Test Vars Expressions
Duration	37 seconds (Feb 4, 2020, 3:17:31 PM - Feb 4, 2020, 3:17:42 PM)
Input Parameters ▾	
URL	{Stage0.Task3.input.http://www.docs.vmware.com}
Workspace	No details available
Output Parameters ▾	
Response	tasks['Stage0.Task3']['output.responseCode']

- 9 Para aprovar o pipeline, clique em **Operações do Usuário** e exiba a lista de aprovações na guia **Itens Ativos**. Ou permaneça em Execuções, clique na tarefa e clique em **Aprovar**.
- 10 Para habilitar os botões **Aprovar** e **Rejeitar**, clique na caixa de seleção ao lado da execução.
- 11 Para ver os detalhes, expanda a seta suspensa.
- 12 Para aprovar a tarefa, clique em **APROVAR**, insira um motivo e clique em **OK**.



**User Operations** GUIDED SETUP

Active Items Inactive Items

✓ APPROVE ✗ REJECT

Index# Execution

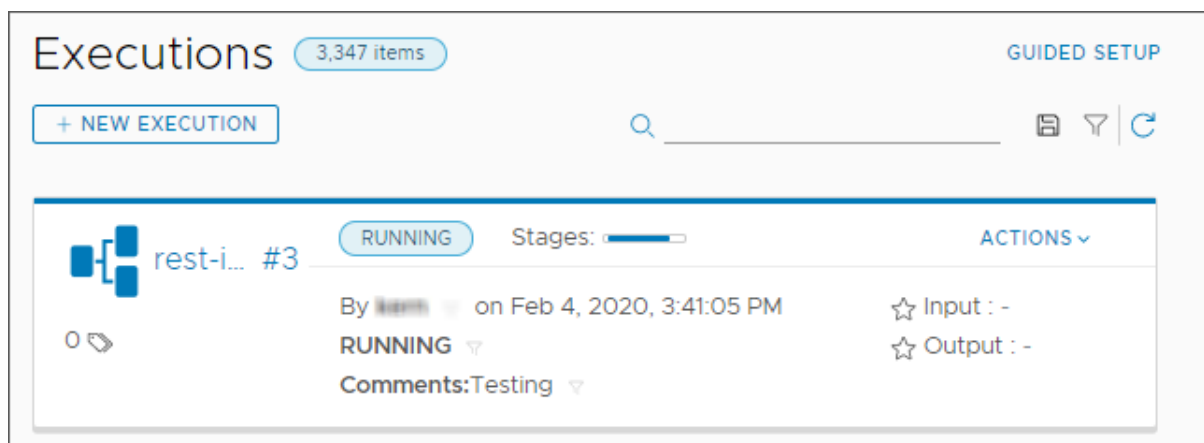
✓ User Operation #f0d252

**Request Details**

Execution	rest-ix-1 #2
Summary	hello
Approvers	kern@vmware.com, fritz@vmware.com
Requested By	kern@vmware.com
Requested On	Feb 4, 2020, 3:17:40 PM
Expires On	Feb 7, 2020, 3:17:40 PM

APPROVE REJECT VIEW DASHBOARD

13 Clique em **Execuções** e observe o pipeline continuar.



**Executions** 3,347 items GUIDED SETUP

+ NEW EXECUTION

rest-i... #3 RUNNING Stages: 0 ACTIONS

By kern on Feb 4, 2020, 3:41:05 PM

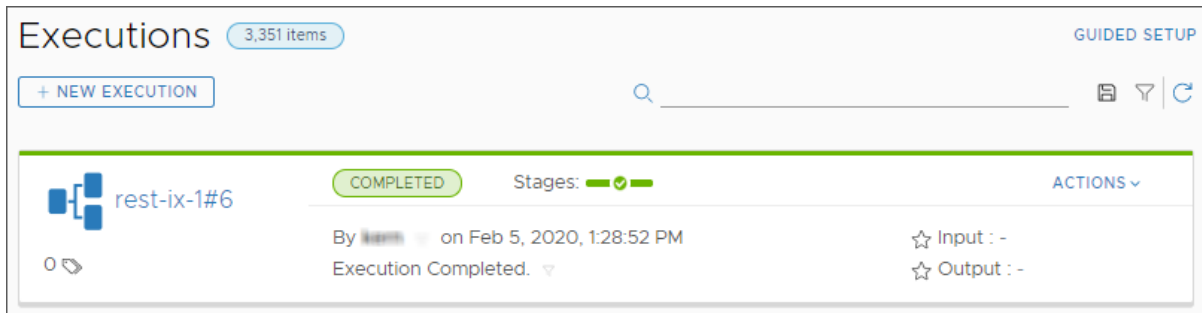
**RUNNING**

Comments: Testing

☆ Input : -

☆ Output : -

14 Se o pipeline falhar, corrija os erros, salve-o e execute-o novamente.



## Como saber mais sobre variáveis e expressões

Para ver detalhes sobre como usar variáveis e expressões ao associar tarefas de pipeline, consulte [Que variáveis e expressões eu posso usar ao associar tarefas de pipeline no Code Stream](#).

Para saber como usar a saída da tarefa de pipeline com uma associação de variável de condição, consulte [Como usar associações de variáveis em uma tarefa de condição para executar ou parar um pipeline no Code Stream](#).

## Como usar associações de variáveis em uma tarefa de condição para executar ou parar um pipeline no Code Stream

É possível fazer com que a saída de uma tarefa no pipeline determine se o pipeline é executado ou interrompido com base em uma condição fornecida. Para aprovar ou reprovar o pipeline com base na saída da tarefa, use a tarefa de Condição.

A tarefa de **Condição** pode ser usada como um ponto de decisão no pipeline. Usando a tarefa Condição com uma expressão de condição fornecida, é possível avaliar quaisquer propriedades em seu pipeline, estágios e tarefas.

O resultado da tarefa Condição determina se a próxima tarefa no pipeline será executada.

- Uma condição verdadeira permite que a execução do pipeline continue.
- Uma condição falsa interrompe o pipeline.

Para obter exemplos de como usar o valor de saída de uma tarefa como a entrada para a próxima tarefa vinculando as tarefas a uma tarefa Condição, consulte [Como usar associações de variáveis em pipelines do Code Stream](#).

**Tabela 3-7. Como a tarefa de Condição e sua expressão de condição se relacionam com o pipeline**

Tarefa de Condição	O que isso afeta	O que ele faz
Tarefa de Condição	Pipeline	A tarefa de <b>Condição</b> determina se o pipeline é executado ou interrompido nesse ponto, dependendo de a saída da tarefa ser verdadeira ou falsa.
Expressão de Condição	Saída da tarefa de Condição	Quando o pipeline é executado, a expressão de condição incluída na tarefa de <b>Condição</b> produz um status de saída verdadeiro ou falso. Por exemplo, uma expressão de condição pode exigir que o status de saída da tarefa de Condição seja <code>Concluído</code> ou pode usar um número de compilação de <code>74</code> .

A expressão de condição aparece na guia Tarefa na tarefa de Condição.

The screenshot shows the configuration for a task named 'Task2' of type 'Condition'. A modal window titled 'Conditional Expression' is open, providing guidance on how to write the condition. It includes an example expression: `${Stage1.task1.output.status} == "COMPLETED" || ${input.buildNumber} == 74`. Below the example, it lists supported constructs and their examples:

Type	Example
Pipeline variables	<code>\$(input.changeSetNumber)</code> (numeric binding) or <code>"\$(input.changeSetOwner)"</code> (string binding)
Task output variables	<code>\$(stage1.task1.output.responseCode)</code> (numeric binding) or <code>"\$(stage1.task1.output.status)"</code> (string binding)
Boolean values	<code>true / false</code>
Numeric values	<code>99</code> or <code>123.45</code> (quotes not allowed)
String values	<code>"Tested"</code> or <code>'Tested'</code>
Relational operators	<code>&lt;</code> , <code>&lt;=</code> , <code>&gt;</code> , <code>&gt;=</code> , <code>==</code> , <code>!=</code>
Arithmetic operators	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>
Boolean	<code>&amp;&amp;</code> (logical and), <code>  </code> (logical or)

A tarefa de **Condição** difere em função e comportamento da configuração **Na Condição** em outros tipos de tarefas.

The screenshot shows the configuration for a task named 'Deploy Phase 1'. The 'Type' is set to 'K8S'. The 'Continue on failure' checkbox is unchecked. The 'Execute task' option is set to 'On condition' (selected with a blue dot). The 'Condition' field is empty, with an information icon (i) to its right. At the top right, there is a 'VALIDATE TASK' button and window controls.

Em outros tipos de tarefas, **Na Condição** determina se a tarefa atual é executada, em vez de tarefas sucessivas, com base na avaliação de sua expressão de condição prévia de verdadeiro ou falso. A expressão de condição da configuração **Na Condição** produz um status de saída verdadeiro ou falso para a tarefa atual quando o pipeline é executado. A configuração **Na Condição** aparece na guia Tarefa com sua própria expressão de condição.

Este exemplo usa a tarefa de Condição.

#### Pré-requisitos

- Verifique a existência de um pipeline e se ele inclui estágios e tarefas.

#### Procedimentos

- 1 No pipeline, determine o ponto de decisão no qual a tarefa de Condição deve aparecer.
- 2 Adicione a tarefa de Condição antes da tarefa que depende do seu status de aprovada ou reprovada.
- 3 Adicione uma expressão de condição à tarefa de Condição.

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

The screenshot shows the vRealize Automation interface. On the left, a workspace view shows a 'Stage0' containing a 'Task1' of type 'Condition'. On the right, the 'Task : Task1' configuration panel is open. The 'Type' is set to 'Condition'. The 'Condition' field contains the expression: `"${Stage1.task1.output.status}" == "COMPLETED" || ${input.buildNumber} == 74`. Below the condition field, there are 'Output Parameters' labeled 'response' and 'status'. A 'VALIDATE TASK' button is at the top right of the configuration panel.

- 4 Valide a tarefa.
- 5 Salve o pipeline, em seguida, ative e execute-o.

## Resultados

Observe as execuções do pipeline e observe se o pipeline continua em execução ou para na tarefa de Condição.

## Próximo passo

Se você reverter uma implantação de pipeline, também poderá usar a tarefa de Condição. Por exemplo, em um pipeline de reversão, a tarefa de Condição ajuda o Code Stream a marcar uma falha de pipeline com base na expressão de condição e pode disparar um único fluxo de reversão para vários tipos de falha.

Para reverter uma implantação, consulte [Como faço para reverter minha implantação no Code Stream](#).

## Que variáveis e expressões eu posso usar ao associar tarefas de pipeline no Code Stream

Com variáveis e expressões, você pode usar parâmetros de entrada e parâmetros de saída com suas tarefas de pipeline. Os parâmetros inseridos vinculam a tarefa de pipeline a uma ou mais variáveis, expressões ou condições e determinam o comportamento do pipeline quando ele é executado.

## Pipelines podem executar soluções de entrega de software simples ou complexas

Ao associar tarefas de pipeline, você pode incluir expressões padrão e complexas. Como resultado, seu pipeline pode executar soluções de entrega de software simples ou complexas.

Para criar os parâmetros no seu pipeline, clique na guia **Entrada** ou **Saída** e adicione uma variável inserindo o símbolo de cifrão \$ e uma expressão. Por exemplo, esse parâmetro é usado como uma entrada de tarefa que chama uma URL: `${Stage0.Task3.input.URL}`.

O formato de associações de variáveis usa componentes de sintaxe chamados de escopos e chaves. O `SCOPE` define o contexto como entrada ou saída, e o `KEY` define os detalhes. No exemplo de parâmetro `${Stage0.Task3.input.URL}`, o `input` é o `SCOPE` e a `URL` é a `KEY`.

As propriedades de saída de qualquer tarefa podem ser resolvidas para qualquer número de níveis aninhados de vinculação de variável.

Para saber mais sobre como usar associações de variáveis em pipelines, consulte [Como usar associações de variáveis em pipelines do Code Stream](#).

## Usando expressões de cifrão com escopos e chaves para associar tarefas de pipeline

Você pode associar tarefas de pipeline usando expressões em variáveis de símbolo de cifrão. Insira expressões como `${SCOPE.KEY.<PATH>}`.

Para determinar o comportamento de uma tarefa de pipeline, em cada expressão, **SCOPE** é o contexto usado pelo Code Stream. O escopo procura um **KEY**, que define os detalhes da ação que a tarefa realiza. Quando o valor para **KEY** é um objeto aninhado, você pode fornecer um **PATH** opcional.

Esses exemplos descrevem **SCOPE** e **KEY** e mostram como você pode usá-los no seu pipeline.

**Tabela 3-8. Usando SCOPE e KEY**

SCOPE	Finalidade da expressão e do exemplo	KEY	Como usar SCOPE e KEY no pipeline
<b>entrada</b>	Propriedades de entrada de um pipeline: <code>\${input.input1}</code>	Nome da propriedade de entrada	<p>Para fazer referência à propriedade de entrada de um pipeline em uma tarefa, use este formato:</p> <pre>tasks:   mytask:     type: REST     input:       url: \$       {input.url}     action: get</pre> <pre>input:   url: https://   www.vmware.com</pre>
<b>saída</b>	Propriedades de saída de um pipeline: <code>\${output.output1}</code>	Nome da propriedade de saída	<p>Para fazer referência a uma propriedade de saída para enviar uma notificação, use este formato:</p> <pre>notifications:   email:     - endpoint:       MyEmailEndpoint       subject:       "Deployment       Successful"       event: COMPLETED       to:       -       user@example.org       body:         Pipeline       deployed       the service       successfully.       Refer \$       {output.serviceURL}</pre>

Tabela 3-8. Usando SCOPE e KEY (continuação)

SCOPE	Finalidade da expressão e do exemplo	KEY	Como usar SCOPE e KEY no pipeline
<b>entrada de tarefa</b>	Entrada para uma tarefa: \$ {MY_STAGE.MY_TASK.input. SOMETHING}	Indica a entrada de uma tarefa em uma notificação	Quando um trabalho do Jenkins é iniciado, ele pode fazer referência ao nome do trabalho disparado na entrada da tarefa. Nesse caso, envie uma notificação usando este formato: <pre> 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.i               nput.job} started               for commit id \$               {input.COMMITID}). </pre>
<b>saída da tarefa</b>	Saída de uma tarefa: \$ {MY_STAGE.MY_TASK.output .SOMETHING}	Indica a saída de uma tarefa em uma tarefa subsequente	Para fazer referência à saída da tarefa 1 do pipeline na tarefa 2, use o seguinte formato: <pre> 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.exa       mple.org/api/       activity       payload: \$       {MY_STAGE.task1.out       put.responseBody} </pre>



Tabela 3-8. Usando SCOPE e KEY (continuação)

SCOPE	Finalidade da expressão e do exemplo	KEY	Como usar SCOPE e KEY no pipeline
<b>var</b>	Variável: <code>\${var.myVariable}</code>	Fazer referência à variável em um endpoint	Para fazer referência a uma variável secreta em um endpoint para uma senha, use este formato: <pre> --- project: MyProject kind: ENDPOINT name: MyJenkinsServer type: jenkins properties:   url: https:// jenkins.example.com username: jenkinsUser password: \$ {var.jenkinsPassword} </pre>
<b>var</b>	Variável: <code>\${var.myVariable}</code>	Fazer referência à variável em um pipeline	Para fazer referência à variável em um URL de pipeline, use este formato: <pre> tasks:   task1:     type: REST     input:       action: get       url: \$ {var.MY_SERVER_URL} </pre>
<b>status da tarefa</b>	Status de uma tarefa: <pre> \$ {MY_STAGE.MY_TASK.status} </pre> <pre> \$ {MY_STAGE.MY_TASK.status Message} </pre>		
<b>status do estágio</b>	Status de um estágio: <pre> \${MY_STAGE.status} </pre> <pre> \$ {MY_STAGE.statusMessage} </pre>		

## Expressões padrão

Você pode usar variáveis com expressões no seu pipeline. Esse resumo inclui as expressões padrão que você pode usar.

Expressão	Descrição
<code>\${comments}</code>	Comentários fornecidos quando na solicitação de execução do pipeline.
<code>\${duration}</code>	Duração da execução do pipeline.
<code>\${endTime}</code>	Hora de término da execução do pipeline em UTC, se concluída.
<code>\${executedOn}</code>	O mesmo que a hora de início, a hora de início da execução do pipeline em UTC.
<code>\${executionId}</code>	ID da execução do pipeline.
<code>\${executionUrl}</code>	URL que navega até a execução do pipeline na interface do usuário.
<code>\${name}</code>	Nome do pipeline.
<code>\${requestBy}</code>	Nome do usuário que solicitou a execução.
<code>\${stageName}</code>	Nome do estágio atual, quando usado no escopo de um estágio.
<code>\${startTime}</code>	Hora de início da execução do pipeline em UTC.
<code>\${status}</code>	Status da execução.
<code>\${statusMessage}</code>	Mensagem de status da execução do pipeline.
<code>\${taskName}</code>	Nome da tarefa atual, quando usada em uma entrada de tarefa ou notificação.

## Usando SCOPE e KEY em tarefas de pipeline

Você pode usar expressões com qualquer uma das tarefas de pipeline com suporte. Esses exemplos mostram como definir o `SCOPE` e `KEY` e confirmar a sintaxe. Os exemplos de códigos usam `MY_STAGE` e `MY_TASK` como nomes de estágios e tarefas de pipeline.

Para saber mais sobre tarefas disponíveis, consulte [Que tipos de tarefas estão disponíveis no Code Stream](#).

Tabela 3-9. Tarefas de isolamento

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
<b>Operação do Usuário</b>			
	Input	<p><code>summary</code>: resumo da solicitação para Operação do Usuário</p> <p><code>description</code>: descrição da solicitação de Operação do Usuário</p> <p><code>approvers</code>: lista de endereços de e-mail de aprovadores, em que cada entrada pode ser uma variável com uma vírgula, ou use ponto-e-vírgula para separar e-mails</p> <p><code>approverGroups</code>: lista de endereços de grupos de aprovadores para a plataforma e a identidade</p> <p><code>sendemail</code>: envia opcionalmente uma notificação por e-mail após a solicitação ou resposta quando definido como "true"</p> <p><code>expirationInDays</code>: número de dias que representa o tempo de expiração da solicitação</p>	<pre> \${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} </pre>
	Output	<p><code>index</code>: string hexadecimal de seis dígitos que representa a solicitação</p> <p><code>respondedBy</code>: nome da conta da pessoa que aprovou/rejeitou a Operação do Usuário</p> <p><code>respondedByEmail</code>: endereço de e-mail da pessoa que respondeu</p> <p><code>comments</code>: comentários fornecidos durante a resposta</p>	<pre> \${MY_STAGE.MY_TASK.output.index} \${MY_STAGE.MY_TASK.output.respondedBy} \$ {MY_STAGE.MY_TASK.output.respondedByEmail} \${MY_STAGE.MY_TASK.output.comments} </pre>
<b>Condição</b>			
	Input	<p><code>condition</code>: condição a ser avaliada. Quando a condição é avaliada como "true", a tarefa é marcada como concluída, enquanto outras respostas reprovam a tarefa</p>	<pre> \${MY_STAGE.MY_TASK.input.condition} </pre>
	Output	<p><code>result</code>: resultado após a avaliação</p>	<pre> \${MY_STAGE.MY_TASK.output.response} </pre>

Tabela 3-10. Tarefas de pipeline

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
Pipeline			
	Input	name: nome do pipeline a ser executado inputProperties: propriedades de entrada a serem transmitidas à execução de pipeline aninhada	<pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.inputProperties} # Referência a todas as propriedades \$ {MY_STAGE.MY_TASK.input.inputProperties.inpu t1} # Referência ao valor de input1 </pre>
	Output	executionStatus: status da execução do pipeline executionIndex: índice da execução do pipeline outputProperties: propriedades de saída de uma execução de pipeline	<pre> \${MY_STAGE.MY_TASK.output.executionStatus} \${MY_STAGE.MY_TASK.output.executionIndex} \${MY_STAGE.MY_TASK.output.outputProperties} # Referência a todas as propriedades \$ {MY_STAGE.MY_TASK.output.outputProperties.ou tput1} # Referência ao valor de output1 </pre>

Tabela 3-11. Automatizar tarefas de integração contínua

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
IC			
	Input	steps: um conjunto de cadeias de caracteres que representam comandos para execução export: variáveis de ambiente a serem preservadas após a execução das etapas artifacts: caminhos de artefatos a serem preservados no caminho compartilhado process: conjunto de elementos de configuração para processamento de JUnit, JaCoCo, Checkstyle, FindBugs	<pre> \${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 } # Referência ao caminho da primeira configuração </pre>
	Output	exports: par de chave/valor, que representa as variáveis de ambiente exportadas da entrada export artifacts: caminho dos artefatos preservados com êxito processResponse: conjunto de resultados processados para a entrada process	<pre> \${MY_STAGE.MY_TASK.output.exports} # Referência a todas as exportações \$ {MY_STAGE.MY_TASK.output.exports.myvar} # Referência ao valor de <b>myvar</b> \${MY_STAGE.MY_TASK.output.artifacts} \$ {MY_STAGE.MY_TASK.output.processRespons e} \$ {MY_STAGE.MY_TASK.output.processRespons e[0].result} # Resultado da primeira configuração do processo </pre>

**Tabela 3-11. Automatizar tarefas de integração contínua (continuação)**

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
<b>Personalizado</b>			
	Input	name: nome da integração personalizada version: uma versão da integração personalizada, liberada ou preterida properties: propriedades a serem enviadas à integração personalizada	<pre>                     \${MY_STAGE.MY_TASK.input.name}                     \${MY_STAGE.MY_TASK.input.version}                     \${MY_STAGE.MY_TASK.input.properties} #                     Referência a todas as propriedades                     \$                     {MY_STAGE.MY_TASK.input.properties.property1} # Referência ao valor de property1                 </pre>
	Output	properties: propriedades de saída da resposta de integração personalizada	<pre>                     \${MY_STAGE.MY_TASK.output.properties} #                     Referência a todas as propriedades                     \$                     {MY_STAGE.MY_TASK.output.properties.property1} # Referência ao valor de property1                 </pre>

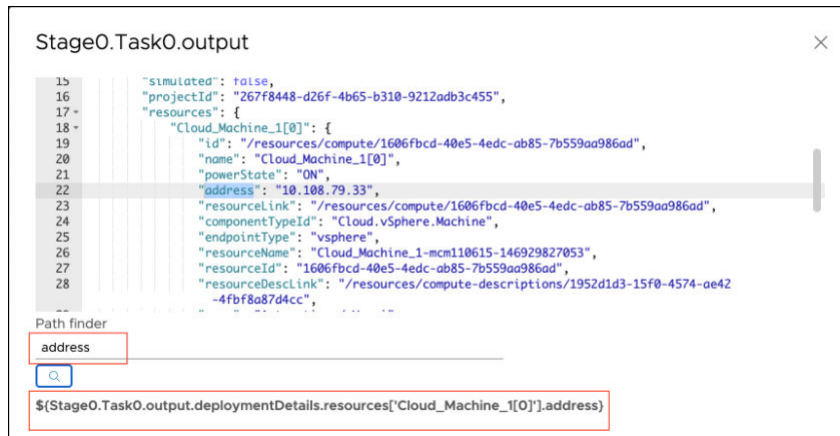
Tabela 3-12. Automatizar tarefas de implantação contínua: Modelo de nuvem

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
Modelo de nuvem			
	Input	<p><code>action</code>: Uma das opções <b>createDeployment</b>, <b>updateDeployment</b>, <b>deleteDeployment</b>, <b>rollbackDeployment</b></p> <p><code>blueprintInputParams</code>: usado para as ações <b>criar implantação</b> e <b>atualizar implantação</b></p> <p><code>allowDestroy</code>: as máquinas podem ser destruídas no processo de implantação da atualização.</p> <p><b>CREATE_DEPLOYMENT</b></p> <ul style="list-style-type: none"> <li>■ <code>blueprintName</code>: Nome do modelo de nuvem</li> <li>■ <code>blueprintVersion</code>: versão do modelo de nuvem</li> </ul> <p>OU</p> <ul style="list-style-type: none"> <li>■ <code>fileUrl</code>: URL do modelo de nuvem remota YAML, após selecionar um servidor GIT.</li> </ul> <p><b>UPDATE_DEPLOYMENT</b></p> <p>Qualquer uma destas combinações:</p> <ul style="list-style-type: none"> <li>■ <code>blueprintName</code>: Nome do modelo de nuvem</li> <li>■ <code>blueprintVersion</code>: versão do modelo de nuvem</li> </ul> <p>OU</p> <ul style="list-style-type: none"> <li>■ <code>fileUrl</code>: URL do modelo de nuvem remota YAML, após selecionar um servidor GIT.</li> </ul> <p>-----</p> <ul style="list-style-type: none"> <li>■ <code>deploymentId</code>: ID da implantação</li> </ul> <p>OU</p> <ul style="list-style-type: none"> <li>■ <code>deploymentName</code>: nome da implantação</li> </ul> <p>-----</p> <p><b>DELETE_DEPLOYMENT</b></p> <ul style="list-style-type: none"> <li>■ <code>deploymentId</code>: ID da implantação</li> </ul>	

Tabela 3-12. Automatizar tarefas de implantação contínua: Modelo de nuvem (continuação)

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
		<p>OU</p> <ul style="list-style-type: none"> <li>■ <code>deploymentName</code>: nome da implantação</li> </ul> <p><b>ROLLBACK_DEPLOYMENT</b></p> <p>Qualquer uma destas combinações:</p> <ul style="list-style-type: none"> <li>■ <code>deploymentId</code>: ID da implantação</li> </ul> <p>OU</p> <ul style="list-style-type: none"> <li>■ <code>deploymentName</code>: nome da implantação</li> </ul> <p>-----</p> <ul style="list-style-type: none"> <li>■ <code>blueprintName</code>: Nome do modelo de nuvem</li> <li>■ <code>rollbackVersion</code>: versão para a qual reverter</li> </ul>	
	Output		<p>Parâmetros que podem se associar a outras tarefas ou à saída de um pipeline:</p> <ul style="list-style-type: none"> <li>■ O nome da implantação pode ser acessado como <code>\${Stage0.Task0.output.deploymentName}</code></li> <li>■ O Id de implantação pode ser acessado como <code>\${Stage0.Task0.output.deploymentId}</code></li> <li>■ Detalhes de Implantação são um objeto complexo, e detalhes internos podem ser acessados usando os resultados do JSON.</li> </ul> <p>Para acessar qualquer propriedade, use o operador de ponto para seguir a hierarquia JSON. Por exemplo, para acessar o endereço do recurso <code>Cloud_Machine_1[0]</code>, a associação <code>\$</code> é:</p> <pre><code>\${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}</code></pre> <p>Da mesma forma, para o tipo, a associação <code>\$</code> é:</p> <pre><code>\${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}</code></pre> <p>Na interface do usuário do Code Stream, você pode obter as associações <code>\$</code> para qualquer propriedade.</p> <ol style="list-style-type: none"> <li>1 Na área de propriedade de saída da tarefa, clique em <b>EXIBIR JSON DE SAÍDA</b>.</li> <li>2 Para encontrar a associação <code>\$</code>, insira qualquer propriedade.</li> <li>3 Clique no ícone de pesquisa, que exibe a associação <code>\$</code> correspondente.</li> </ol>

Exemplo de saída JSON:



Amostra de objeto de detalhes de implantação:

```
{
  "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"
  }
}

```

Tabela 3-13. Automatizar tarefas de implantação contínua: Kubernetes

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
Kubernetes			
	Input	<p><b>action:</b> uma das opções <b>GET, CREATE, APPLY, DELETE, ROLLBACK</b></p> <ul style="list-style-type: none"> <li>■ <b>timeout:</b> tempo limite geral para qualquer ação</li> <li>■ <b>filterByLabel:</b> rótulo adicional para filtragem da ação <b>GET</b> usando K8S labelSelector</li> </ul> <p><b>GET, CREATE, DELETE, APPLY</b></p> <ul style="list-style-type: none"> <li>■ <b>yaml:</b> YAML inline para processar e enviar ao Kubernetes</li> <li>■ <b>parameters:</b> par de CHAVE, VALOR - Substitua <b>\$\$KEY</b> por <b>VALOR</b> na área de entrada do YAML inline</li> <li>■ <b>filePath:</b> caminho relativo do endpoint SCM Git, se fornecido, a partir do qual obter o YAML</li> <li>■ <b>scmConstants:</b> par de CHAVE, VALOR - Substitua <b>\$\$KEY</b> por <b>VALOR</b> no YAML obtido no SCM.</li> <li>■ <b>continueOnConflict:</b> quando definido como verdadeiro, se um recurso já estiver presente, a tarefa continuará.</li> </ul> <p><b>ROLLBACK</b></p> <ul style="list-style-type: none"> <li>■ <b>resourceType:</b> tipo de recurso para reverter</li> <li>■ <b>resourceName:</b> nome do recurso para reverter</li> <li>■ <b>namespace:</b> namespace no qual a reversão deve ser realizada</li> <li>■ <b>revision:</b> revisão para a qual reverter</li> </ul>	<p><code>\${MY_STAGE.MY_TASK.input.action}</code> # Determina a ação a ser executada.</p> <p><code>\${MY_STAGE.MY_TASK.input.timeout}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filterByLabel}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.yaml}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.parameters}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filePath}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.scmConstants}</code></p> <p><code>\$</code> <code>{MY_STAGE.MY_TASK.input.continueOnConflict}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceType}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceName}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.namespace}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.revision}</code></p>
	Output	<p><b>response:</b> captura toda a resposta</p> <p><b>response.&lt;RESOURCE&gt;:</b> o recurso corresponde a configMaps, implantações, endpoints, entradas, trabalhos, namespaces, pods, replicaSets, replicationControllers, segredos, serviços, statefulSets, nós, loadBalancers.</p> <p><b>response.&lt;RESOURCE&gt;.&lt;KEY&gt;:</b> a chave corresponde a um de apiVersion, tipo, metadados, especificação</p>	<p><code>\${MY_STAGE.MY_TASK.output.response}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.response.}</code></p>

Tabela 3-14. Integre aplicativos de desenvolvimento, teste e implantação

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
<b>Bamboo</b>			
	Input	plan: nome do plano planKey: chave do plano variables: variáveis a serem transmitidas ao plano parameters: parâmetros a serem transmitidos ao plano	<code>\${MY_STAGE.MY_TASK.input.plan}</code> <code>\${MY_STAGE.MY_TASK.input.planKey}</code> <code>\${MY_STAGE.MY_TASK.input.variables}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code> # Referência a todos os parâmetros <code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # Referência ao valor de param1
	Output	resultUrl: a URL da compilação resultante buildResultKey: a chave da compilação resultante buildNumber: número da compilação buildTestSummary: resumo da execução dos testes successfulTestCount: resultado do teste aprovado failedTestCount: resultado do teste reprovado skippedTestCount: resultado de teste ignorado artifacts: artefatos da compilação	<code>\${MY_STAGE.MY_TASK.output.resultUrl}</code> <code>\${MY_STAGE.MY_TASK.output.buildResultKey}</code> <code>\${MY_STAGE.MY_TASK.output.buildNumber}</code> <code>\${MY_STAGE.MY_TASK.output.buildTestSummary}</code> # Referência a todos os resultados <code>\${MY_STAGE.MY_TASK.output.successfulTestCount}</code> # Referência à contagem de testes específica <code>\${MY_STAGE.MY_TASK.output.buildNumber}</code>
<b>Jenkins</b>			
	Input	job: nome do trabalho do Jenkins parameters: parâmetros a serem transmitidos ao trabalho	<code>\${MY_STAGE.MY_TASK.input.job}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code> # Referência a todos os parâmetros <code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # Referência ao valor de um parâmetro
	Output	job: nome do trabalho do Jenkins jobId: ID do trabalho resultante, como 1234 jobStatus: status no Jenkins jobResults: coleção de resultados de cobertura de teste/código jobUrl: URL da execução do trabalho resultante	<code>\${MY_STAGE.MY_TASK.output.job}</code> <code>\${MY_STAGE.MY_TASK.output.jobId}</code> <code>\${MY_STAGE.MY_TASK.output.jobStatus}</code> <code>\${MY_STAGE.MY_TASK.output.jobResults}</code> # Referência a todos os resultados <code>\${MY_STAGE.MY_TASK.output.jobResults.junitResponse}</code> # Referência aos resultados do JUnit <code>\${MY_STAGE.MY_TASK.output.jobResults.jacocoResponse}</code> # Referência aos resultados do JaCoCo <code>\${MY_STAGE.MY_TASK.output.jobUrl}</code>
<b>TFS</b>			

Tabela 3-14. Integre aplicativos de desenvolvimento, teste e implantação (continuação)

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
	Input	projectCollection: coleção de projetos do TFS teamProject: projeto selecionado da coleção disponível buildDefinitionId: ID de Definição da Compilação a ser executado	<code>\${MY_STAGE.MY_TASK.input.projectCollection}</code> <code>\${MY_STAGE.MY_TASK.input.teamProject}</code> <code>\${MY_STAGE.MY_TASK.input.buildDefinitionId}</code>
	Output	buildId: ID da compilação resultante buildUrl: URL para visitar o resumo da compilação logUrl: URL para visitar logs dropLocation: local de destino dos artefatos, se houver	<code>\${MY_STAGE.MY_TASK.output.buildId}</code> <code>\${MY_STAGE.MY_TASK.output.buildUrl}</code> <code>\${MY_STAGE.MY_TASK.output.logUrl}</code> <code>\${MY_STAGE.MY_TASK.output.dropLocation}</code>
<b>vRO</b>			
	Input	workflowId: ID do fluxo de trabalho a ser executado parameters: parâmetros a serem transmitidos ao fluxo de trabalho	<code>\${MY_STAGE.MY_TASK.input.workflowId}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code>
	Output	workflowExecutionId: ID da execução do fluxo de trabalho properties: propriedades de saída da execução do fluxo de trabalho	<code>\${MY_STAGE.MY_TASK.output.workflowExecutionId}</code> <code>\${MY_STAGE.MY_TASK.output.properties}</code>

Tabela 3-15. Integre outros aplicativos por meio de uma API

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
<b>REST</b>			
	Input	url: URL para chamar action: método HTTP a ser usado headers: cabeçalhos HTTP a serem aprovados payload: carga útil da solicitação fingerprint: impressão digital a ser correspondida para a uma URL que é https allowAllCerts: quando definido como "true", pode ser qualquer certificado que tenha uma URL https	<code>\${MY_STAGE.MY_TASK.input.url}</code> <code>\${MY_STAGE.MY_TASK.input.action}</code> <code>\${MY_STAGE.MY_TASK.input.headers}</code> <code>\${MY_STAGE.MY_TASK.input.payload}</code> <code>\${MY_STAGE.MY_TASK.input.fingerprint}</code> <code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code>

Tabela 3-15. Integre outros aplicativos por meio de uma API (continuação)

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
	Output	<p>responseCode: código da resposta HTTP</p> <p>responseHeaders: cabeçalhos da resposta HTTP</p> <p>responseBody: formato de cadeia de caracteres da resposta recebida</p> <p>responseJson: resposta de passagem se o tipo de conteúdo for application/json</p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseHeaders}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.output.responseHeaders.header1}</code> # Referência ao cabeçalho de resposta "header1"</p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson}</code> # Referência à resposta como JSON</p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson.a.b.c}</code> # Referência a um objeto aninhado após o caminho JSON a.b.c na resposta</p>
Sondagem			

Tabela 3-15. Integre outros aplicativos por meio de uma API (continuação)

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
	Input	<p>url: URL para chamar</p> <p>headers: cabeçalhos HTTP a serem aprovados</p> <p>exitCriteria: critérios a serem atendidos para que a tarefa seja bem-sucedida ou falhe. Um par de chave/valor de "success" → Expressão "failure" → Expressão</p> <p>pollCount: número de iterações a serem executadas. Um administrador do Code Stream pode definir a contagem de sondagens como um máximo de 10.000.</p> <p>pollIntervalSeconds: número de segundos para aguardar entre cada iteração. O intervalo de sondagem deve ser maior que ou igual a 60 segundos.</p> <p>ignoreFailure: quando definido como "true", ignora as falhas de resposta intermediárias</p> <p>fingerprint: impressão digital a ser correspondida para a uma URL que é https</p> <p>allowAllCerts: quando definido como "true", pode ser qualquer certificado que tenha uma URL https</p>	<p><code>\${MY_STAGE.MY_TASK.input.url}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.headers}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.exitCriteria}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollCount}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollIntervalSeconds}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.ignoreFailure}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.fingerprint}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code></p>
	Output	<p>responseCode: código da resposta HTTP</p> <p>responseBody: formato de cadeia de caracteres da resposta recebida</p> <p>responseJson: resposta de passagem se o tipo de conteúdo for <b>application/json</b></p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson}</code> # Refer to response as JSON</p>

Tabela 3-16. Execute scripts remotos e definidos pelo usuário

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
<b>PowerShell</b> Para executar uma tarefa do PowerShell, você deve: <ul style="list-style-type: none"> <li>■ Ter uma sessão ativa com um host Windows remoto.</li> <li>■ Se você pretende inserir um comando base64 do PowerShell, calcule primeiro o comprimento geral desse comando. Para obter detalhes, consulte <a href="#">Que tipos de tarefas estão disponíveis no Code Stream</a>.</li> </ul>			
	Entrada	host: endereço IP ou nome do host da máquina username: nome de usuário a ser usado para conexão password: senha a ser usada para conexão useTLS: tentar conexão https trustCert: quando definido como "true", confia em certificados autoassinados script: script a ser executado workingDirectory: caminho do diretório para o qual alternar antes de executar o script environmentVariables: um par de chave/valor de variável de ambiente a ser definido arguments: argumentos a serem transmitidos ao script	<pre> \${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} </pre>
	Saída	response: conteúdo do arquivo \$SCRIPT_RESPONSE_FILE responseFilePath: valor de \$SCRIPT_RESPONSE_FILE exitCode: código de saída do processo logFilePath: caminho para o arquivo contendo stdout errorFilePath: caminho para o arquivo contendo stderr	<pre> \${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} </pre>
<b>SSH</b>			

Tabela 3-16. Execute scripts remotos e definidos pelo usuário (continuação)

Tarefa	Escopo	Key	Como usar SCOPE e KEY na tarefa
	Input	host: endereço IP ou nome do host da máquina username: nome de usuário a ser usado para conexão password: senha a ser usada para conexão (opcionalmente, é possível usar privateKey) privateKey: chave privada a ser usada para conexão passphrase: senha opcional para desbloquear privateKey script: script a ser executado workingDirectory: caminho do diretório para o qual alternar antes de executar o script environmentVariables: par de chave/valor da variável de ambiente a ser definida	<pre> \${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} </pre>
	Output	response: conteúdo do arquivo \$SCRIPT_RESPONSE_FILE responseFilePath: valor de \$SCRIPT_RESPONSE_FILE exitCode: código de saída do processo logFilePath: caminho para o arquivo contendo stdout errorFilePath: caminho para o arquivo contendo stderr	<pre> \${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} </pre>

## Como usar uma associação de variáveis entre tarefas

Este exemplo mostra como usar associações de variáveis nas suas tarefas de pipeline.

Tabela 3-17. Exemplos de formatos de sintaxe

Exemplo	Sintaxe
Para usar um valor de saída de tarefa para notificações de pipeline e propriedades de saída de pipeline	<code>\${&lt;Stage Key&gt;.&lt;Task Key&gt;.output.&lt;Task output key&gt;}</code>
Para fazer referência ao valor de saída da tarefa anterior como entrada para a tarefa atual	<code>\${&lt;Previous/Current Stage key&gt;.&lt;Previous task key not in current Task group&gt;.output.&lt;task output key&gt;}</code>

## Para saber mais

Para saber mais sobre variáveis de associação em tarefas, consulte:

- [Como usar associações de variáveis em pipelines do Code Stream](#)
- [Como usar associações de variáveis em uma tarefa de condição para executar ou parar um pipeline no Code Stream](#)
- [Que tipos de tarefas estão disponíveis no Code Stream](#)

## Como enviar notificações sobre meu pipeline no Code Stream

Notificações são maneiras de se comunicar com suas equipes e informá-las sobre o status dos pipelines no Code Stream.

Para enviar notificações quando um pipeline é executado, você pode configurar notificações do Code Stream com base no status de todo o pipeline, estágio ou tarefa.

- Uma notificação por e-mail envia um e-mail em:
  - Conclusão, espera, falha, cancelamento ou início do pipeline.
  - Conclusão, falha ou início do estágio.
  - Conclusão, espera, falha ou início da tarefa.
- Uma notificação de tíquete cria um tíquete e o atribui a um membro da equipe em:
  - Falha ou conclusão do pipeline.
  - Falha do estágio.
  - Falha da tarefa.
- Uma notificação de webhook envia uma solicitação a outro aplicativo em:
  - Falha, conclusão, espera, cancelamento ou início do pipeline.
  - Falha, conclusão ou início do estágio.
  - Falha, conclusão, espera ou início da tarefa.

Por exemplo, você pode configurar uma notificação por e-mail sobre uma tarefa de operação do usuário para obter aprovação em um ponto específico do seu pipeline. Quando o pipeline for executado, essa tarefa enviará um e-mail para a pessoa que deve aprová-la. Se a tarefa Operação do Usuário tiver um tempo limite de expiração definido em dias, horas ou minutos, o usuário necessário deverá aprovar o pipeline antes da expiração da tarefa. Caso contrário, o pipeline falhará conforme esperado.

Para criar um tíquete do Jira quando uma tarefa de pipeline falha, você pode configurar uma notificação. Ou, para enviar uma solicitação a um canal do Slack sobre o status de um pipeline com base no evento de pipeline, você pode configurar uma notificação de webhook.



Você pode usar variáveis em todos os tipos de notificações. Por exemplo, é possível usar `${var}` na URL de uma notificação de Webhook.

### Pré-requisitos

- Verifique se um ou mais pipelines foram criados. Veja os casos de uso em [Capítulo 5 Tutoriais para usar o Code Stream](#).
- Para enviar notificações por e-mail, confirme se é possível acessar um servidor de e-mail ativo. Para obter ajuda, consulte o administrador.
- Para criar tíquetes, como um tíquete do Jira, confirme se o endpoint existe. Consulte [O que são endpoints no Code Stream](#).
- Para enviar uma notificação com base em uma integração, crie uma notificação de webhook. Em seguida, confirme se o webhook foi adicionado e está funcionando. Você pode usar notificações com aplicativos como o Slack, o GitHub ou o GitLab.

### Procedimentos

- 1 Abra um pipeline.
- 2 Para criar uma notificação para o status geral do pipeline, ou o status de um estágio ou tarefa:

Para criar uma notificação para:	O que você faz:
Status do pipeline	Clique em uma área em branco na tela do pipeline.
Status de um estágio	Clique em uma área em branco em um estágio do pipeline.
Status de uma tarefa	Clique em uma tarefa em um estágio do pipeline.

- 3 Clique na guia **Notificações**.
- 4 Clique em **Adicionar**, selecione o tipo de notificação e configure os detalhes da notificação.
- 5 Para criar uma notificação de Slack quando um pipeline for bem-sucedido, crie uma notificação de webhook.
  - a Selecione **Webhook**.
  - b Para configurar a notificação do Slack, insira as informações.
  - c Clique em **Salvar**.
  - d Quando o pipeline é executado, o canal de Slack recebe a notificação sobre o status do pipeline. Por exemplo, os usuários podem ver o seguinte no canal de Slack:

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

- 6 Para criar um tíquete do Jira, configure as informações do tíquete.
  - a Selecione **Tíquete**.
  - b Para configurar a notificação do Jira, insira as informações.
  - c Clique em **Salvar**.

Notification

Send notification type ☐ Email ☒ Ticket ☐ Webhook

When pipeline ☒ Fails ☐ Completes

Jira endpoint

Create Ticket

Jira project

Issue type

Assignee

Summary

Description

## Resultados

Parabéns! Você aprendeu que pode criar vários tipos de notificações em diversas áreas do pipeline no Code Stream.

## Próximo passo

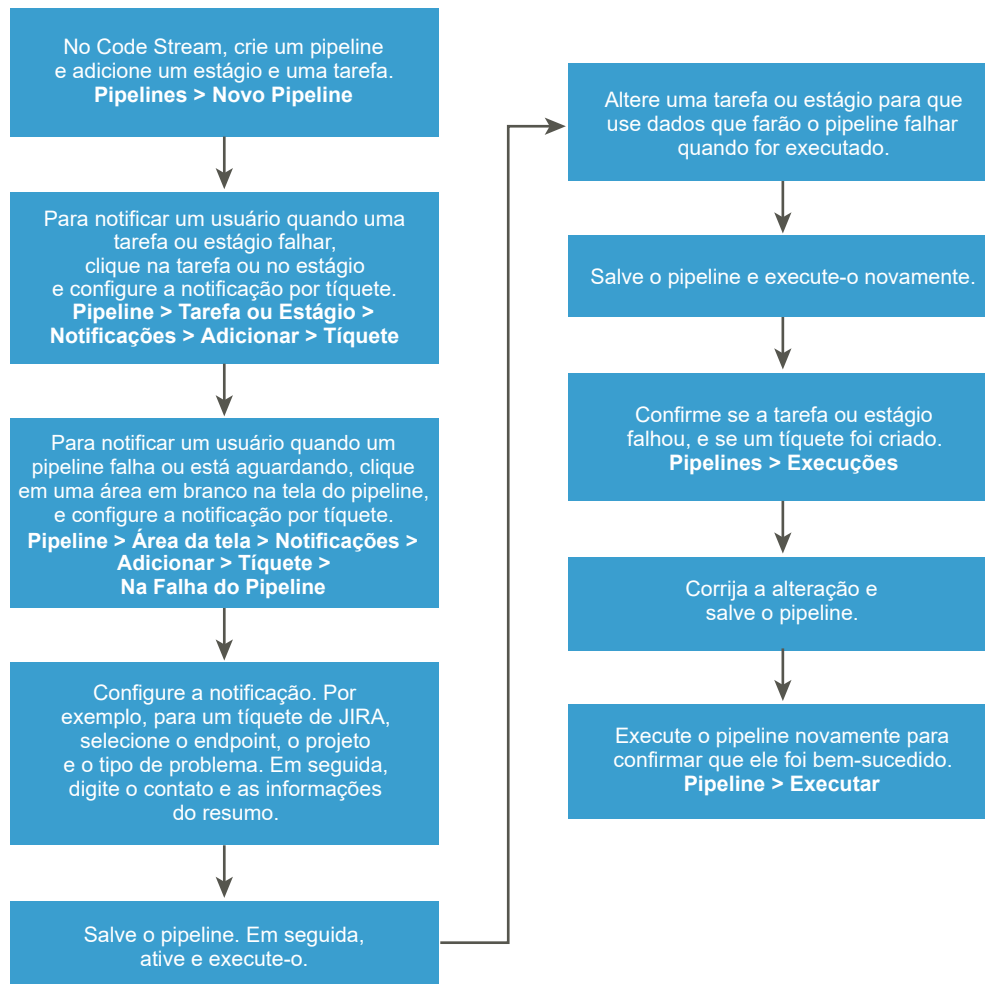
Para obter um exemplo detalhado sobre como criar uma notificação, consulte [Como criar um tíquete do Jira no Code Stream quando uma tarefa de pipeline falhar](#).

# Como criar um tíquete do Jira no Code Stream quando uma tarefa de pipeline falhar

Se um estágio ou uma tarefa de pipeline falhar, será possível fazer com que o Code Stream crie um tíquete do Jira. Você pode atribuir esse tíquete à pessoa que deve resolver o problema. Também é possível criar um tíquete quando o pipeline estiver aguardando ou quando for bem-sucedido.

Você pode adicionar e configurar notificações em uma tarefa, um estágio ou um pipeline. O Code Stream cria o tíquete com base no status da tarefa, do estágio ou do pipeline no qual você adicionou a notificação. Por exemplo, se um endpoint não estiver disponível, será possível fazer com que o Code Stream crie um tíquete do Jira para a tarefa que falha porque ela não pode se conectar ao endpoint.

Também é possível criar notificações quando o pipeline for bem-sucedido. Por exemplo, você pode informar sua equipe de controle de qualidade sobre pipelines bem-sucedidos, para que ela possa confirmar a compilação e executar um pipeline de teste diferente. Ou você pode informar sua equipe de desempenho para que ela seja capaz de medir o desempenho do pipeline e se preparar para uma atualização para validação ou produção.



Este exemplo cria um tíquete do Jira quando uma tarefa de pipeline falha.

#### Pré-requisitos

- Verifique se você tem uma conta válida no Jira e poderá fazer login na instância do Jira.
- Verifique a existência de um endpoint do Jira e se ele está ativo.

## Procedimentos

- 1 No pipeline, clique em uma tarefa.
- 2 Na área de configuração de tarefas, clique em **Notificações**.
- 3 Clique em **Adicionar** e configure as informações do tíquete.
  - a Clique em **Tíquete**.
  - b Selecione o endpoint do Jira.
  - c Insira o projeto Jira e o tipo de problema.
  - d Insira o endereço de e-mail da pessoa que recebe o tíquete.
  - e Digite um resumo e uma descrição do tíquete e clique em **Salvar**.

Notification

Send notification type

☐ Email
 ☒ Ticket
 ☐ Webhook

When task \*

☒ Fails

Jira endpoint \*

TestJira

Create Ticket

Jira project \*

YourProject

Issue type \*

Bug

Assignee \*

username@yourcompany.com

Summary \$ \*

CI task failed

Description \$

Research and correct

CANCEL

SAVE

- 4 Salve o pipeline, em seguida, ative e execute-o.
- 5 Teste o tíquete.
  - a Altere as informações da tarefa para incluir dados que façam a tarefa falhar.
  - b Salve o pipeline e execute-o novamente.
  - c Clique em **Execuções** e confirme que o pipeline falhou.

- d Na execução, confirme que o Code Stream criou o tíquete e o enviou.
- e Altere as informações da tarefa novamente para corrigi-la e execute o pipeline novamente, para verificar se ela é bem-sucedida.

## Resultados

Parabéns! O Code Stream criou um tíquete do Jira quando a tarefa de pipeline falhou e o atribuiu à pessoa responsável pela sua solução.

## Próximo passo

Continue a adicionar notificações para alertar sua equipe sobre os pipelines.

# Como faço para reverter minha implantação no Code Stream

Configure a reversão como um pipeline com tarefas que retornam sua implantação a um estado estável anterior após uma falha em um pipeline de implantação. Para reverter se ocorrer uma falha, você anexa o pipeline de reversão a tarefas ou estágios.

Dependendo da sua função, seus motivos para reversão podem variar.

- Como engenheiro de liberação, quero que o Code Stream verifique o sucesso durante uma liberação para eu saber se devo continuar com a versão ou revertê-la. Possíveis falhas incluem falha na tarefa, uma rejeição em UserOps, excedendo o limite de métricas.
- Como proprietário do ambiente, desejo reimplantar uma versão anterior para poder retornar rapidamente um ambiente para um estado em boas condições.
- Como proprietário do ambiente, quero oferecer suporte à reversão de uma implantação de Azul-Verde para que eu possa minimizar o tempo de inatividade das versões com falha.

Ao usar um modelo de pipeline inteligente para criar um pipeline de CD com a opção reversão selecionada, a reversão é automaticamente adicionada às tarefas no pipeline. Nesse caso de uso, o modelo de pipeline inteligente será usado para definir a reversão de uma implantação de aplicativo em um cluster Kubernetes usando o modelo de implantação de upgrade contínuo. O modelo de pipeline inteligente cria um pipeline de implantação e um ou mais pipelines de reversão.

- No pipeline de implantação, a reversão será necessária se as tarefas de implantação de atualização ou de implantação de verificação falharem.
- No pipeline de reversão, a implantação é atualizada com uma imagem antiga.

Também é possível criar manualmente um pipeline de reversão usando um modelo em branco. Antes de criar um pipeline de reversão, é necessário planejar o fluxo de reversão. Para obter mais informações básicas sobre a reversão, consulte [Planejando uma reversão no Code Stream](#).


## Pré-requisitos


- Verifique se você é membro de um projeto no Code Stream. Se não for, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).
- Configure os clusters do Kubernetes em que o pipeline implantará seu aplicativo. Configure um cluster de desenvolvimento e um cluster de produção.
- Verifique se há uma configuração de registro do Docker.
- Identifique um projeto que agrupará todo o seu trabalho, incluindo o pipeline, os endpoints e os painéis.
- Familiarize-se com o modelo inteligente de CD, conforme descrito na parte sobre CD de [Como planejar uma compilação nativa de CI/CD no Code Stream antes de usar o modelo de pipeline inteligente](#), por exemplo:
  - Crie os endpoints de produção e desenvolvimento do Kubernetes que implantam a imagem do aplicativo nos clusters do Kubernetes.
  - Prepare o arquivo YAML do Kubernetes que cria o Namespace, o Serviço e a Implantação. Se você precisar baixar uma imagem de um repositório de propriedade privada, o arquivo YAML deverá incluir uma seção com o Segredo de configuração do Docker.

## Procedimentos


- 1 Clique em **Pipelines > Novo Pipeline > Modelo Inteligente > Entrega Contínua**.
- 2 Digite as informações no modelo de pipeline inteligente.
  - a Selecione um projeto.
  - b Digite o nome do pipeline, como **RollingUpgrade-Example**.
  - c Selecione os ambientes para seu aplicativo. Para adicionar a reversão à implantação, é necessário selecionar **Prod..**
  - d Clique em **Selecionar**, escolha um arquivo YAML do Kubernetes e clique em **Processar**.  
O modelo de pipeline inteligente exibe os serviços e os ambientes de implantação disponíveis.
  - e Selecione o serviço que o pipeline usará para a implantação.
  - f Selecione os endpoints do cluster para o ambiente Dev e o ambiente Prod.
  - g Para a origem da imagem, selecione **Entrada do tempo de execução do pipeline**.
  - h Para o modelo de implantação, selecione **Atualização Contínua**.
  - i Clique em **Reverter**.
  - j Forneça a **URL de verificação de integridade**.

### Smart Template: Continuous Delivery

Endpoint prerequisites  Kubernetes Docker Registry


Project \*  

Pipeline name \*

Environment  ☒ Development ☒ Production

Kubernetes YAML files \* SELECT PROCESS  
Processed files: cdTemplate.yaml

Select service

Deployment name	Service	Namespace	Image
 codestream-demo	codestream-demo	bgreen	symphony-tango-beta2.jfrog.io/codestream-demo

1 services

Deployment



Environment	Cluster Endpoint	Namespace
Development	Dev-VKE-Cluster 	bgreen-596788
Production	Prod-VKE-Cluster 	bgreen


Image source \* ☐ Docker trigger ☒ Pipeline runtime input

Deployment model \* ☐ Canary ☒ Rolling upgrade ☐ Blue-Green

Rollback ☒

Health check URL \*

CREATE CANCEL



- 3 Para criar o pipeline chamado RollbackUpgrade-Example, clique em **Criar**.

O pipeline denominado RollbackUpgrade-Example é exibido, e o ícone de reversão aparece nas tarefas que podem ser revertidas no estágio Desenvolvimento e no estágio Produção.

RollbackUpgrade-Example Disabled

Workspace

Input

Model

Output

Development

Create Namespace  
Kubernetes

Create Secret  
Kubernetes

Create Serv  
Kubernetes

Production

late

Update Deployment  
Kubernetes

Verify Deployment  
POLL

Task: Create Secret

Notifications

Rollback

VALIDATE TASK

Task name

Create Secret

Type

Kubernetes

Continue on failure

☐

Execute task

☒ Always
 ☐ On condition

Kubernetes Task Properties

Kubernetes cluster

Dev-VKE-Cluster

Timeout (in Mins)

5

Action

☐ Get
 ☒ Create
 ☐ Apply
 ☐ Delete
 ☐ Rollback

Continue on conflict

☒

Payload source

☐ Source control
 ☒ Local definition

Local YAML definition

FILE

```

1 apiVersion: v1
2 data:
3   .dockercfg: eyJ2ZW50bG9ueS10bW5nby11ZXRN1Sqn3vZy5pb16eyJ1C
4     2ybybFZ5IjIiInRhbmdvLW01dGdy1WzGdFZc3dvcmQ1O1JhR0N0cm01L
5     1UQ111e1c1C31bWpbc16inRhbmdvLW01dGdyQHZtd2FyZS5jb281LCk1
6     hXRo1joiZEgdyOyOHRZbVYwVRJ11Vun3WpEpsVGkxdFZF5XRT5G8Z
7     In19
8   kind: Secret
9   metadata:
10    name: jfrog-beta2
11    namespace: bgreen-549930
12    type: kubernetes.io/dockercfg

```

Parameters

ADD

DELETE

Key

Value

Output Parameters

status

response

EDIT

RUN

CLOSE

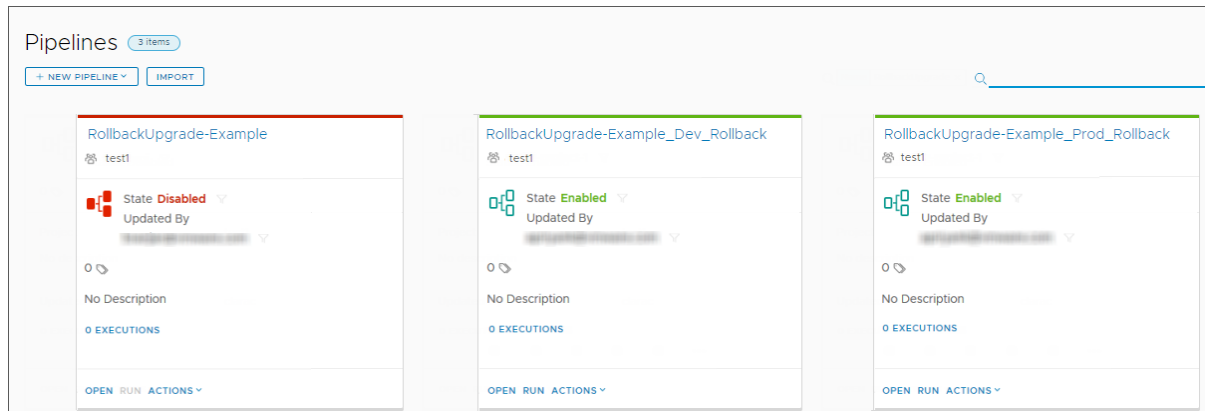
Last saved 9 minutes ago

#### 4 Feche o pipeline.

Na página Pipelines, o pipeline que você criou é exibido, e um novo pipeline para cada estágio no seu pipeline é exibido.

- RollingUpgrade-Example. O Code Stream desativa o pipeline que você criou por padrão, o que garante que você possa revisá-lo antes de o executar.
- RollingUpgrade-Example\_Dev\_Rollback. Falha de tarefas na fase de desenvolvimento, como **Criar serviço**, **Criar segredo**, **Criar implantação** e **Verificar implantação** invocam esse pipeline de desenvolvimento de reversão. Para garantir a reversão de tarefas de desenvolvimento, o Code Stream ativa o pipeline de desenvolvimento de reversão por padrão.
- RollingUpgrade-Example\_Prod\_Rollback. Falha de tarefas no estágio de produção, como **Implantar - Fase 1**, **Verificar - Fase 1**, **Implantar - Fase de lançamento**, **Finalizar - Fase de lançamento** e **Verificar - Fase de lançamento**, invocam esse pipeline de produção de reversão. Para garantir a reversão das tarefas de produção, o Code Stream ativa o pipeline de produção de reversão por padrão.





5 Ative e execute o pipeline criado.

Quando você iniciar a execução, o Code Stream solicitará parâmetros de entrada. Forneça a imagem e a tag para o endpoint no repositório Docker que está sendo usado.

6 Na página Execuções, selecione **Ações > Visualizar Execução** e observe a execução do pipeline.

O pipeline começa a **EXECUÇÃO** e passa pelas tarefas de estágio de desenvolvimento. Se o pipeline não conseguir executar uma tarefa durante o estágio de desenvolvimento, o pipeline chamado RollingUpgrade-Example\_Dev\_Rollback será disparado para reverter a implantação e o status do pipeline será alterado para **ROLLING\_BACK**.

[< BACK](#)

**RollbackUpgrade-Example #1**
ROLLING\_BACK
0
ACTIONS ▾

Development

✓ Create Namespace
✓ Create Secret
✓ Create Service
● Create Deployment
⌂
● Verify Deplo

---

Project

test1

Execution

RollbackUpgrade-Example #1

Status

ROLLING\_BACK
RUNNING

Updated by

Executed by

Duration

12m 9s 186ms (01/11/2019 1:24 PM - )

Input Parameters ▾

image

demo-image-cs

tag

latest

Workspace

Details not available

Output Parameters ▾

The Execution did not output any properties

Após a reversão, a página Execuções listará duas execuções do pipeline RollingUpgrade-Example.

- O pipeline que você criou é revertido e exibe **ROLLBACK\_COMPLETED**.
- O pipeline de desenvolvimento de reversão que acionou e executou a reversão exibe **COMPLETED**.

**Executions**
604 items

[+ NEW EXECUTION](#)
🔍

---

**RollbackUpgrade-Example\_Dev... #1**
COMPLETED
Stages: 2/2

1
Rollback for RollbackUpgrade-Example#1

By started on 01/11/2019 1:36 PM

Execution Completed.

Comments: Triggered to rollback Development. Create Deployment of RollbackUpgrade-Example#1

---

**RollbackUpgrade-Example#1**
ROLLBACK\_COMPLETED
Stages: 1/1

0

By started on 01/11/2019 1:24 PM

Create Deployment ROLLBACK\_COMPLETED

## Resultados

Parabéns! Você definiu com sucesso um pipeline com reversão e observou o Code Stream reverter o pipeline no ponto de falha.

# Planejamento para compilar, integrar e entregar seu código de forma nativa no Code Stream

## 4

Antes que o Code Stream compile, integre e entregue seu código usando o recurso nativo que cria um pipeline de CICD, CI ou CD para você, planeje sua compilação nativa. Em seguida, você poderá criar seu pipeline usando um dos modelos de pipeline inteligentes ou adicionando manualmente os estágios e as tarefas.

Para planejar sua criação de integração contínua e entrega contínua, incluímos vários exemplos que mostram como. Esses planos descrevem os pré-requisitos e visões gerais que podem ajudar você a preparar e usar o recurso de criação nativa de maneira eficaz ao criar seus pipelines.

Este capítulo inclui os seguintes tópicos:

- Como configurar o espaço de trabalho do pipeline
- Como planejar uma compilação nativa de CICD no Code Stream antes de usar o modelo de pipeline inteligente
- Como planejar uma compilação nativa de integração contínua no Code Stream antes de usar o modelo de pipeline inteligente
- Como planejar uma compilação nativa de entrega contínua no Code Stream antes de usar o modelo de pipeline inteligente
- Planejando uma compilação nativa de CICD no Code Stream antes de adicionar tarefas manualmente
- Planejando uma reversão no Code Stream

## Como configurar o espaço de trabalho do pipeline

Para executar tarefas de integração contínua e tarefas personalizadas, você deve configurar um espaço de trabalho para seu pipeline do Code Stream.

No espaço de trabalho do pipeline, selecione **Tipo** como Docker ou Kubernetes e forneça o respectivo endpoint. As plataformas Docker e Kubernetes gerenciam todo o ciclo de vida do contêiner que o Code Stream implanta para execução da tarefa de integração contínua (continuous integration, CI) ou personalizada.

- O espaço de trabalho do Docker requer o endpoint do host do Docker, a URL da imagem do construtor, o registro da imagem, o diretório de trabalho, o cache, as variáveis de ambiente, o limite de CPU e o limite de memória. Você também pode criar um clone do repositório Git.

- O espaço de trabalho do Kubernetes requer o endpoint API do Kubernetes, a URL da imagem do construtor, o registro de imagem, o namespace, a NodePort, a Reivindicação de Volume Persistente (Persistent Volume Claim, PVC), o diretório de trabalho, as variáveis de ambiente, o limite de CPU e o limite de memória. Você também pode criar um clone do repositório Git.

A configuração do espaço de trabalho do pipeline tem muitos parâmetros comuns e outros parâmetros que são específicos para o tipo de espaço de trabalho, conforme descrito na tabela a seguir.

**Tabela 4-1. Áreas, detalhes e disponibilidade do espaço de trabalho**

Seleção	Descrição	Detalhes e disponibilidade
<b>Tipo</b>	Tipo de espaço de trabalho.	Disponível com Docker ou Kubernetes.
<b>Endpoint do Host</b>	Endpoint do host no qual as tarefas de integração contínua (continuous integration, CI) e personalizadas são executadas.	Disponível com o espaço de trabalho do Docker quando você seleciona o endpoint do host do Docker.  Disponível com o espaço de trabalho do Kubernetes quando você seleciona o endpoint da API do Kubernetes.
<b>URL da imagem do construtor</b>	Nome e localização da imagem do construtor. Um contêiner é criado usando essa imagem no host do Docker e no cluster do Kubernetes. As tarefas de integração contínua (CI) e personalizadas são executadas nesse contêiner.	Exemplo: <b>fedora:latest</b> A imagem do construtor deve ter <code>curl</code> ou <code>wget</code> .
<b>Registro de imagem</b>	Se a imagem do construtor estiver disponível em um registro, e se o registro exigir credenciais, você deverá criar um endpoint de Registro de Imagem e, depois, selecioná-lo aqui para que a imagem possa ser extraída do registro.	Disponível com os espaços de trabalho do Docker e do Kubernetes.
<b>Diretório de trabalho</b>	O diretório de trabalho é o local dentro do contêiner onde as etapas da tarefa de integração contínua (CI) são executadas e onde o código é clonado quando um webhook do Git dispara uma execução de pipeline.	Disponível com Docker ou Kubernetes.
<b>Namespace</b>	Se você não inserir um namespace, o Code Stream criará um nome exclusivo no cluster do Kubernetes que você especificou.	Específico para o espaço de trabalho do Kubernetes.

Tabela 4-1. Áreas, detalhes e disponibilidade do espaço de trabalho (continuação)

Seleção	Descrição	Detalhes e disponibilidade
Proxy	<p>Para se comunicar com o pod do workspace no cluster do Kubernetes, o Code Stream implanta uma única instância de proxy no namespace <code>codestream-proxy</code> para cada cluster do Kubernetes. Você pode selecionar o tipo <b>NodePort</b> ou <b>LoadBalancer</b> com base na configuração do cluster.</p> <p>A opção selecionada depende da natureza do cluster do Kubernetes implantado.</p> <ul style="list-style-type: none"> <li>■ Normalmente, se a URL do servidor da API do Kubernetes que é especificada no endpoint for exposta por meio de um dos nós principais, selecione <b>NodePort</b>.</li> <li>■ Se a URL do servidor da API do Kubernetes for exposta por um Balanceador de Carga, como no Amazon EKS (Elastic Kubernetes Service), selecione <b>LoadBalancer</b>.</li> </ul>	
NodePort	<p>O Code Stream usa NodePort para se comunicar com o contêiner executado no cluster do Kubernetes.</p> <p>Se você não selecionar uma porta, o Code Stream usará uma porta efêmera atribuída pelo Kubernetes. Você deve garantir que a configuração das regras de firewall permita o ingresso no intervalo de portas efêmeras (30000-32767).</p> <p>Se você inserir uma porta, deverá garantir que outro serviço no cluster ainda não a esteja usando e que as regras de firewall permitam a porta.</p>	Específico para o espaço de trabalho do Kubernetes.
Reivindicação de Volume Persistente	<p>Oferece uma maneira para o espaço de trabalho do Kubernetes persistir os arquivos em todas as execuções do pipeline. Quando você especifica um nome de reivindicação de volume persistente, ele pode armazenar os logs, os artefatos e o cache.</p> <p>Para obter mais informações sobre como criar uma reivindicação de volume persistente, consulte a documentação do Kubernetes em <a href="https://kubernetes.io/docs/concepts/storage/persistent-volumes/">https://kubernetes.io/docs/concepts/storage/persistent-volumes/</a>.</p>	Específico para o espaço de trabalho do Kubernetes.

Tabela 4-1. Áreas, detalhes e disponibilidade do espaço de trabalho (continuação)

Seleção	Descrição	Detalhes e disponibilidade
<b>Variáveis de ambiente</b>	Os pares de chave-valor especificados aqui estarão disponíveis para todas as tarefas de integração contínua (CI) e personalizadas em um pipeline quando ele for executado.	Disponível com Docker ou Kubernetes. As referências a variáveis podem ser transmitidas aqui. As variáveis de ambiente inseridas no espaço de trabalho são transmitidas para todas as tarefas de integração contínua (CI) e personalizadas no pipeline. Se as variáveis de ambiente não forem especificadas aqui, elas deverão ser inseridas claramente em cada tarefa de integração contínua (CI) e personalizada no pipeline.
<b>Limites de CPU</b>	Os limites de recursos de CPU para o contêiner de integração contínua (CI) ou de tarefa personalizada.	O padrão é 1.
<b>Limites de memória</b>	Os limites de memória para o contêiner de integração contínua (CI) ou de tarefa personalizada.	A unidade é MB.
<b>Clone do Git</b>	Quando você seleciona <b>Clone do Git</b> , e um webhook do Git invoca o pipeline, o código é clonado no espaço de trabalho (contêiner).	Se você não ativar o <b>Clone do Git</b> , deverá configurar outra tarefa de integração contínua (CI) explícita no pipeline para clonar o código primeiro e, depois, realizar outras etapas, como compilação e teste.
<b>Cache</b>	<p>O espaço de trabalho do Code Stream permite que você armazene em cache um conjunto de diretórios ou arquivos para acelerar as execuções de pipeline subsequentes. Alguns exemplos desses diretórios são <code>.m2</code> e <code>npm_modules</code>. "Se você não precisar de cache de dados entre execuções de pipeline, uma reivindicação de volume persistente não será necessária."</p> <p>Artefatos, como arquivos ou diretórios no contêiner, são armazenados em cache para reutilização nas execuções de pipeline. Por exemplo, é possível armazenar as pastas <code>node_modules</code> ou <code>.m2</code> em cache. <b>Cache</b> aceita uma lista de caminhos. Por exemplo:</p> <pre>workspace:   type: K8S   endpoint: K8S-Micro   image: fedora:latest   registry: Docker Registry   path: ''   cache:     - /path/to/m2     - /path/to/node_modules</pre>	<p>Específico para o tipo de espaço de trabalho.</p> <p>No espaço de trabalho do Docker, você acessa o <b>Cache</b> por um caminho compartilhado no host do Docker para persistir os dados, artefatos e logs armazenados em cache.</p> <p>No espaço de trabalho do Kubernetes, para permitir o uso do <b>Cache</b>, você deve fornecer uma reivindicação de volume persistente. Caso contrário, o <b>Cache</b> não estará disponível.</p>

Ao usar um endpoint API do Kubernetes no espaço de trabalho do pipeline, o Code Stream cria os recursos do Kubernetes necessários, como ConfigMap, Segredo e Pod, para executar a tarefa de integração contínua (CI) ou personalizada. O Code Stream se comunica com o contêiner usando a NodePort.

Para compartilhar os dados em todas as execuções do pipeline, você deve fornecer uma reivindicação de volume persistente, e o Code Stream montará a reivindicação de volume persistente no contêiner para armazenar os dados e a usará nas execuções seguintes do pipeline.

## Como planejar uma compilação nativa de CI/CD no Code Stream antes de usar o modelo de pipeline inteligente

Para criar um pipeline de integração contínua e entrega contínua (CI/CD) no Code Stream, é possível usar o modelo de pipeline inteligente de CI/CD. Para planejar a compilação nativa de CI/CD, colete as informações para o modelo de pipeline inteligente antes de criar o pipeline neste plano de exemplo.

Para criar um pipeline de CI/CD, é necessário planejar os estágios de integração contínua (continuous integration, CI) e entrega contínua (continuous delivery, CD) do pipeline.

Depois que você inserir as informações no modelo de pipeline inteligente e salvá-las, o modelo criará um pipeline que inclui estágios e tarefas. Ele também indica o destino de implantação da sua imagem com base nos tipos de ambiente selecionados, como Dev e Prod. O pipeline publicará a imagem do contêiner e executará as ações necessárias para executá-lo. Após a execução do pipeline, você poderá monitorar as tendências nas execuções de pipeline.

Quando um pipeline inclui uma imagem do Docker Hub, você deve garantir que essa imagem tenha `cURL` ou `wget` incorporado antes de executar o pipeline. Quando o pipeline é executado, o Code Stream baixa um arquivo binário que usa `cURL` ou `wget` para executar comandos.

Para obter informações sobre a configuração do espaço de trabalho, consulte [Como configurar o espaço de trabalho do pipeline](#).

## Como planejar o estágio de integração contínua (CI)

Para planejar o estágio de CI do pipeline, defina os requisitos externos e internos, e determine as informações que serão necessárias à parte de CI do modelo de pipeline inteligente. Aqui está um resumo.

Neste exemplo, é usado um espaço de trabalho do Docker.

Endpoints e repositórios que serão necessários:

- Um repositório de código-fonte Git no qual os desenvolvedores verificam o código. O Code Stream recebe o código mais recente para o pipeline quando os desenvolvedores confirmam as alterações.
- Um endpoint do Git para o repositório no qual reside o código-fonte do desenvolvedor.



- Um endpoint do Docker para o host de compilação do Docker que executará os comandos de compilação dentro de um contêiner.
- Um endpoint do Kubernetes para que o Code Stream possa implantar sua imagem em um cluster do Kubernetes.
- Uma imagem do construtor que cria o contêiner no qual os testes de integração contínua são executados.
- Um endpoint de registro de imagem para que o host de compilação do Docker possa receber a imagem do construtor a partir dele.

Você precisará de acesso a um projeto. O projeto agrupa todo o seu trabalho, incluindo pipeline, endpoints e painéis. Verifique se você é membro de um projeto no Code Stream. Se não for, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).

Você precisará de um webhook Git que permita que o Code Stream use o gatilho Git para disparar o pipeline quando os desenvolvedores confirmam alterações de código. Consulte [Como usar o gatilho Git no Code Stream para executar um pipeline](#).

Seus conjuntos de ferramentas de compilação:

- Seu tipo de compilação, como Maven.
- Todas as ferramentas de compilação pós-processo usadas, como JUnit, JaCoCo, Checkstyle e FindBugs.

Sua ferramenta de publicação:

- Uma ferramenta como o Docker que implantará seu contêiner de compilação.
- Uma tag de imagem, que é a ID de confirmação ou o número da compilação.

Seu espaço de trabalho de compilação:

- Um host de compilação do Docker, que é o endpoint do Docker.
- Um registro de imagem. A parte de CI do pipeline recebe a imagem do endpoint do registro selecionado. O contêiner executa as tarefas de CI e implanta sua imagem. Se o registro precisar de credenciais, será necessário criar um endpoint de Registro de imagem e, em seguida, selecioná-lo aqui para que o host possa extrair a imagem do registro.
- O URL da imagem do construtor que cria o contêiner no qual as tarefas de integração contínua são executadas.

## Como planejar o estágio de entrega contínua (CD)

Para planejar o estágio de CD do pipeline, defina os requisitos externos e internos, e determine as informações que serão inseridas na parte de CD do modelo de pipeline inteligente.

Endpoints que serão necessários:

- Um endpoint do Kubernetes para que o Code Stream possa implantar sua imagem em um cluster do Kubernetes.

## Arquivos e tipos de ambiente:

- Todos os tipos de ambiente em que o Code Stream implantará seu aplicativo, como Des e Prod. O modelo de pipeline inteligente cria os estágios e as tarefas em seu pipeline com base nos tipos de ambiente selecionados.

**Tabela 4-2. Estágios de pipeline criados pelo modelo de pipeline inteligente de CICD**

Conteúdo do pipeline	O que ele faz
Estágio de compilação-publicação	Compila e testa seu código, cria a imagem do construtor e publica a imagem no host do Docker.
Estágio de desenvolvimento	Usa um cluster de desenvolvimento do Amazon Web Services (AWS) para criar e implantar a imagem. Neste estágio, é possível criar um namespace no cluster e criar uma chave secreta.
Estágio de produção	Usa uma versão de produção do VMware Tanzu Kubernetes Grid Integrated Edition (anteriormente conhecido como VMware Enterprise PKS) para implantar sua imagem em um cluster Kubernetes de produção.

- Um arquivo YAML do Kubernetes selecionado por você na seção de CD do modelo de pipeline inteligente de CICD.

O arquivo YAML do Kubernetes inclui três seções necessárias para Namespace, Serviço e Implantação e uma seção opcional para Segredo. Se você planeja criar um pipeline baixando uma imagem de um repositório de propriedade privada, deverá incluir uma seção com o Segredo de configuração do Docker. Se o pipeline criado só usar imagens disponíveis publicamente, um segredo não será necessário. O seguinte arquivo YAML de amostra inclui quatro seções.

```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream
  namespace: codestream
---
apiVersion: v1
data:
  .dockerconfigjson:
eyJhdXRocyI6eyJodHRwczovL2luZ12345678901ci5pby92MS8iOmsidXNlcm5hbWUiOiJhdXRvbWV0aW9uYmV0YSIsInBhc3N3b3JkIjoieVkl3YXJlQDEyMyIsImVtYWlsIjoieYXV0b21hdGlvbmJldGF1c2VyQGdtYWlsLmNvbSIsImF1dGgiOiJZWYyYjIxaGRhbHZibUpsZEdFNlZrMTNZWEpsUURFeU13PT0ifX19
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

```

**Observação** O arquivo YAML do Kubernetes também é usado no modelo de pipeline inteligente de CD, como nos seguintes exemplos de caso de uso:

- [Como implantar meu aplicativo no Code Stream na implantação Azul-Verde](#)
- [Como faço para reverter minha implantação no Code Stream](#)
- [Como usar o gatilho do Docker no Code Stream para executar um pipeline de entrega contínua](#)

Para aplicar o arquivo no Modelo Inteligente, clique em **Selecionar** e selecione o arquivo YAML do Kubernetes. Em seguida, clique em **Processo**. O modelo de pipeline inteligente exibe os serviços e os ambientes de implantação disponíveis. Selecione um serviço, o endpoint do cluster e a estratégia de implantação. Por exemplo, para usar o modelo de implantação Canário, selecione **Canário** e digite o percentual da fase de implantação.

**Smart Template: CI/CD**

*Step 2 of 2*

Environment ⓘ ☒ Development ☒ Production

Kubernetes YAML files ⓘ     
 Processed files: codestream.yaml

Select service

Deployment name	Service	Namespace	Image
codestream-demo	codestream-demo	codestream	https://codestream/Myapp

1 services

Deployment

Environment	Cluster Endpoint	Namespace
Development	Dev-AWS-Cluster	codestream-454709
Production	Prod-AWS-Cluster	codestream

Image source ⓘ ☐ Docker trigger ☒ Pipeline runtime input

Deployment model ⓘ ☒ Canary ☐ Rolling upgrade ☐ Blue-Green

Phase 1 ⓘ 20 %

Rollback ☐

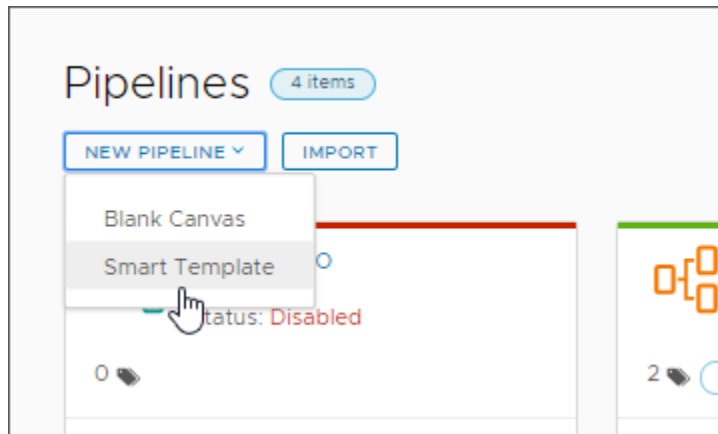
Health check URL ⓘ

Para ver um exemplo sobre como usar o modelo de pipeline inteligente para criar um pipeline para uma implantação Azul-Verde, consulte [Como implantar meu aplicativo no Code Stream na implantação Azul-Verde](#).

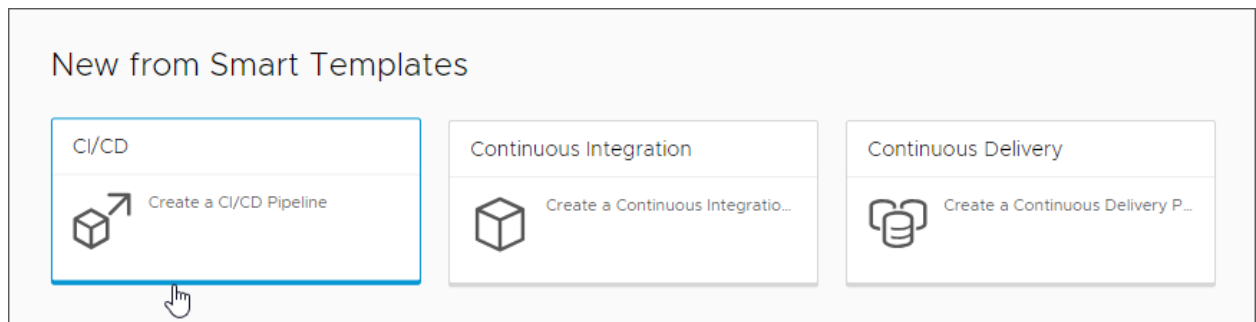
## Como criar o pipeline de CI/CD usando o modelo de pipeline inteligente

Depois de reunir todas as informações e definir o que você precisa, veja como criar um pipeline a partir do modelo de pipeline inteligente de CI/CD.

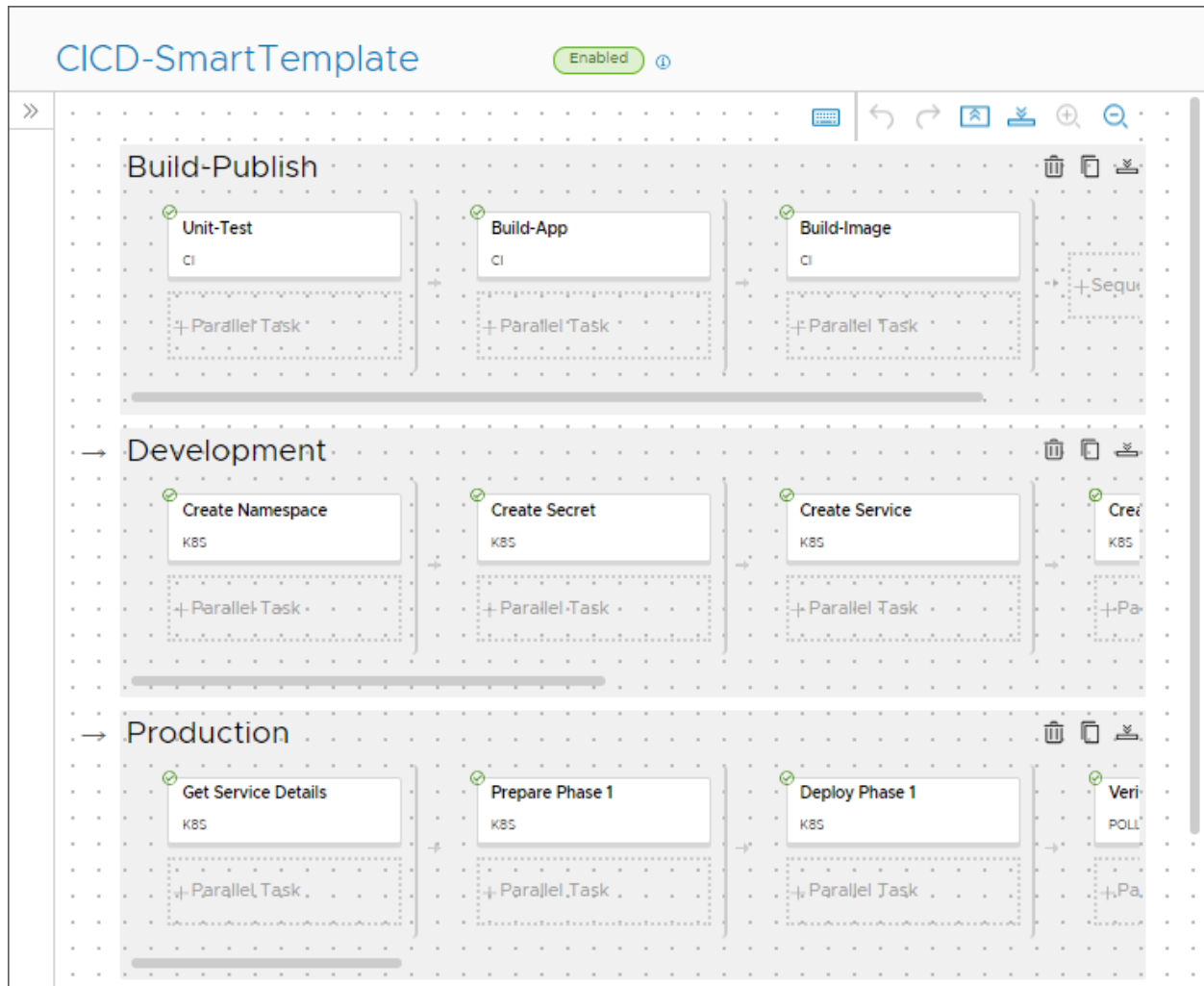
Em Pipelines, selecione **Novo Pipeline > Modelos Inteligentes**.



Você selecionará o modelo de pipeline inteligente de CI/CD.



Preencha o modelo e salve o pipeline com os estágios criados por ele. Se você precisar fazer quaisquer alterações finais, poderá editar o pipeline e salvá-lo.



Em seguida, ative e execute o pipeline. Após a execução, veja algumas coisas que você pode procurar:

- Verifique se o seu pipeline foi bem-sucedido. Clique em **Execuções** e pesquise por seu pipeline. Se falhou, corrija todos os erros e execute-o novamente.
- Verifique se o Git webhook está funcionando corretamente. A guia Git **Atividade** exibe os eventos. Clique em **Gatilhos > Git > Atividade**.
- Observe o painel de pipeline e examine as tendências. Clique em **Painéis** e pesquise pelo painel do seu pipeline. Também é possível criar um painel personalizado para relatar KPIs adicionais.

Para obter um exemplo detalhado, consulte [Como integrar continuamente o código do meu repositório do GitHub ou GitLab ao pipeline no Code Stream](#).

## Como planejar uma compilação nativa de integração contínua no Code Stream antes de usar o modelo de pipeline inteligente

Para criar um pipeline de integração contínua (continuous integration, CI) no VMware Code Stream, você pode usar o modelo de pipeline inteligente de integração contínua. Para planejar a compilação nativa de integração contínua, colete as informações para o modelo de pipeline inteligente antes de criar o pipeline neste plano de exemplo.

Quando você preenche o modelo de pipeline inteligente, ele cria um pipeline de integração contínua no repositório e executa as ações de modo que o pipeline possa ser executado. Após a execução do pipeline, você poderá monitorar as tendências nas execuções de pipeline.

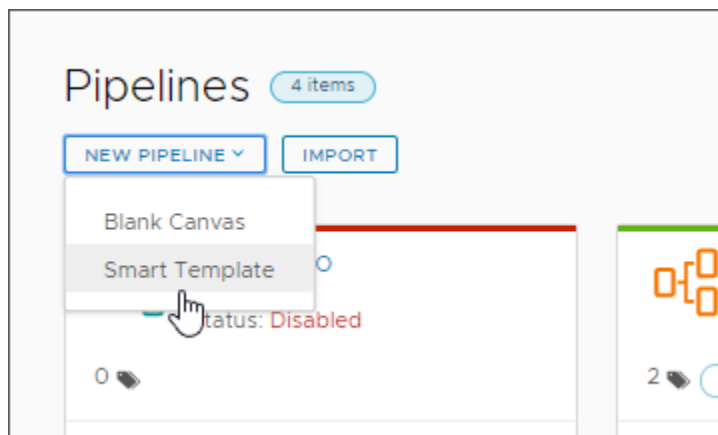
Para planejar sua compilação antes de usar o modelo de pipeline inteligente de integração contínua:

- Identifique um projeto que agrupará todo o seu trabalho, incluindo o pipeline, os endpoints e os painéis.
- Colete as informações para sua compilação, conforme descrito na parte sobre entrega contínua de [Como planejar uma compilação nativa de CI/CD no Code Stream antes de usar o modelo de pipeline inteligente](#).

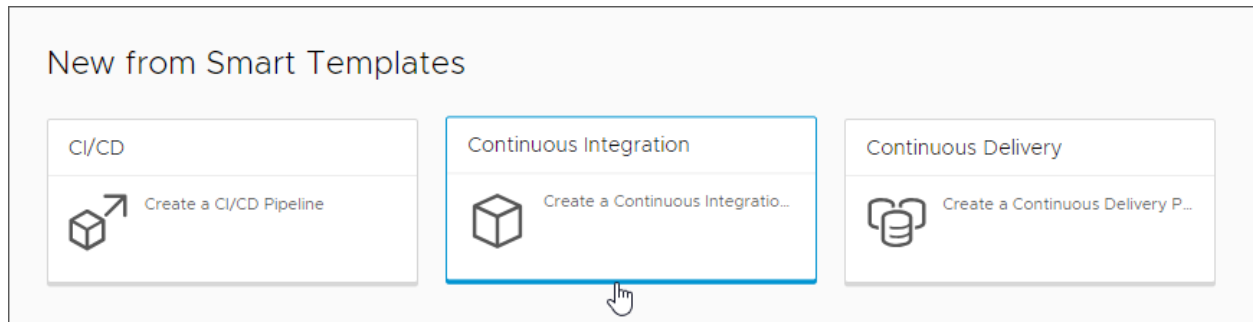
Por exemplo, adicione um endpoint do Kubernetes em que o Code Stream implantará o contêiner.

Na sequência, crie um pipeline usando o modelo de pipeline inteligente de integração contínua.

Em Pipelines, selecione **Modelos Inteligentes**.



Selecione o modelo de pipeline inteligente de integração contínua.



Para salvar o pipeline com os estágios que ele cria, você preenche o modelo e insere um nome para o pipeline. Para salvar o pipeline com os estágios que ele cria, clique em **Criar**.

O espaço de trabalho do pipeline do Code Stream é compatível com o Docker e o Kubernetes para tarefas de integração contínua e personalizadas.

Para obter informações sobre a configuração do espaço de trabalho, consulte [Como configurar o espaço de trabalho do pipeline](#).

Para fazer qualquer alteração final, você pode editar o pipeline. Em seguida, você pode ativar o pipeline e executá-lo. Após a execução do pipeline:

- Verifique se o seu pipeline foi bem-sucedido. Clique em **Execuções** e pesquise por seu pipeline. Se falhou, corrija todos os erros e execute-o novamente.
- Verifique se o Git webhook está funcionando corretamente. A guia Git **Atividade** exibe os eventos. Clique em **Gatilhos > Git > Atividade**.
- Observe o painel de pipeline e examine as tendências. Clique em **Painéis** e pesquise pelo painel do seu pipeline. Para incluir mais indicadores-chave de desempenho no relatório, você pode criar um painel personalizado.

Para obter um exemplo detalhado, consulte [Como integrar continuamente o código do meu repositório do GitHub ou GitLab ao pipeline no Code Stream](#).

## Como planejar uma compilação nativa de entrega contínua no Code Stream antes de usar o modelo de pipeline inteligente

Para criar um pipeline de entrega contínua (continuous delivery, CD) no Code Stream, você pode usar o modelo de pipeline inteligente de entrega contínua. Para planejar a compilação nativa de entrega contínua, colete as informações para o modelo de pipeline inteligente antes de criar o pipeline neste plano de exemplo.

Quando você preenche o modelo de pipeline inteligente, ele cria um pipeline de entrega contínua no repositório e executa as ações de modo que o pipeline possa ser executado. Após a execução do pipeline, você poderá monitorar as tendências nas execuções de pipeline.

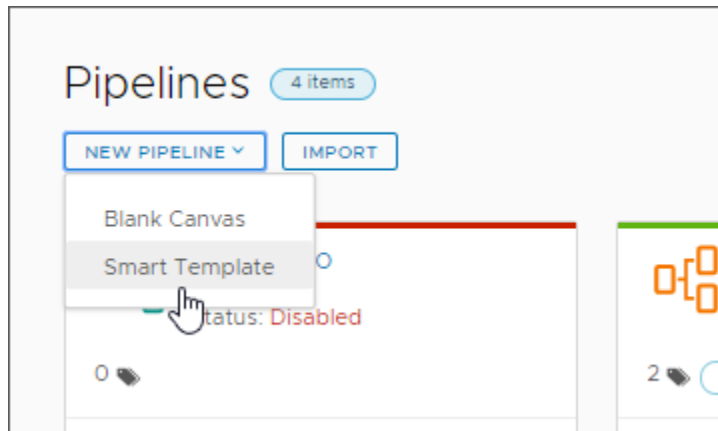


Para planejar sua compilação antes de usar o modelo de pipeline inteligente de entrega contínua:

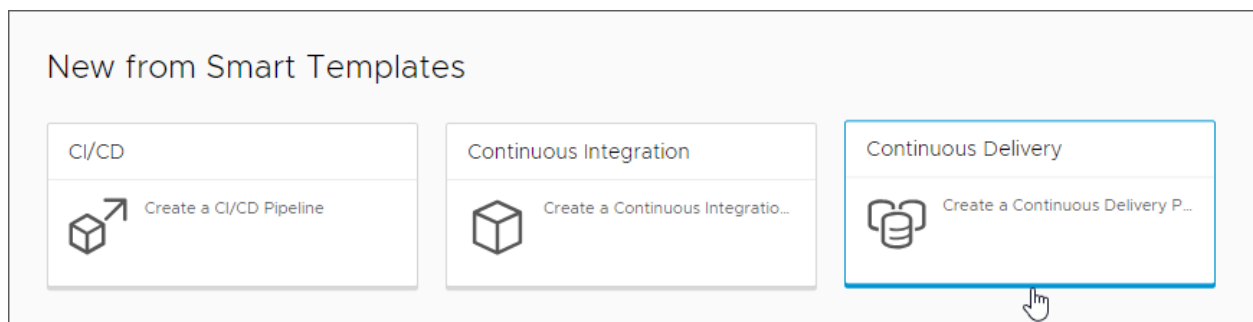
- Identifique um projeto que agrupará todo o seu trabalho, incluindo o pipeline, os endpoints e os painéis.
- Colete as informações para sua compilação, conforme descrito na parte sobre entrega contínua de [Como planejar uma compilação nativa de CI/CD no Code Stream antes de usar o modelo de pipeline inteligente](#). Por exemplo:
  - Adicione um endpoint do Kubernetes em que o Code Stream implantará o contêiner.
  - Prepare o arquivo YAML do Kubernetes que cria o Namespace, o Serviço e a Implantação. Para baixar uma imagem de um repositório de propriedade privada, o arquivo YAML deve incluir uma seção com o segredo de configuração do Docker.

Na sequência, crie um pipeline usando o modelo de pipeline inteligente de entrega contínua.

Em Pipelines, selecione **Modelos Inteligentes**.



Selecione o modelo de pipeline inteligente de entrega contínua.



Preencha o modelo e insira um nome para o pipeline. Para salvar o pipeline com os estágios que ele cria, clique em **Criar**.

O espaço de trabalho do pipeline do Code Stream é compatível com o Docker e o Kubernetes para tarefas de integração contínua e personalizadas.

Para obter informações sobre a configuração do espaço de trabalho, consulte [Como configurar o espaço de trabalho do pipeline](#).

Para fazer qualquer alteração final, você pode editar o pipeline. Em seguida, você pode ativar o pipeline e executá-lo. Após a execução do pipeline:

- Verifique se o seu pipeline foi bem-sucedido. Clique em **Execuções** e pesquise por seu pipeline. Se falhou, corrija todos os erros e execute-o novamente.
- Verifique se o Git webhook está funcionando corretamente. A guia Git **Atividade** exibe os eventos. Clique em **Gatilhos > Git > Atividade**.
- Observe o painel de pipeline e examine as tendências. Clique em **Painéis** e pesquise pelo painel do seu pipeline. Para incluir mais indicadores-chave de desempenho no relatório, você pode criar um painel personalizado.

Para obter um exemplo detalhado, consulte [Como integrar continuamente o código do meu repositório do GitHub ou GitLab ao pipeline no Code Stream](#).

## Planejando uma compilação nativa de CI/CD no Code Stream antes de adicionar tarefas manualmente

Para criar um pipeline de integração contínua e entrega contínua (CI/CD) no Code Stream, é possível adicionar manualmente estágios e tarefas. Para planejar sua compilação nativa de CI/CD, colete as informações necessárias e, em seguida, crie um pipeline e adicione manualmente os estágios e as tarefas a ele.

Será necessário planejar os estágios de integração contínua (continuous integration, CI) e de entrega contínua (continuous delivery, CD) do pipeline. Após criar o pipeline e executá-lo, você pode monitorar as tendências nas execuções do pipeline.

Quando um pipeline inclui uma imagem do Docker Hub, você deve garantir que essa imagem tenha `cURL` ou `wget` incorporado antes de executar o pipeline. Quando o pipeline é executado, o Code Stream baixa um arquivo binário que usa `cURL` ou `wget` para executar comandos.

O espaço de trabalho do pipeline do Code Stream é compatível com o Docker e o Kubernetes para tarefas de integração contínua e personalizadas.

Para obter informações sobre a configuração do espaço de trabalho, consulte [Como configurar o espaço de trabalho do pipeline](#).

## Como planejar requisitos externos e internos

Para planejar os estágios de CI e CD do pipeline, os requisitos a seguir indicam o que você deve fazer antes de criar o pipeline.

Neste exemplo, é usado um espaço de trabalho do Docker.

Para criar um pipeline a partir do plano de exemplo, você usará um host do Docker, um repositório Git, um Maven e várias ferramentas de compilação pós-processo.

Endpoints e repositórios que serão necessários:

- Um repositório de código-fonte Git no qual os desenvolvedores verificam o código. O Code Stream recebe o código mais recente para o pipeline quando os desenvolvedores confirmam as alterações.
- Um endpoint do Docker para o host de compilação do Docker que executará os comandos de compilação dentro de um contêiner.
- Uma imagem do construtor que cria o contêiner no qual os testes de integração contínua são executados.
- Um endpoint de registro de imagem para que o host de compilação do Docker possa receber a imagem do construtor a partir dele.

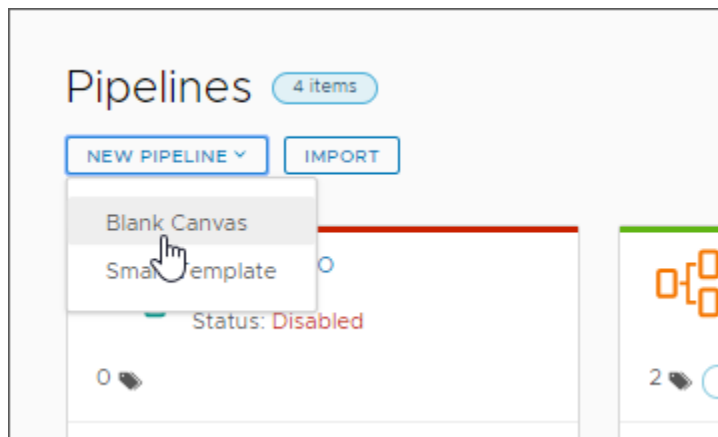
Você precisará de acesso a um projeto. O projeto agrupa todo o seu trabalho, incluindo pipeline, endpoints e painéis. Verifique se você é membro de um projeto no Code Stream. Se não for, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).

Você precisará de um webhook Git que permita que o Code Stream use o gatilho Git para disparar o pipeline quando os desenvolvedores confirmam alterações de código. Consulte [Como usar o gatilho Git no Code Stream para executar um pipeline](#).

## Como criar o pipeline de CI/CD e configurar o espaço de trabalho

Será necessário criar o pipeline e depois configurar o espaço de trabalho, os parâmetros de entrada de pipeline e as tarefas.

Para criar o pipeline, clique em **Pipelines > Novo Pipeline > Tela em Branco**.



Na guia Espaço de Trabalho, insira as informações de integração contínua:

- Inclua o host de compilação do Docker.
- Digite o URL da sua imagem do construtor.

- Selecione o endpoint de registro de imagem para que o pipeline possa receber a imagem dele. O contêiner executa as tarefas de CI e implanta sua imagem. Se o registro precisar de credenciais, será necessário primeiro criar o endpoint de registro de imagem e, em seguida, selecioná-lo aqui para que o host possa receber a imagem do registro.
- Adicione os artefatos que devem ser armazenados em cache. Para que uma compilação seja bem-sucedida, os artefatos, como diretórios, são baixados como dependências. O cache é o local onde esses artefatos residem. Por exemplo, os artefatos dependentes podem incluir o diretório `.m2` para Maven e o diretório `node_modules` para Node.js. Esses diretórios são armazenados em cache nas execuções do pipeline para economizar tempo durante as compilações.

The screenshot shows the 'Workspace' tab of the vRealize Automation interface. It contains a form for configuring continuous integration tasks. The 'Type' is set to 'Docker'. The 'Host endpoint' is 'codestream-ci-test'. The 'Builder Image URL' is 'automationbeta/cs-builder:latest'. The 'Image registry' is 'Docker Registry'. The 'Working directory' is empty. The 'Cache' section is at the bottom with a plus icon to add items.

Provide details about the container and host for running continuous integration tasks.

**Type \***  
☒ Docker ☐ Kubernetes

**Host endpoint \***  
 codestream-ci-test  
 Host endpoint for build location in CI task or for running Custom Integration Task code.

**Builder Image URL \***  
 automationbeta/cs-builder:latest  
 Name and location of the builder image. The CI tasks run on the container that the image creates.

**Image registry**  
 Docker Registry  
 The pipeline pulls the image from the selected registry endpoint. The container runs the CI tasks, and deploys your image. If the host can pull the image from the registry.

**Working directory**  
 CI pipeline tasks run steps for continuous integration. The working directory is the location where the steps run, and is where code is checked out.

**Cache**

Na guia de entrada, configure os parâmetros de entrada do pipeline.

- Se o pipeline for usar parâmetros de entrada de um evento de gatilho Git, Gerrit ou Docker, selecione o tipo de gatilho para Parâmetros de injeção automática. Os eventos podem incluir Alterar Assunto para o Gerrit ou Git ou Nome do Proprietário do Evento para o Docker. Se o seu pipeline não usar parâmetros de entrada transmitidos do evento, deixe Parâmetros de injeção automática definido como **Nenhum**.
- Para aplicar um valor e uma descrição a um parâmetro de entrada de pipeline, clique nos três pontos verticais, e clique em **Editar**. O valor digitado é usado como entrada para tarefas, estágios ou notificações.
- Para adicionar um parâmetro de entrada de pipeline, clique em **Adicionar**. Por exemplo, é possível adicionar `approvers` para exibir um valor padrão para cada execução, mas que pode ser substituído por um aprovador diferente em tempo de execução.
- Para adicionar ou remover um parâmetro injetado, clique em **Adicionar/Remover Parâmetro Injetado**. Por exemplo, remova um parâmetro não utilizado para reduzir a desordem na página de resultados e exibir apenas os parâmetros em uso.

**Input Parameters**

The input parameters for this pipeline are passed to the pipeline before it runs.

When you add input parameters, and star the most useful or unique input parameter for each pipeline, the parameter appears in locations like the pipeline execution cards. For example, if you include the committer ID (GIT\_COMMIT\_ID) as an input parameter, you can select it as the starred input parameter to identify which developer commits trigger a pipeline execution before the pipeline runs.

**Auto inject parameters**

☐ Gerrit ☐ Git ☐ Docker ☒ None

[ADD](#) [ADD/REMOVE INJECTED PARAMETERS](#)

Starred	Name	Value	Description
<input checked="" type="checkbox"/>	GIT_BRANCH_NAME		
<input checked="" type="checkbox"/>	GIT_CHANGE_SUBJECT		
<input checked="" type="checkbox"/>	GIT_COMMIT_ID		
<input checked="" type="checkbox"/>	GIT_EVENT_DESCRIPTION		
<input checked="" type="checkbox"/>	GIT_EVENT_OWNER_NAME		
<input checked="" type="checkbox"/>	GIT_EVENT_TIMESTAMP		
<input checked="" type="checkbox"/>	GIT_REPO_NAME		
<input checked="" type="checkbox"/>	GIT_SERVER_URL		

8 items

Configure o pipeline para testar seu código:

- Adicione e configure uma tarefa de CI.
- Inclua etapas para executar o `mvn test` no código.
- Para identificar quaisquer problemas após a execução da tarefa, execute ferramentas de compilação pós-processo, como o JUnit e o JaCoCo, o FindBugs e o Checkstyle.

**Task: Unit-Test** Notifications Rollback [VALIDATE TASK](#)

**Task name \*** Unit-Test  
Can contain alphanumeric (a-z, A-Z, 0-9), whitespace, hyphen(-), and underscore(\_) characters. Dot(.) is not allowed.

**Type \*** CI

**Precondition**

[SYNTAX GUIDE](#)

**Continue on failure** ☐

**Continuous Integration**  
A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.

**Steps**

```
1 cd demo-project
2 mvn test
```

**Preserve artifacts**  
Specify the paths of artifact to preserve.

**Export**  
Enter comma separated values

**JUnit** JUnit  
/demo-project

**JaCoCo** Jacoco  
/demo-project

**FindBugs** Findbugs  
/demo-project

**Checkstyle** Checkstyle  
/demo-project

Configure o pipeline para compilar seu código:

- Adicione e configure uma tarefa de CI.

- Incluir etapas que executam `mvn clean install` no seu código.
- Inclua o local e o nome de arquivo JAR para que ele preserve o artefato.

**Task Build-App** Notifications Rollback VALIDATE TASK

**Task name \*** Build-App  
Can contain alphanumeric (a-z, A-Z, 0-9), whitespace, hyphen(-), and underscore(\_) characters. Dot(.) is not allowed.

**Type \*** CI

**Precondition**  [SYNTAX GUIDE](#)

**Continue on failure** ☐

**Continuous Integration**  
A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.

**Steps \***

```
1 cd demo-project;
2 mvn clean install -DskipTests
```

**Preserve artifacts**  
Specify the paths of artifact to preserve. [+](#)

**Export**  
Enter comma separated values

**JUnit**  
JUnit  
/demo-project [+](#)

**JaCoCo**  
Jacoco  
/demo-project [+](#)

**Findbugs**  
Findbugs  
/demo-project [+](#)

**Checkstyle**  
Checkstyle  
/demo-project [+](#)

Configure o pipeline para publicar a imagem no host do Docker:

- Adicione e configure uma tarefa de CI.
- Adicione etapas que vão confirmar, exportar, compilar e enviar por push a sua imagem.
- Adicione a chave de exportação de `IMAGE` para a tarefa seguinte consumir.

The screenshot shows the 'Build-Image' task configuration window. At the top, there are tabs for 'Task', 'Build-Image', 'Notifications', and 'Rollback', along with a 'VALIDATE TASK' button. The 'Task name' is 'Build-Image' with a note about character restrictions. The 'Type' is set to 'CI'. There is a 'Precondition' field with a 'SYNTAX GUIDE' button. The 'Continue on failure' checkbox is unchecked. The 'Continuous Integration' section includes a note about Docker host setup. The 'Steps' section contains a code editor with the following script:

```
1 cd demo-project
2 export IMAGE=automationbeta/demo-cicd-smart-template:{{executionIndex}}
3 export DOCKER_HOST=http://10.211.211.27:4243
4 docker login --username=automationbeta --password
5 docker build -t $IMAGE --file ./docker/Dockerfile .
6 docker push $IMAGE
```

Below the code editor are sections for 'Preserve artifacts', 'Export' (with an 'IMAGE' button), and test tool configurations for 'JUnit', 'JaCoCo', 'FindBugs', and 'Checkstyle', each with 'Label' and 'Path' fields.

Depois de configurar o espaço de trabalho, os parâmetros de entrada, as tarefas de teste e as tarefas de compilação, salve seu pipeline.

## Como ativar e executar o pipeline

Depois de configurar seu pipeline com estágios e tarefas, você pode salvar e habilitar o pipeline.

Em seguida, aguarde até que o pipeline seja executado e concluído e verifique se foi bem-sucedido. Se falhou, corrija todos os erros e execute-o novamente.

Após o pipeline ter sido executado com êxito, estes são alguns aspectos que você pode querer confirmar:

- Examine a execução do pipeline e visualize os resultados das etapas da tarefa.
- No espaço de trabalho da execução do pipeline, localize os detalhes sobre seu contêiner e o repositório Git clonado.
- No espaço de trabalho, confira os resultados das suas ferramentas pós-processo e verifique se há erros, problemas de cobertura de código, bugs e problemas de estilo.
- Confirme se o seu artefato está preservado. Confirme também se a imagem foi exportada com o nome e o valor IMAGE.
- Vá para o repositório do Docker e verifique se o pipeline publicou o seu contêiner.

Para obter um exemplo detalhado que mostra como o Code Stream integra continuamente seu código, consulte [Como integrar continuamente o código do meu repositório do GitHub ou GitLab ao pipeline no Code Stream](#).

## Planejando uma reversão no Code Stream

Se a execução de um pipeline falhar, você poderá usar a reversão para retornar seu ambiente a um estado anteriormente estável. Para usar a reversão, planeje um fluxo de reversão e entenda como implementá-lo.

Um fluxo de reversão determina as etapas obrigatórias para reverter uma falha na implantação. O fluxo tem o formato de um pipeline de reversão que inclui uma ou mais tarefas sequenciais que variam dependendo do tipo de implantação que foi executada e falhou. Por exemplo, a implantação e a reversão de um aplicativo tradicional são diferentes da implantação e da reversão de um aplicativo de contêiner.

Para retornar a um bom estado de implantação, um pipeline de reversão normalmente inclui tarefas para:

- Limpar estados ou ambientes.
- Executar um script especificado pelo usuário para reverter as alterações.
- Implantar uma revisão anterior de uma implantação.

Para adicionar a reversão a um pipeline de implantação existente, anexe o pipeline de reversão às tarefas ou estágios no pipeline de implantação que deseja reverter antes de executá-lo.

## Como configurar a reversão

Para configurar a reversão na implantação, é necessário:

- Criar um pipeline de implantação.
- Identificar os possíveis pontos de falha no pipeline de implantação que acionarão a reversão para que você possa anexar o pipeline de reversão. Por exemplo, é possível anexar seu pipeline de reversão a uma condição ou tipo de tarefa de sondagem no pipeline de implantação que verifica se uma tarefa anterior foi concluída com êxito. Para obter informações sobre tarefas de condição, consulte [Como usar associações de variáveis em uma tarefa de condição para executar ou parar um pipeline no Code Stream](#).
- Determine o escopo da falha que disparará o pipeline de reversão, como uma falha de tarefa ou estágio. Também é possível anexar a reversão a um estágio.
- Decida quais tarefas ou tarefas de reversão serão executadas em caso de falha. Você criará seu pipeline de reversão com essas tarefas.

É possível criar manualmente um pipeline de reversão ou o Code Stream pode criar um para você.

- Usando uma tela em branco, é possível criar manualmente um pipeline de reversão que segue um fluxo em paralelo a um pipeline de implantação existente. Em seguida, anexe o pipeline de reversão a uma ou mais tarefas no pipeline de implantação que dispara a reversão em caso de falha.



- Usando um modelo de pipeline inteligente, é possível configurar um pipeline de implantação com a ação de reversão. Em seguida, o Code Stream cria automaticamente um ou mais pipelines de reversão padrão com tarefas predefinidas que reverterem a implantação em caso de falha.

Para obter um exemplo detalhado sobre como configurar um pipeline de CD com reversão usando um modelo de pipeline inteligente, consulte o [Como faço para reverter minha implantação no Code Stream](#).

## O que acontece se meu pipeline de implantação tiver várias tarefas ou estágios com reversão

Se você tiver várias tarefas ou tarefas e estágios com a reversão adicionada, lembre-se de que a sequência de reversão varia.

**Tabela 4-3. Como determinar a sequência de reversão**

Se você adicionar a reversão para...	Quando a reversão ocorre...
Tarefas paralelas	Se uma das tarefas paralelas falhar, a reversão para essa tarefa ocorrerá depois que todas as tarefas paralelas forem concluídas ou falharem. A reversão não ocorrerá imediatamente após a falha na tarefa.
A tarefa em um estágio e o estágio	Se uma tarefa falhar, a reversão da tarefa será executada. Se a tarefa estiver em um grupo de tarefas paralelas, a reversão da tarefa será executada depois que todas as tarefas paralelas forem concluídas ou falharem. Depois que a reversão da tarefa for concluída ou falhar ao ser concluída, a reversão do estágio será executada.

Considere um pipeline com:

- Um estágio de produção com reversão.
- Um grupo de tarefas paralelas, cada tarefa com sua própria reversão.

A tarefa chamada **UPD Deploy US** tem o pipeline de reversão **RB\_Deploy\_US**. Se o **UPD Deploy US** falhar, a reversão seguirá o fluxo definido no pipeline de **RB\_Deploy\_US**.

The screenshot displays the vRealize Automation Code Stream interface for a workflow titled "RollbackUpgrade-Example", which is currently "Enabled". The interface is divided into four main sections: Workspace, Input, Model, and Output. The "Model" section is active, showing a workflow diagram. The workflow starts with a "Production" stage, which contains three sequential tasks: "UPD Deploy US", "UPD Deploy UK", and "UPD Deploy AU", all using the "Kubernetes" provider. Below these tasks is a "+ Parallel Task" option. The workflow then transitions to a "Rollback" stage, which contains a single task named "RB\_Deploy\_US", which is a pipeline. The "Rollback" stage is highlighted with a blue bar. The right pane shows the details of the "RB\_Deploy\_US" pipeline. At the bottom of the interface, there are three buttons: "SAVE", "RUN", and "CLOSE", along with a status indicator "Last saved 12 days ago".

Se o **UPD Deploy US** falhar, o pipeline de **RB\_Deploy\_US** será executado após o **UPD Deploy UK** e o **UPD Deploy AU** terem sido concluídos ou falharem. A reversão não ocorrerá imediatamente após o **UPD Deploy US** falhar. E como o estágio de produção também tem reversão, após a execução do pipeline de **RB\_Deploy\_US**, o pipeline de reversão de estágio será executado.

# Tutoriais para usar o Code Stream

# 5

O Code Stream modela e oferece suporte ao seu ciclo de vida de lançamento do DevOps e testa e libera continuamente seus aplicativos para ambientes de desenvolvimento e ambientes de produção.

Você já configurou tudo o que precisa para que possa usar o Code Stream. Consulte [Capítulo 2 Como configurar o Code Stream para modelar meu processo de liberação](#).

Agora, é possível criar pipelines que automatizam a compilação e o teste do código do desenvolvedor antes de liberá-lo para produção. O Code Stream pode implantar aplicativos tradicionais ou baseados em contêiner.

**Tabela 5-1. Como usar o Code Stream no seu ciclo de vida do DevOps**

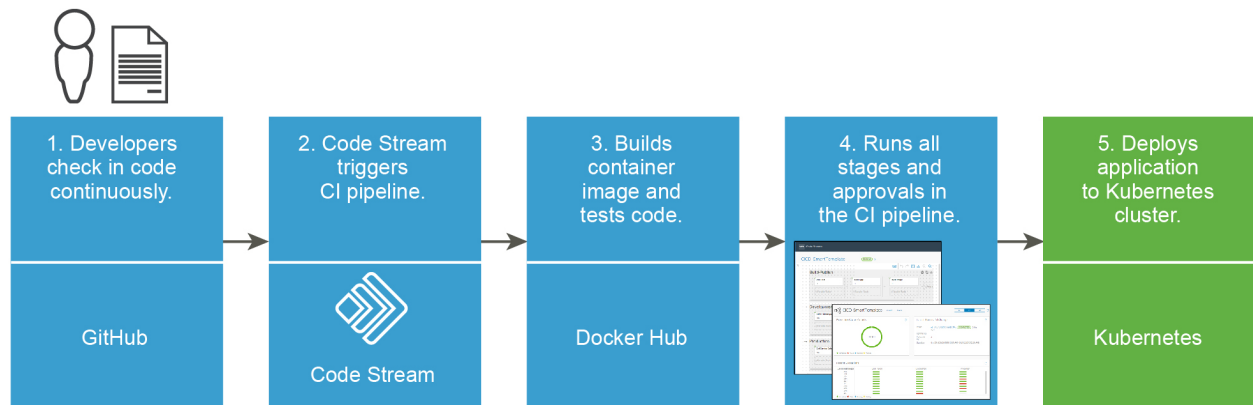
Recursos	Exemplos do que você pode fazer
Use o recurso de compilação nativo no Code Stream.	<p>Crie pipelines de Integração e Entrega Contínuas (Continuous Integration and Delivery, CICD), Integração Contínua (Continuous Integration, CI) e Entrega Contínua (Continuous Delivery, CD) que integram, containerizam e entregam seu código continuamente.</p> <ul style="list-style-type: none"><li>■ Use um modelo de pipeline inteligente que crie um pipeline para você.</li><li>■ Adicione manualmente os estágios e tarefas a um pipeline.</li></ul>
Libere seus aplicativos e automatize as liberações.	<p>Integre e libere seus aplicativos de várias maneiras.</p> <ul style="list-style-type: none"><li>■ Integre continuamente seu código de um repositório GitHub ou GitLab no seu pipeline.</li><li>■ Integre um host do Docker para executar tarefas de Integração Contínua conforme documentado neste artigo do blog (em inglês) <a href="#">Creating a Docker host for vRealize Automation Code Stream</a>.</li><li>■ Automatize a implantação do seu aplicativo usando um modelo de nuvem YAML.</li><li>■ Automatize a implantação do seu aplicativo em um cluster do Kubernetes.</li><li>■ Libere seu aplicativo para uma implantação Azul-Verde.</li><li>■ Integre o Code Stream às suas ferramentas de compilação, teste e implantação.</li><li>■ Use uma REST API que integre o Code Stream a outros aplicativos.</li></ul>
Acompanhe tendências, métricas e indicadores-chave de desempenho (KPIs).	<p>Crie painéis personalizados e obtenha informações sobre o desempenho de seus pipelines.</p>
Resolva os problemas.	<p>Quando a execução de um pipeline falha, faça com que o Code Stream crie um tíquete do Jira.</p>

Este capítulo inclui os seguintes tópicos:

- Como integrar continuamente o código do meu repositório do GitHub ou GitLab ao pipeline no Code Stream
- Como automatizar a liberação de um aplicativo implantado a partir de um modelo de nuvem YAML no Code Stream
- Como automatizar a liberação de um aplicativo no Code Stream para um cluster do Kubernetes
- Como implantar meu aplicativo no Code Stream na implantação Azul-Verde
- Como integrar minhas próprias ferramentas de compilação, teste e implantação com o Code Stream
- Como usar as propriedades de recursos de uma tarefa de modelo de nuvem na minha próxima tarefa
- Como usar uma REST API para integrar o Code Stream a outros aplicativos
- Como alavancar o pipeline como código no Code Stream

## Como integrar continuamente o código do meu repositório do GitHub ou GitLab ao pipeline no Code Stream

Como desenvolvedor, você deseja integrar continuamente o código de um repositório GitHub ou de um repositório corporativo GitLab. Sempre que os desenvolvedores atualizam o código e confirmam alterações no repositório, o Code Stream pode detectar essas alterações e disparar o pipeline.



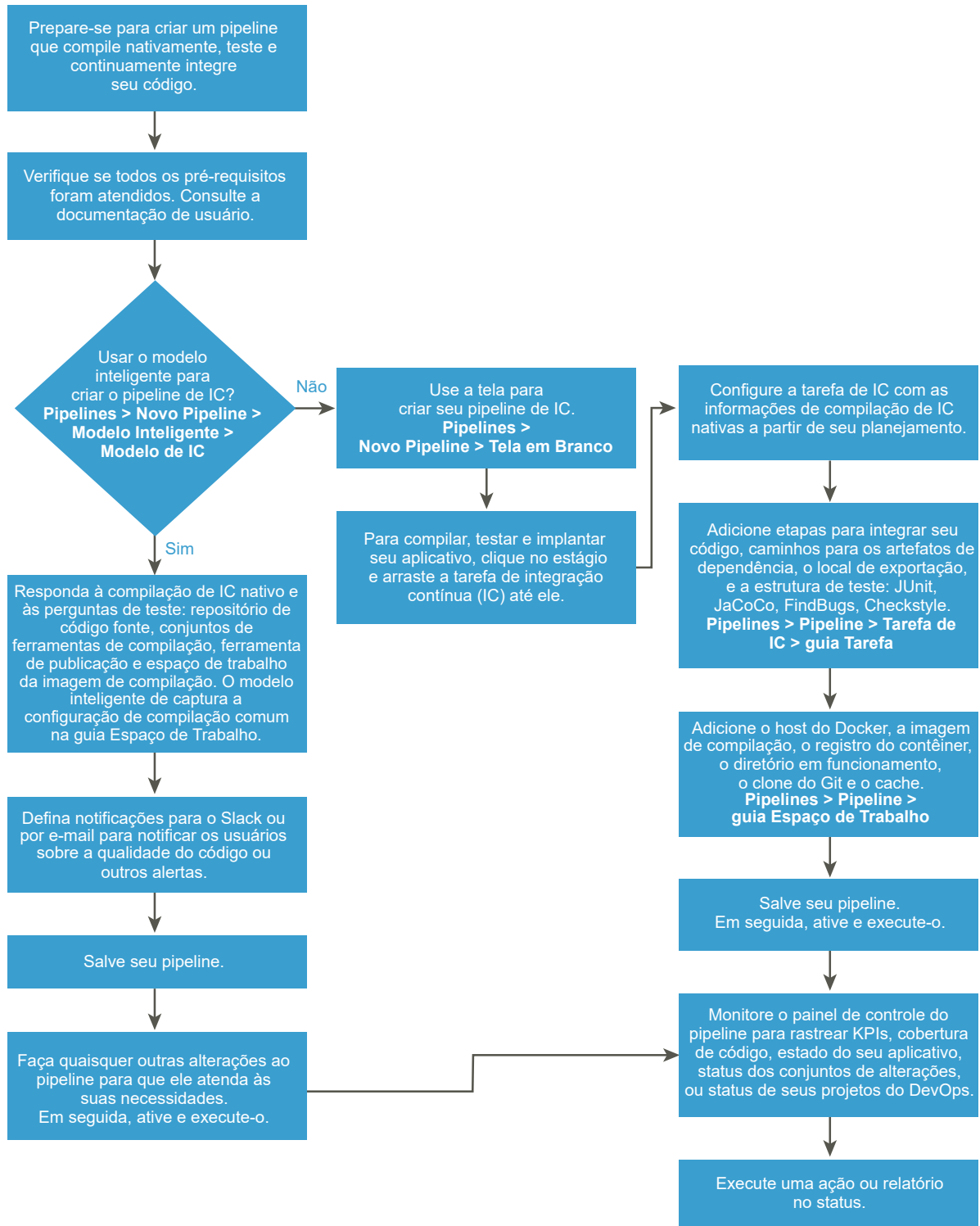
Para que o Code Stream dispare seu pipeline nas alterações de código, use o gatilho Git. O Code Stream acionará seu pipeline todas as vezes que você confirmar alterações na seu código.

O espaço de trabalho do pipeline do Code Stream é compatível com o Docker e o Kubernetes para tarefas de integração contínua e personalizadas.

Para obter mais informações sobre a configuração do espaço de trabalho, consulte [Como configurar o espaço de trabalho do pipeline](#).

O fluxograma a seguir mostra o fluxo de trabalho que você pode realizar se usar um modelo de pipeline inteligente para criar seu pipeline ou construir o pipeline manualmente.

Figura 5-1. Fluxo de trabalho que usa um modelo de pipeline inteligente ou cria um pipeline manualmente



No exemplo a seguir, é usado um espaço de trabalho do Docker.

Para compilar seu código, use um host do Docker. É possível usar o JUnit e o JaCoCo como ferramentas de estrutura de teste, que executam testes de unidade e cobertura de código, que você inclui no pipeline.

Em seguida, você pode usar o modelo de pipeline inteligente de integração contínua que cria um pipeline de integração contínua que constrói, testa e implanta seu código para o cluster Kubernetes da equipe do projeto na AWS. Para armazenar os artefatos de dependência de código para sua tarefa de integração contínua, o que pode economizar tempo em construções de código, você pode usar um cache.

Na tarefa de pipeline que cria e testa seu código, você pode incluir várias etapas de integração contínua. Essas etapas podem residir no mesmo diretório de trabalho no qual o Code Stream clona o código-fonte quando o pipeline é acionado.

Para implantar seu código no cluster Kubernetes, você pode usar uma tarefa Kubernetes no seu pipeline. Você deve então habilitar e executar seu pipeline. Em seguida, fará uma alteração em seu código no repositório e observará o pipeline disparar. Para monitorar e relatar as tendências do pipeline após a execução do pipeline, use os dashboards.

No exemplo a seguir, para criar um pipeline de integração contínua que integra continuamente seu código ao pipeline, você usa o modelo de pipeline inteligente de integração contínua. Neste exemplo, é usado um espaço de trabalho do Docker.

Outra alternativa é a criação manual do pipeline, adicionando estágios e tarefas a ele. Para obter mais informações sobre como planejar uma compilação de integração contínua e criar manualmente o pipeline, consulte [Planejando uma compilação nativa de CI/CD no Code Stream antes de adicionar tarefas manualmente](#).

#### Pré-requisitos

- Planeje sua compilação de integração contínua. Consulte [Como planejar uma compilação nativa de integração contínua no Code Stream antes de usar o modelo de pipeline inteligente](#).
- Verifique a existência de um repositório de códigos-fonte GitLab. Para obter ajuda, consulte o administrador do Code Stream.
- Adicione um endpoint do Git. Para obter um exemplo, consulte [Como usar o gatilho Git no Code Stream para executar um pipeline](#).
- Para que o Code Stream detecte as alterações no repositório GitHub ou GitLab e dispare um pipeline quando houver alterações, adicione um webhook. Para obter um exemplo, consulte [Como usar o gatilho Git no Code Stream para executar um pipeline](#).
- Adicione um endpoint de host do Docker, que cria um contêiner para a tarefa de integração contínua que várias tarefas de integração contínua podem usar. Para obter mais informações sobre endpoint, consulte [O que são endpoints no Code Stream](#).
- Obtenha a URL da imagem, o host de compilação e a URL da imagem de compilação. Para obter ajuda, consulte o administrador do Code Stream.
- Verifique se você usa JUnit e JaCoCo para as ferramentas de estrutura de teste.

- Configure uma instância externa para sua construção de integração contínua: Jenkins, TFS ou Bamboo. O plug-in Kubernetes implanta seu código. Para obter ajuda, consulte o administrador do Code Stream.

## Procedimentos

- 1 Siga os pré-requisitos.
- 2 Para criar o pipeline usando o modelo de pipeline inteligente, abra o modelo de pipeline inteligente de integração contínua e preencha o formulário.
  - a Clique em **Pipelines > Novo Pipeline > Modelo Inteligente > Integração Contínua**.
  - b Responda às perguntas no modelo sobre o repositório de códigos-fonte, os conjuntos de ferramentas de compilação, a ferramenta de publicação e o espaço de trabalho de imagem de compilação.
  - c Adicione notificações por e-mail ou notificações do Slack para sua equipe.
  - d Para que o modelo de pipeline inteligente crie o pipeline, clique em **Criar**.
  - e Para fazer outras alterações no pipeline, clique em **Editar**, faça as alterações e clique em **Salvar**.
  - f Ative o pipeline e execute-o.
- 3 Para criar o pipeline manualmente, inclua estágios e tarefas na tela e inclua suas informações de construção de integração contínua nativa na tarefa de integração contínua.
  - a Clique em **Pipelines > Novo Pipeline > Tela em Branco**.
  - b Clique no estágio e arraste as várias tarefas de integração contínua do painel de navegação até esse estágio.
  - c Para configurar a tarefa de integração contínua, clique nela e clique na guia **Tarefa**.
  - d Adicione as etapas que integram continuamente o código.
  - e Inclua os caminhos para os artefatos de dependência.
  - f Adicione o local de exportação.
  - g Adicione as ferramentas de estrutura de teste que serão usadas.
  - h Adicione o host do Docker e a imagem de compilação.
  - i Adicione o registro do contêiner, o diretório de trabalho e o cache.
  - j Salve o pipeline e, em seguida, ative-o.
- 4 Faça uma alteração no código em seu repositório GitHub ou GitLab.  
O gatilho Git ativa o pipeline, que começa a ser executado.
- 5 Para verificar se a alteração do código disparou o pipeline, clique em **Gatilhos > Git > Atividade**.



- 6 Para exibir a execução do pipeline, clique em **Execuções** e verifique se as etapas criaram e exportaram a imagem de compilação.

The screenshot displays the vRealize Automation Code Stream interface. On the left, a sidebar contains navigation links: Dashboards, Executions, User Operations, Pipelines, and Manage. The main area shows the execution details for a pipeline named 'CICD-SmartTemplate #51'. The pipeline is in a 'COMPLETED' state. The progress bar indicates two stages: 'Build-Publish' and 'Development'. The 'Build-Publish' stage is currently active, showing a task named 'Build-Image' with a status of 'COMPLETED'. The task details include its type (CI), duration (5s), and a log of steps executed successfully, such as setting environment variables, logging into Docker, and building the image. The 'Development' stage shows tasks like 'Create Namespace', 'Create Secret', 'Create Service', and 'Create Deployment'.

- 7 Para monitorar o painel do pipeline e rastrear KPIs e tendências, clique em **Painéis > Painéis de Pipeline**.

### Resultados

Parabéns! Você criou um pipeline que integra continuamente o código a partir de um repositório GitHub ou GitLab ao pipeline e implanta a imagem de compilação.

### Próximo passo

Para saber mais, consulte [Mais recursos para administradores e desenvolvedores do Code Stream](#).

## Como automatizar a liberação de um aplicativo implantado a partir de um modelo de nuvem YAML no Code Stream

Como desenvolvedor, você precisa de um pipeline que obtenha um modelo de nuvem de automação de uma instância do GitHub local todas as vezes que uma alteração é confirmada.

Você precisa do pipeline para implantar um aplicativo WordPress no Amazon Web Services (AWS) EC2 ou em um centro de dados. O Code Stream chama o modelo de nuvem do pipeline e automatiza a integração contínua e a entrega contínua (CICD) desse modelo de nuvem para implantar seu aplicativo.

Para criar e acionar seu pipeline, você precisará de um Modelo de Nuvem VMware.

Para **Origem do modelo de nuvem** na tarefa de modelo de nuvem do Code Stream, é possível selecionar:

- **Modelo do Cloud Assembly** como controle de origem. Nesse caso, não é necessário um repositório GitLab ou GitHub.
- **Controle de Origem** se você usar o GitLab ou o GitHub para controle de origem. Nesse caso, é necessário ter um webhook do Git e disparar o pipeline por meio desse webhook.

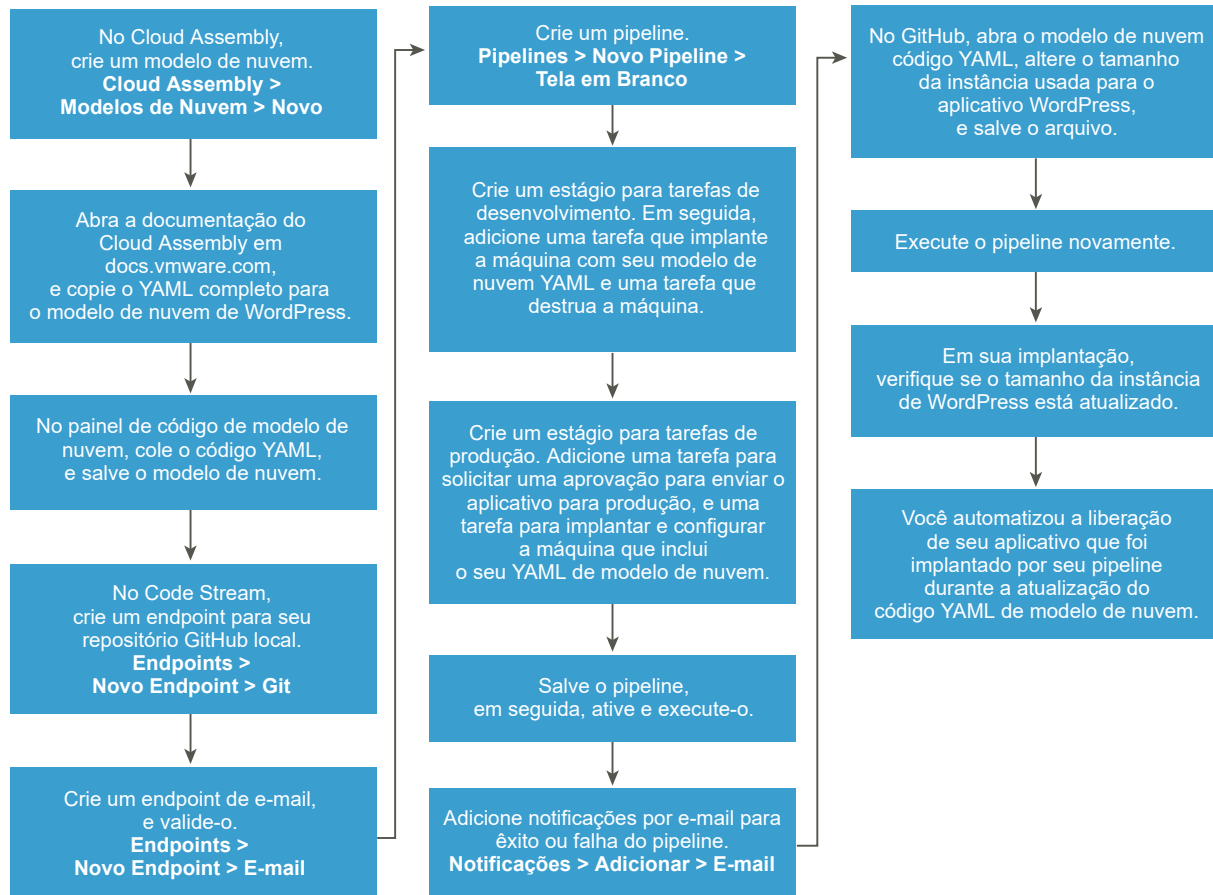
Se você tiver um modelo de nuvem YAML no repositório do GitHub e quiser usar esse modelo de nuvem no pipeline, será necessário fazer o seguinte:

- 1 No Cloud Assembly, envie o modelo de nuvem ao seu repositório do GitHub.
- 2 No Code Stream, crie um endpoint do Git. Em seguida, crie um webhook do Git que use seu endpoint do Git e seu pipeline.
- 3 Para disparar o pipeline, atualize qualquer arquivo no repositório do GitHub e confirme a alteração.

Se você não tiver um modelo de nuvem do YAML no repositório do GitHub e quiser usar um modelo de nuvem do controle de origem, siga este procedimento para saber como. Ele mostra como criar um modelo de nuvem para um aplicativo WordPress e dispará-lo a partir de um repositório do GitHub local. Sempre que você fizer uma alteração no modelo de nuvem do YAML, o pipeline será disparado e automatizará a liberação do seu aplicativo.

- No Cloud Assembly, você adicionará uma conta de nuvem, adicionará uma zona de nuvem e criará o modelo de nuvem.
- No Code Stream, você adicionará um endpoint para o repositório do GitHub local que hospeda seu modelo de nuvem. Em seguida, adicionará esse modelo de nuvem ao pipeline.

Este exemplo de caso de uso mostra como usar um modelo de nuvem a partir de um repositório do GitHub local.

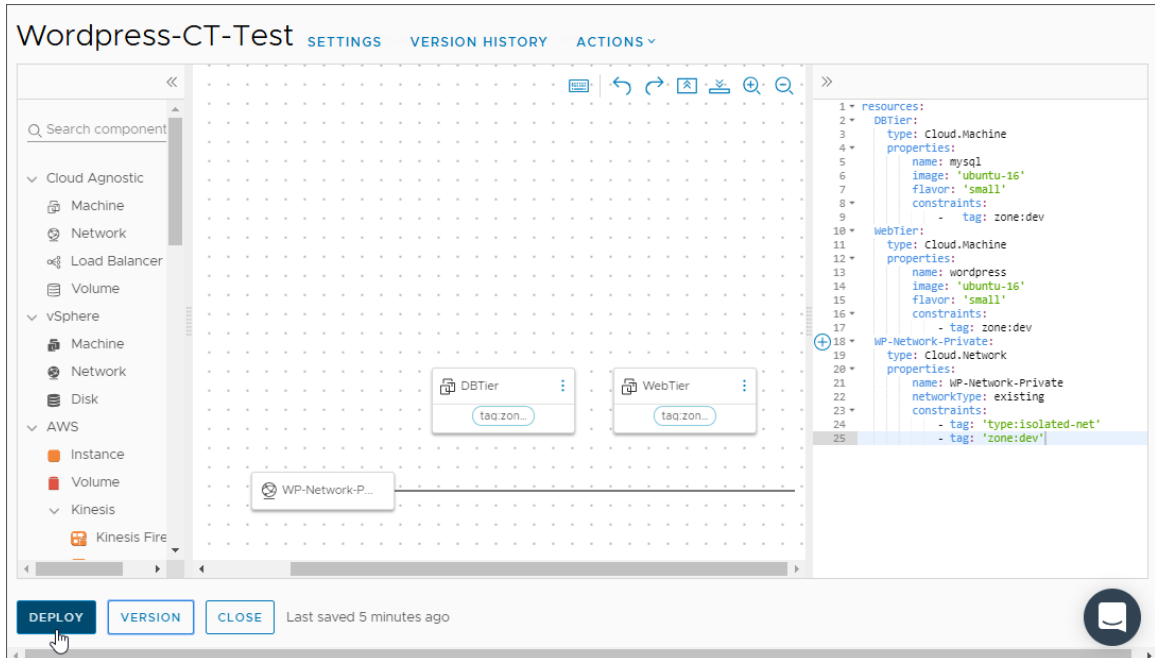


### Pré-requisitos

- Adicione uma conta de nuvem e uma zona de nuvem na infraestrutura do vRealize Automation Cloud Assembly. Consulte a documentação do vRealize Automation Cloud Assembly.
- Para criar o modelo de nuvem no procedimento a seguir, copie o código YAML do WordPress para a área de transferência. Consulte o código YAML do modelo de nuvem no caso de uso do WordPress na documentação do vRealize Automation Cloud Assembly.
- Adicione o código YAML para o aplicativo WordPress à instância do GitHub.
- Adicione um webhook para o gatilho do Git para que o pipeline possa extrair o código YAML sempre que você confirmar alterações nele. No Code Stream, clique em **Gatilhos > Git > Webhooks para Git**.
- Para trabalhar com uma tarefa de modelo de nuvem, você deve ter qualquer uma das funções do Cloud Assembly.

## Procedimentos

- 1 No Cloud Assembly, siga estas etapas.
  - a Clique em **VMware Cloud Templates** e crie um modelo de nuvem e uma implantação para o aplicativo WordPress.
  - b Cole o código YAML do WordPress copiado para a área de transferência no modelo de nuvem e implante-o.



## 2 No Code Stream, crie endpoints.

- a Crie um endpoint do Git para o repositório GitHub local no qual o arquivo YAML reside.
- b Adicione um endpoint de e-mail que possa notificar os usuários sobre o status do pipeline quando ele é executado.

## Add Endpoint

Project *	Codestream
Type *	Email
Name *	Enter value here
Description	
Mark as restricted	<input type="checkbox"/> non-restricted
Sender's Address *	eg: abc@xyz.com
Encryption Method *	SSL
Outbound Host *	myimap.org
Outbound Port *	Port number
Outbound Protocol *	smtp
Outbound Username	username
Outbound Password	password

CREATE

VALIDATE

CANCEL

- 3 Crie um pipeline e adicione notificações para quando o pipeline for bem-sucedido ou falhar.

## Notification

**Send notification type**

☒ Email ☐ Ticket ☐ Webhook

**When pipeline**

☒ Completes ☐ Is Waiting ☐ Fails ☐ Is cancelled ☐ Starts to run

Email server ⓘ \*

--Select Email server-- ▾

Send Email

To ⓘ \$ \*

Email IDs of recipients

Subject \$ \*

Email Subject

Body ⓘ \$ \*

1

CANCEL

SAVE

#### 4 Adicione um estágio para o desenvolvimento e adicione uma tarefa de modelo em nuvem.

- a Adicione uma tarefa de modelo de nuvem que implante a máquina e configure-a para usar o YAML do modelo de nuvem para o aplicativo WordPress.

```
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 Adicione uma tarefa de modelo de nuvem que destrua a máquina para liberar recursos.

## 5 Adicione um estágio para produção e inclua tarefas de aprovação e implantação.

- a Adicione uma tarefa de operação do usuário para exigir aprovação para enviar o aplicativo WordPress para produção.
- b Adicione uma tarefa de modelo de nuvem para implantar a máquina e configure-a com o YAML de modelo de nuvem para o aplicativo WordPress.

Quando você seleciona **Criar**, o nome da implantação deve ser exclusiva. Se você deixar o nome em branco, o Code Stream atribuirá a ele um nome aleatório exclusivo.

Veja o que você deve saber se selecionar **Reverter** no seu próprio caso de uso: se você selecionar a ação **Reverter** e inserir uma **Versão de Reversão**, essa versão deverá estar no formato de **n-X**. Por exemplo, **n-1**, **n-2**, **n-3** e assim por diante. Se você criar e atualizar a implantação em qualquer local que não seja Code Stream, a reversão será permitida.

Quando você faz login no Code Stream, ele obtém um token de usuário, que é válido por 30 minutos. Para durações de pipeline de longa execução, quando a tarefa anterior à tarefa de modelo de nuvem demora 30 minutos ou mais para ser executada, o token de usuário expira. Como resultado, a tarefa de modelo de nuvem falha.

Para garantir que o pipeline possa ser executado por mais de 30 minutos, você pode inserir um token de API opcional. Quando o Code Stream invocar o modelo de nuvem, o token de API persistirá, e a tarefa de modelo de nuvem continuará a usar o token de API.

Quando você usa o token de API como uma variável, ele é criptografado. Caso contrário, ele será usado como texto simples.

The screenshot shows the configuration window for a task named 'Deploy CT'. The interface includes tabs for 'Task: Deploy CT', 'Notifications', and 'Rollback', with a 'VALIDATE TASK' button in the top right. The configuration fields are as follows:

- Task name:** Deploy CT
- Type:** VMware cloud template
- Continue on failure:** ☐
- Execute task:** ☒ Always ☐ On condition
- Deployment Task:**
  - Action:** ☒ Create ☐ Update ☐ Delete ☐ Rollback
  - API token:** API token (with a red 'x' icon and a 'CREATE VARIABLE' button)
  - Deployment Name:** Enter deployment name
  - Cloud template source:** ☒ VMware cloud templates ☐ Source Control
  - Cloud template:** --Select template--
  - Version:** --Select template Version--
- Output Parameters:** (Section header)



**6** Execute o pipeline.

Para examinar se cada tarefa foi concluída com êxito, clique na tarefa em execução e verifique o status nos detalhes da implantação para ver informações detalhadas sobre os recursos.

**7** No GitHub, modifique o tipo de instância de servidor do WordPress de `small` para `medium`.

Quando você confirmar as alterações, o pipeline será disparado. Ele extrairá o código atualizado do repositório do GitHub e compilará o aplicativo.

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

**8** Execute o pipeline novamente, verifique se foi bem-sucedido e se ele alterou o tipo de instância do WordPress de `pequena` para `média`.**Resultados**

Parabéns! Você automatizou o lançamento do aplicativo implantado a partir de um modelo de nuvem YAML.

**Próximo passo**

Para saber mais sobre como usar o Code Stream, consulte [Capítulo 5 Tutoriais para usar o Code Stream](#).

Para obter referências adicionais, consulte [Mais recursos para administradores e desenvolvedores do Code Stream](#).

## Como automatizar a liberação de um aplicativo no Code Stream para um cluster do Kubernetes

Como administrador ou desenvolvedor do Code Stream, você pode usar o Code Stream e o VMware Tanzu Kubernetes Grid Integrated Edition (anteriormente conhecido como VMware Enterprise PKS) para automatizar a implantação dos seus aplicativos de software em um cluster Kubernetes. Esse caso de uso menciona outros métodos possíveis para automatizar a liberação do aplicativo.

Neste caso de uso, você criará um pipeline que inclui dois estágios e usará o Jenkins para criar e implantar o aplicativo.

- O primeiro estágio é para desenvolvimento. Ele usa o Jenkins para extrair o código de uma ramificação no repositório GitHub, depois o compila, testa e publica.

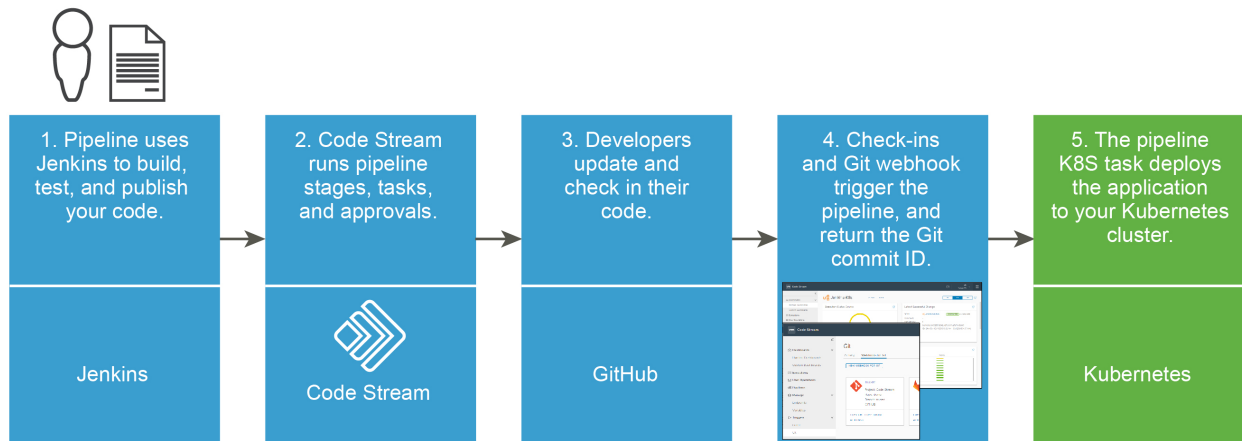
- O segundo estágio é para implantação. Ele executa uma tarefa de operação do usuário que requer a aprovação de usuários principais antes que o pipeline possa implantar o aplicativo no cluster do Kubernetes.

Ao usar um endpoint API do Kubernetes no espaço de trabalho do pipeline, o Code Stream cria os recursos do Kubernetes necessários, como ConfigMap, Segredo e Pod, para executar a tarefa de integração contínua (CI) ou personalizada. O Code Stream se comunica com o contêiner usando a NodePort.

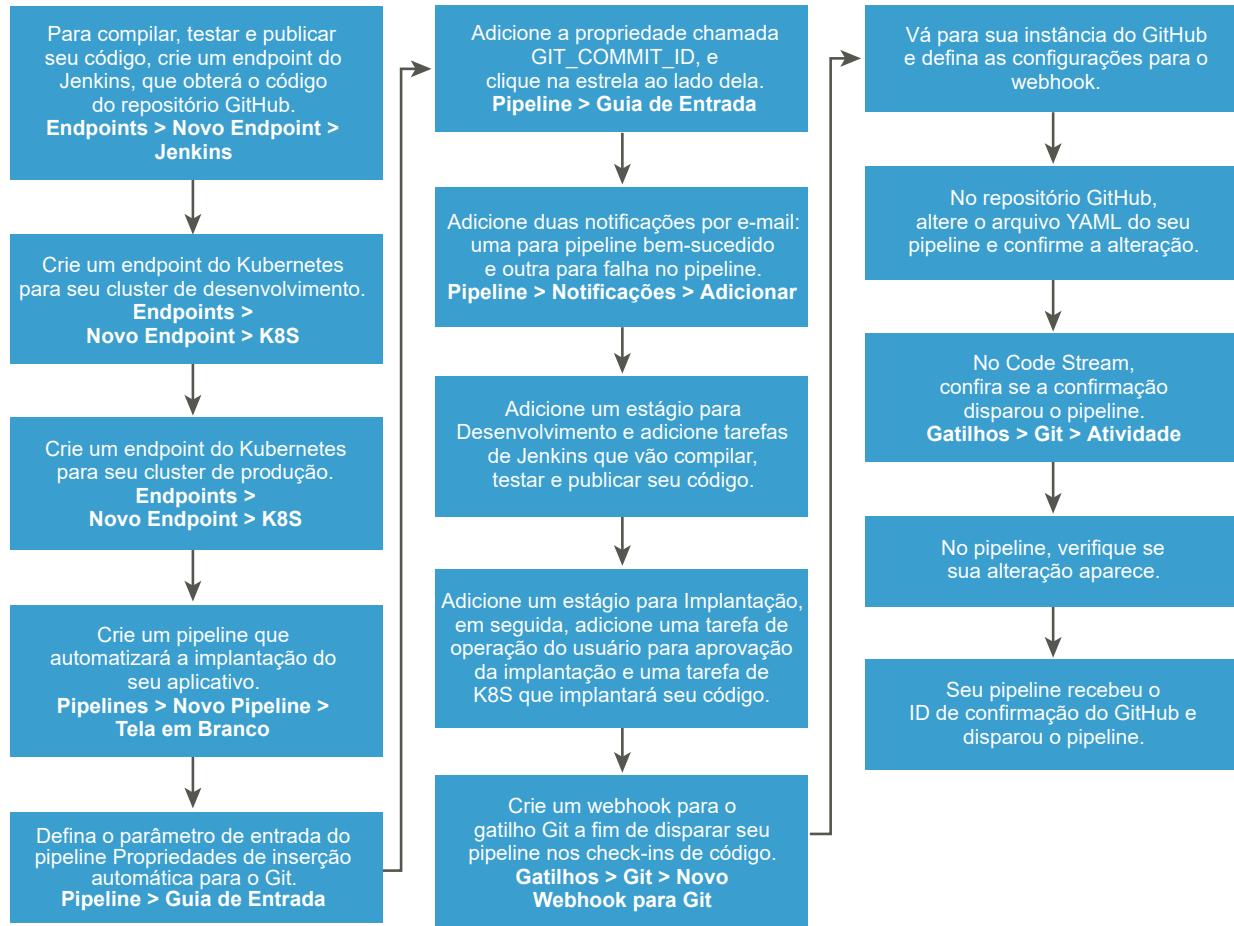
Para compartilhar os dados em todas as execuções do pipeline, você deve fornecer uma reivindicação de volume persistente, e o Code Stream montará a reivindicação de volume persistente no contêiner para armazenar os dados e a usará nas execuções seguintes do pipeline.

O espaço de trabalho do pipeline do Code Stream é compatível com o Docker e o Kubernetes para tarefas de integração contínua e personalizadas.

Para obter mais informações sobre a configuração do espaço de trabalho, consulte [Como configurar o espaço de trabalho do pipeline](#).



As ferramentas de desenvolvimento, instâncias de implantação e o arquivo YAML de pipeline devem estar disponíveis para que o pipeline possa compilar, testar, publicar e implantar o aplicativo. O pipeline implantará o aplicativo em instâncias de desenvolvimento e de produção dos clusters do Kubernetes no AWS.



Outros métodos que automatizam a liberação do aplicativo:

- Em vez de construir seu aplicativo usando o Jenkins, você pode usar a capacidade de compilação nativa do Code Stream e um host de compilação Docker.
- Em vez de implantar seu aplicativo em um cluster Kubernetes, você pode implantá-lo em um cluster Amazon Web Services (AWS).

Para obter mais informações sobre como usar o recurso de compilação nativa do Code Stream e um host do Docker, consulte:

- [Como planejar uma compilação nativa de CI/CD no Code Stream antes de usar o modelo de pipeline inteligente](#)
- [Planejando uma compilação nativa de CI/CD no Code Stream antes de adicionar tarefas manualmente](#)

#### Pré-requisitos

- Verifique se o código do aplicativo a ser implantado reside em um repositório GitHub em funcionamento.
- Verifique se há uma instância de trabalho Jenkins ativa.
- Verifique se há um servidor de e-mail ativo.

- No Code Stream, crie um endpoint de e-mail que se conecte ao servidor de e-mail.
- Configure dois clusters do Kubernetes no Amazon Web Services (AWS), para desenvolvimento e produção, onde o pipeline implantará o aplicativo.
- Verifique se o repositório GitHub contém o código YAML para o pipeline e, como alternativa, um arquivo YAML que defina os metadados e as especificações do ambiente.

## Procedimentos

- 1 No Code Stream, clique em **Endpoints > Novo Endpoint** e crie um endpoint do Jenkins que será usado no pipeline para extrair o código do seu repositório GitHub.

- 2 Para criar endpoints do Kubernetes, clique em **Novo Endpoint**.

- a Crie um endpoint para seu cluster de desenvolvimento do Kubernetes.
- b Crie um endpoint para seu cluster de produção do Kubernetes.

A URL do seu cluster Kubernetes pode ou não incluir um número de porta.

Por exemplo:

```
https://10.111.222.333:6443
```

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

- 3 Crie um pipeline que implante um contêiner do aplicativo, como o WordPress, no cluster de desenvolvimento do Kubernetes e defina as propriedades de entrada para o pipeline.

- a Para permitir que o seu pipeline reconheça uma confirmação de código no GitHub que disparará o pipeline, clique na guia **Entrada**, no próprio pipeline, e selecione **Propriedades de Inserção Automática**.

- b Adicione a propriedade chamada **GIT\_COMMIT\_ID** e clique na estrela ao lado dela.

Quando o pipeline for executado, sua execução exibirá a ID de confirmação retornada pelo gatilho Git.

**Jenkins-K8s** Enabled

Pipeline **Input** Output Notifications CI Workspace

Pipeline Input Parameters

Auto inject properties ☐ Gerrit ☒ Git ☐ None

**ADD**

Starred	Name	Value	Description
<input type="checkbox"/>	GIT_BRANCH_NAME		
<input type="checkbox"/>	GIT_CHANGE_SUBJECT		
<input checked="" type="checkbox"/>	GIT_COMMIT_ID		
<input type="checkbox"/>	GIT_EVENT_DESCRIPTION		
<input type="checkbox"/>	GIT_EVENT_OWNER_NAME		
<input type="checkbox"/>	GIT_EVENT_TIMESTAMP		
<input type="checkbox"/>	GIT_REPO_NAME		
<input type="checkbox"/>	GIT_SERVER_URL		

8 input parameters

**SAVE** **RUN** **CLOSE** Last saved 5 days ago

- 4 Adicione notificações para enviar um e-mail quando o pipeline for bem-sucedido ou falhar.
  - a No pipeline, clique na guia **Notificações** e clique em **Adicionar**.
  - b Para adicionar uma notificação por e-mail quando o pipeline terminar de ser executado, selecione **E-mail** e depois **Completa**. Em seguida, selecione o servidor de e-mail, digite os endereços de e-mail e clique em **Salvar**.
  - c Para adicionar outra notificação por e-mail para uma falha de pipeline, selecione **Falhar** e clique em **Salvar**.

### Notification

**Send notification type**

☒ Email
 ☐ Ticket
 ☐ Webhook

**When pipeline**

☒ Completes
 ☐ Is Waiting
 ☐ Fails
 ☐ Is cancelled
 ☐ Starts to run

Email server ⓘ \*

--Select Email server-- ▾

Send Email

To ⓘ \$ \*

Email IDs of recipients

Subject \$ \*

Email Subject

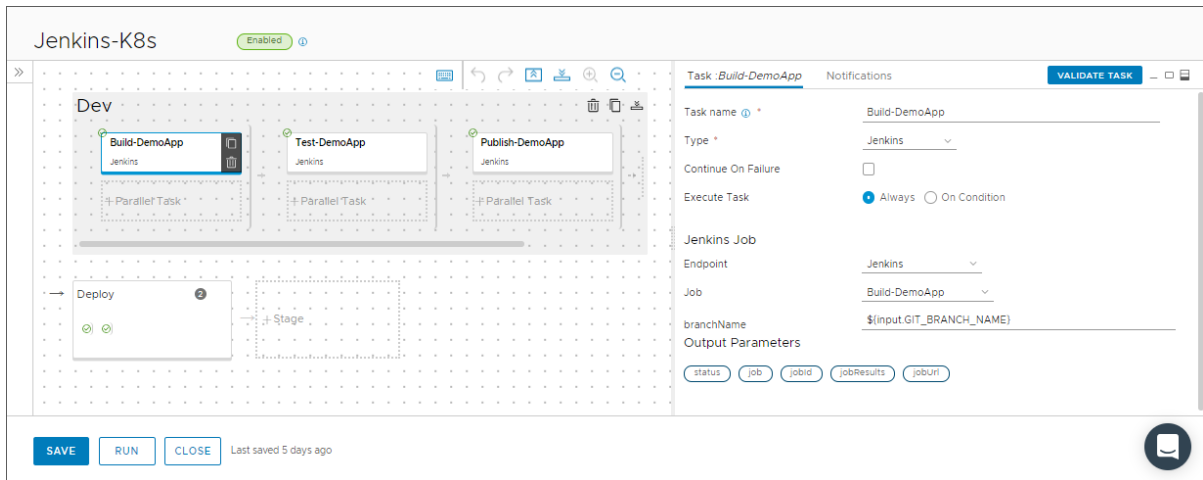
Body ⓘ \$ \*

1

CANCEL

SAVE

- 5 Adicione um estágio de desenvolvimento ao pipeline e tarefas que compilem, testem e publiquem seu aplicativo. Em seguida, valide cada tarefa.
  - a Para compilar seu aplicativo, adicione uma tarefa de Jenkins que use o endpoint do Jenkins e execute um trabalho de compilação a partir do servidor Jenkins. Depois, para que o pipeline receba seu código, insira a ramificação Git desta forma: `${input.GIT_BRANCH_NAME}`
  - b Para testar seu aplicativo, adicione uma tarefa de Jenkins que use o mesmo endpoint do Jenkins e execute um trabalho de teste no servidor Jenkins. Em seguida, insira a mesma ramificação Git.
  - c Para publicar seu aplicativo, adicione uma tarefa de Jenkins que use o mesmo endpoint do Jenkins e execute um trabalho de publicação no servidor Jenkins. Em seguida, insira a mesma ramificação Git.



- 6 Adicione um estágio de implantação ao seu pipeline e, depois, adicione uma tarefa que exija uma aprovação para a implantação do aplicativo, e outra tarefa que implante o aplicativo no cluster do Kubernetes. Em seguida, valide cada tarefa.
  - a Para exigir uma aprovação na implantação do seu aplicativo, adicione uma tarefa de Operação do Usuário, adicione os endereços de e-mail para os usuários que devem aprová-la e digite uma mensagem. Em seguida, ative **Enviar E-mail**.
  - b Para implantar o aplicativo, adicione uma tarefa do Kubernetes. Depois, nas propriedades da tarefa do Kubernetes, selecione seu cluster de desenvolvimento do Kubernetes, selecione a ação **Criar** e selecione a fonte de payload **Definição de Local**. Em seguida, selecione seu arquivo YAML local.

- 7 Adicione um webhook do Git para permitir que o Code Stream use o gatilho Git, que dispara seu pipeline quando os desenvolvedores confirmam o código.

- 8 Para testar o pipeline, acesse o repositório GitHub, atualize o arquivo YAML do aplicativo e confirme a alteração.
  - a No Code Stream, verifique se a confirmação é exibida.
  - a Clique em **Gatilhos > Git > Atividade**.
  - b Procure o gatilho do pipeline.
  - c Clique em **Painéis > Painéis de Pipeline**.
  - d No painel de pipeline, encontre o GIT\_COMMIT\_ID na última área de alteração bem-sucedida.
- 9 Verifique o código de pipeline e verifique se a alteração é exibida.

## Resultados

Parabéns! Você automatizou a implantação do seu aplicativo de software no cluster do Kubernetes.

## Exemplo: Exemplo de YAML de pipeline que implanta um aplicativo em um cluster do Kubernetes

Para o tipo de pipeline usado nesse exemplo, o YAML deve ser semelhante ao seguinte código:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
eyJzeWlwaG9ueS10YW5nbY1iZXRhMi5qZnJvZy5pbyI6eyJlc2VybmFtZSI6InRhbmRvLWJldGEyIiwicGFzc3dvcmQI Oi
JhRGstcmVOLWlUQil1IejciLCJlbWFnYyI6InRhbmRvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoizEdGdVoyOHRZbVYw
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

```

### Próximo passo

Para implantar o aplicativo de software no cluster de produção do Kubernetes, execute as etapas novamente e selecione o cluster de produção.

Para saber mais sobre a integração do Code Stream com o Jenkins, consulte [Como integrar o Code Stream ao Jenkins](#).

## Como implantar meu aplicativo no Code Stream na implantação Azul-Verde

Azul-Verde é um modelo de implantação que usa dois hosts do Docker, implantados e configurados de forma idêntica em um cluster do Kubernetes. Com o modelo de implantação Blue-Green, reduz-se o tempo de inatividade que pode ocorrer no ambiente quando os pipelines no Code Stream implantam os aplicativos.

As instâncias azul e verde no modelo de implantação têm finalidades diferentes. Somente uma instância por vez aceita o tráfego dinâmico que implanta seu aplicativo, e cada instância aceita esse tráfego em momentos específicos. A instância azul recebe a primeira versão do aplicativo, e a instância verde recebe a segunda.

O balanceador de carga no ambiente Azul-Verde determina qual rota o tráfego dinâmico assumirá ao implantar o aplicativo. Ao usar o modelo Azul-Verde, seu ambiente permanece em operação, os usuários não percebem nenhum tempo de inatividade, o pipeline faz a integração de forma contínua e implanta o aplicativo no ambiente de produção.

O pipeline criado no Code Stream representa o modelo de implantação Azul-Verde em dois estágios. Um estágio é para desenvolvimento, e o outro estágio é para produção.

O espaço de trabalho do pipeline do Code Stream é compatível com o Docker e o Kubernetes para tarefas de integração contínua e personalizadas.

Para obter informações sobre a configuração do espaço de trabalho, consulte [Como configurar o espaço de trabalho do pipeline](#).

**Tabela 5-2. Tarefas do estágio de desenvolvimento para implantação Azul-Verde**

Tipo de tarefa	Tarefa
Kubernetes	Cria um namespace para sua implantação Azul-Verde.
Kubernetes	Cria uma chave secreta para o Hub do Docker.
Kubernetes	Cria o serviço usado para implantar o aplicativo.
Kubernetes	Cria a implantação azul.
Pesquisa	Verifica a implantação azul.
Kubernetes	Remove o namespace.

**Tabela 5-3. Tarefas do estágio de produção para implantação Azul-Verde**

Tipo de tarefa	Tarefa
Kubernetes	O verde obtém os detalhes de serviço do azul.
Kubernetes	Obtém os detalhes para o conjunto de réplicas verde.
Kubernetes	Cria a implantação verde e use a chave secreta para receber a imagem do contêiner.
Kubernetes	Atualiza o serviço.
Pesquisa	Verifica se a implantação foi bem-sucedida no URL de produção.
Kubernetes	Conclui a implantação azul.
Kubernetes	Remove a implantação azul.

Para implantar o aplicativo no seu próprio modelo de implantação Azul-Verde, crie um pipeline no Code Stream com dois estágios. O primeiro estágio inclui as tarefas azuis que implantam o aplicativo na instância azul, e o segundo estágio inclui tarefas verdes que implantam o aplicativo na instância verde.

É possível criar seu pipeline usando o modelo de pipeline inteligente de CI/CD. O modelo cria os estágios e as tarefas do pipeline, além de incluir as seleções de implantação.

Se você criar o pipeline manualmente, deverá planejar os estágios do pipeline. Para obter um exemplo, consulte [Planejando uma compilação nativa de CI/CD no Code Stream antes de adicionar tarefas manualmente](#).

Neste exemplo, você usa o modelo de pipeline inteligente de CI/CD para criar o pipeline Azul-Verde.

#### Pré-requisitos

- Verifique se é possível acessar um cluster do Kubernetes ativo no AWS.

- Verifique se há um ambiente de implantação Azul-Verde configurado e se há instâncias azul e verde configuradas de forma idêntica.
- Crie um endpoint do Kubernetes no Code Stream que implanta a imagem do aplicativo no cluster do Kubernetes no AWS.
- Familiarize-se usando o modelo de pipeline inteligente de CI/CD. Consulte [Como planejar uma compilação nativa de CI/CD no Code Stream antes de usar o modelo de pipeline inteligente](#).

#### Procedimentos

- 1 Clique em **Pipelines > Novo Pipeline > Modelos Inteligentes > Modelo de CI/CD**.
- 2 Digite as informações para a parte de CI do modelo de pipeline inteligente de CI/CD e clique em **Seguinte**.

Para obter ajuda, consulte [Como planejar uma compilação nativa de CI/CD no Code Stream antes de usar o modelo de pipeline inteligente](#).

- 3 Conclua a parte sobre CD do modelo de pipeline inteligente
  - a Selecione os ambientes para a implantação do aplicativo. Por exemplo, **Des.** e **Prod.**
  - b Selecione o serviço que o pipeline usará para a implantação.
  - c Na área Implantação, selecione o endpoint de cluster para o ambiente Dev e o ambiente Prod.
  - d Para o modelo de implantação de produção, selecione **Azul-Verde** e clique em **Criar**.

**Smart Template: CI/CD**

*Step 2 of 2*

Environment ⓘ \* ☒ Dev ☒ Prod

K8s YAML files \*

Processed files: codestream.yaml

Select service

Deployment name	Service	Namespace	Image
codestream-demo	codestream-demo	codestream	https://codestream/Myapp

1 services

Deployment

Environment	Cluster Endpoint	Namespace
Dev	Dev-AWS-Cluster	codestream-139606
Prod	Prod-AWS-Cluster	codestream

Prod deployment model \* ☐ Canary ☐ Rolling Upgrade ☒ Blue-Green

Rollback strategy ☐

Health check URL \*

## Resultados

Parabéns! O modelo de pipeline inteligente foi usado para criar um pipeline que implanta o aplicativo em suas instâncias Azul-Verde no cluster de produção do Kubernetes na AWS.

## Exemplo: Exemplo de código de YAML para algumas tarefas de implantação Azul-Verde

O código YAML que aparece nas tarefas de pipeline do Kubernetes para sua implantação Azul-Verde pode ser semelhante aos exemplos a seguir, que criam Namespace, Serviço e Implantação. Se você precisar baixar uma imagem de um repositório de propriedade privada, o arquivo YAML deverá incluir uma seção com o Segredo de configuração do Docker. Consulte a parte sobre CD de [Como planejar uma compilação nativa de CI/CD no Code Stream antes de usar o modelo de pipeline inteligente](#).

Depois que o modelo de pipeline inteligente cria o pipeline, é possível modificar as tarefas conforme necessário para sua própria implantação.

Código do YAML para criar um exemplo de namespace:

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

Código YAML para criar um exemplo de serviço:

```
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
```

Código YAML para criar um exemplo de implantação:

```
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
```

### Próximo passo

Para saber mais sobre como usar o Code Stream, consulte [Capítulo 5 Tutoriais para usar o Code Stream](#).

Para reverter uma implantação, consulte [Como faço para reverter minha implantação no Code Stream](#).

Para obter referências adicionais, consulte [Mais recursos para administradores e desenvolvedores do Code Stream](#).

## Como integrar minhas próprias ferramentas de compilação, teste e implantação com o Code Stream

Como administrador ou desenvolvedor de DevOps, é possível criar scripts personalizados que estendem o recurso do Code Stream.

Com o script, é possível integrar o Code Stream às suas próprias ferramentas de integração contínua (IC) e entrega contínua (EC) e APIs que compilam, testam e implantam os aplicativos. Os scripts personalizados são especialmente úteis se as APIs do aplicativo não forem expostas publicamente.

O script personalizado pode fazer quase tudo o que é necessário para que as ferramentas de compilação, teste e implantação sejam integradas ao Code Stream. Por exemplo, ele pode trabalhar com o espaço de trabalho do pipeline para permitir tarefas de integração contínua que compilam e testam o aplicativo e tarefas de entrega contínua que implantam o aplicativo. Ele pode enviar uma mensagem para o Slack quando um pipeline é concluído e muito mais.

O espaço de trabalho do pipeline do Code Stream é compatível com o Docker e o Kubernetes para tarefas de integração contínua e personalizadas.

Para obter mais informações sobre a configuração do espaço de trabalho, consulte [Como configurar o espaço de trabalho do pipeline](#).

Escreva o script personalizado em um dos idiomas compatíveis. No script, inclua a lógica de negócios e defina entradas e saídas. Os tipos de saída podem incluir número, cadeia de caracteres, texto e senha. Você pode criar várias versões de um script personalizado com diferentes lógicas de negócios, entradas e saídas.

Você instrui o pipeline a executar uma versão do seu script em uma tarefa personalizada. Os scripts criados residem na instância do Code Stream.

Quando um pipeline usar uma integração personalizada, se você tentar excluir a integração personalizada, uma mensagem de erro será exibida e indicará que não é possível excluí-la.

A exclusão de uma integração personalizada remove todas as versões do seu script personalizado. Se você tiver um pipeline existente com uma tarefa personalizada que usa qualquer versão do script, esse pipeline falhará. Para garantir que os pipelines existentes não falhem, você poderá reprovar e retirar a versão do script que não deseja mais usar. Se nenhum pipeline estiver usando essa versão, você poderá excluí-la.

**Tabela 5-4. O que fazer após escrever o script personalizado**

O que fazer...	Mais informações sobre esta ação...
Adicione uma tarefa personalizada ao pipeline.	<p>A tarefa personalizada:</p> <ul style="list-style-type: none"> <li>■ É executada no mesmo contêiner de outras tarefas de IC no pipeline.</li> <li>■ Inclui as variáveis de entrada e saída que o script preenche antes que o pipeline execute a tarefa personalizada.</li> <li>■ Oferece suporte a vários tipos de dados e vários tipos de metadados definidos como entradas e saídas no script.</li> </ul>
Selecione o script na tarefa personalizada.	Declare as propriedades de entrada e saída no script.
Salve seu pipeline e, em seguida, ative-o e execute-o.	Quando o pipeline for executado, a tarefa personalizada chamará a versão do script especificada e executará a lógica de negócios nele, que integra sua ferramenta de compilação, teste e implantação ao Code Stream.
Após a execução do pipeline, observe as execuções.	Verifique se o pipeline gerou os resultados esperados.

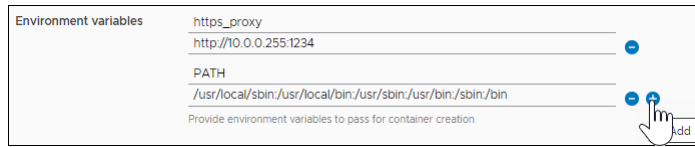
Quando você usa uma tarefa personalizada que chama uma versão de integração personalizada, é possível incluir variáveis de ambiente personalizadas como pares de nome/valor na guia **Espaço de Trabalho** do pipeline. Quando a imagem do construtor cria o contêiner de espaço de trabalho que executa a tarefa de CI e implanta a sua imagem, o Code Stream transmite as variáveis de ambiente para esse contêiner.

Por exemplo, quando a sua instância do Code Stream requer um proxy Web e você usa um host do Docker para criar um contêiner para uma integração personalizada, o Code Stream executa o pipeline e transmite as variáveis de configuração do proxy Web para esse contêiner.

**Tabela 5-5. Exemplo de pares de nome/valor de variáveis de ambiente**

Nome	Valor
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

Os pares de nome/valor aparecem na interface do usuário da seguinte maneira:



Este exemplo cria uma integração personalizada que conecta o Code Stream à instância do Slack e publica uma mensagem em um canal do Slack.

### Pré-requisitos

- Para escrever seu script personalizado, verifique se você tem uma destas linguagens: Python 2, Python 3, Node.js ou qualquer uma das linguagens de shell: Bash, sh ou zsh.
- Gere uma imagem de contêiner usando o Node.js instalado ou o tempo de execução do Python.

### Procedimentos

#### 1 Crie a integração personalizada.

- a Clique em **Integrações Personalizadas > Novo** e insira um nome relevante.
- b Selecione o ambiente de tempo de execução preferencial.
- c Clique em **Criar**.

Seu script é aberto e exibe o código, que inclui o ambiente de tempo de execução necessário. Por exemplo, `runtime: "nodejs"`. O script deve incluir o tempo de execução, utilizado pela imagem do compilador, para que a tarefa personalizada adicionada ao pipeline seja bem-sucedida quando o pipeline for executado. Caso contrário, a tarefa personalizada falhará.

As principais áreas do YAML de integração personalizada incluem o tempo de execução, código, propriedades de entrada e propriedades de saída. Esse procedimento explica vários tipos e sintaxes.

Chaves de YAML de integração personalizada	Descrição
tempo de execução	<p>Ambiente do tempo de execução da tarefa em que o Code Stream executa o código, que pode ser uma dessas cadeias de caracteres que não diferenciam maiúsculas de minúsculas:</p> <ul style="list-style-type: none"> <li>■ nodejs</li> <li>■ python2</li> <li>■ python3</li> <li>■ shell</li> </ul> <p>Se nada for fornecido, o shell será o padrão presumido.</p>
código	Lógica de negócios personalizada a ser executada como parte da tarefa personalizada.



Chaves de YAML de integração personalizada	
Chaves de YAML de integração personalizada	Descrição
<code>inputProperties</code>	Matriz de propriedades de entrada a serem capturadas como parte da configuração da tarefa personalizada. Essas propriedades normalmente são usadas no código.
<code>outputProperties</code>	Matriz de propriedades de saída que você pode exportar da tarefa personalizada para propagar ao pipeline.

- 2 Declare as propriedades de entrada no script usando os tipos de dados e metadados disponíveis.

As propriedades de entrada são transmitidas como contexto ao script na seção `code:` do YAML.

Chaves de entrada YAML da tarefa personalizada		
Chaves de entrada YAML da tarefa personalizada	Descrição	Obrigatório
<code>type</code>	Tipos de entrada para renderização: <ul style="list-style-type: none"> <li>■ <code>text</code></li> <li>■ <code>textarea</code></li> <li>■ <code>number</code></li> <li>■ <code>checkbox</code></li> <li>■ <code>password</code></li> <li>■ <code>select</code></li> </ul>	Sim
<code>name</code>	Nome ou cadeia de caracteres da entrada à tarefa personalizada, que é injetada no código YAML de integração personalizada. Deve ser exclusivo para cada propriedade de entrada definida para uma integração personalizada.	Sim
<code>title</code>	Rótulo da cadeia de caracteres de texto da propriedade de entrada à tarefa personalizada na tela do modelo de pipeline. Se deixado em branco, o <code>name</code> será usado por padrão.	Não
<code>required</code>	Determina se um usuário deve inserir a propriedade de entrada ao configurar a tarefa personalizada. Defina como verdadeiro ou falso. Quando verdadeiro, se um usuário não fornecer um valor ao configurar a tarefa customizada na tela do pipeline, o estado da tarefa permanecerá como não configurado.	Não
<code>placeholder</code>	Texto padrão na área de entrada da propriedade de entrada quando nenhum valor está presente. Mapeia para o atributo do marcador de posição HTML. Compatível apenas com certos tipos de propriedades de entrada.	Não
<code>defaultValue</code>	Valor padrão que preenche a área de entrada da propriedade de entrada quando a tarefa personalizada é renderizada na página do modelo de pipeline.	Não
<code>bindable</code>	Determina se a propriedade de entrada aceita variáveis de cifrão ao modelar a tarefa personalizada na tela do pipeline. Adiciona o indicador <code>\$</code> ao lado do título. Compatível apenas com certos tipos de propriedades de entrada.	Não

Chaves de entrada YAML da tarefa personalizada		
YAML da tarefa personalizada	Descrição	Obrigatório
<b>labelMessage</b>	Cadeia de caracteres que funciona como uma dica de ajuda aos usuários. Adiciona um ícone de dica de ferramenta i próximo ao título de entrada.	Não
<b>enum</b>	<p>Recebe uma matriz de valores que exibe as opções para selecionar propriedades de entrada. Compatível apenas com certos tipos de propriedades de entrada.</p> <p>Quando um usuário seleciona uma opção e a salva para a tarefa personalizada, o valor de <b>inputProperty</b> corresponde a esse valor e aparece na modelagem da tarefa personalizada.</p> <p>Por exemplo, o valor 2015.</p> <ul style="list-style-type: none"> <li>■ 2015</li> <li>■ 2016</li> <li>■ 2017</li> <li>■ 2018</li> <li>■ 2019</li> <li>■ 2020</li> </ul>	Não
<b>options</b>	<p>Recebe uma matriz de objetos usando <b>optionKey</b> e <b>optionValue</b>.</p> <ul style="list-style-type: none"> <li>■ <b>optionKey</b>. Valor propagado para a seção de código da tarefa.</li> <li>■ <b>optionValue</b>. Cadeia de caracteres que exibe a opção na interface do usuário.</li> </ul> <p>Compatível apenas com certos tipos de propriedades de entrada.</p> <p>Options:</p> <p><b>optionKey</b>: key1. Quando selecionado e salvo para a tarefa personalizada, o valor deste inputProperty corresponde a <b>key1</b> na seção de código.</p> <p><b>optionValue</b>: "Rótulo para 1". Exibe o valor de <b>key1</b> na interface do usuário e não aparece em nenhum outro lugar da tarefa personalizada.</p> <p><b>optionKey</b>: key2</p> <p><b>optionValue</b>: "Rótulo para 2"</p> <p><b>optionKey</b>: key3</p> <p><b>optionValue</b>: "Rótulo para 3"</p>	Não
<b>minimum</b>	Recebe um número que atua como o valor mínimo válido para essa propriedade de entrada. Compatível apenas com a propriedade de entrada do tipo de número.	Não
<b>maximum</b>	Recebe um número que atua como o valor máximo válido para essa propriedade de entrada. Compatível apenas com a propriedade de entrada do tipo de número.	Não

Tabela 5-6. Tipos de dados e metadados de dados compatíveis para scripts personalizados

Tipos de dados compatíveis	Metadados compatíveis para entrada
<ul style="list-style-type: none"> <li>■ Cadeia de caracteres</li> <li>■ Texto</li> <li>■ Lista: como uma lista de qualquer tipo</li> <li>■ Mapa: como um mapa [string] qualquer</li> <li>■ Seguro: apresentado como uma caixa de texto de senha, criptografado quando você salva a tarefa personalizada</li> <li>■ Número</li> <li>■ Boolean: aparece como caixas de texto</li> <li>■ URL: igual à cadeia de caracteres, com validação adicional</li> <li>■ Botão de seleção, opção</li> </ul>	<ul style="list-style-type: none"> <li>■ tipo: uma da cadeia de caracteres   Texto...</li> <li>■ padrão: valor padrão</li> <li>■ opções: lista ou um mapa de opções, a ser usado com botão de seleção ou opção</li> <li>■ mínimo: valor ou tamanho mínimo</li> <li>■ máximo: valor ou tamanho máximo</li> <li>■ título: nome detalhado da caixa de texto</li> <li>■ marcador de posição: marcador de posição da IU</li> <li>■ descrição: se torna uma dica de ferramenta</li> </ul>

Por exemplo:

```
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 Declare as propriedades de saída no script.

O script captura as propriedades de saída da seção de lógica de negócios `code`: do script, na qual você declara o contexto para a saída.

Quando o pipeline é executado, é possível inserir o código de resposta para a saída da tarefa. Por exemplo, **200**.

Chaves que o Code Stream suporta para cada **outputProperty**.

chave	Descrição
type	Atualmente inclui um valor único de <b>label</b> .
name	Chave que o bloco de código do YAML de integração personalizada emite.
title	Rótulo na interface do usuário que exibe <b>outputProperty</b> .

Por exemplo:

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

- 4 Para interagir com a entrada e saída do script personalizado, obtenha uma propriedade de entrada ou defina uma propriedade de saída usando **context**.

Para uma propriedade de entrada: `(context.getInput("key"))`

Para uma propriedade de saída: `(context.setOutput("key", "value"))`

Para Node.js:

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

Para Python:

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

Para Shell:

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

- 5 Na seção `code:`, declare toda a lógica de negócios para sua integração personalizada.

Por exemplo, com o ambiente de tempo de execução Node.js:

```
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)
  }
};

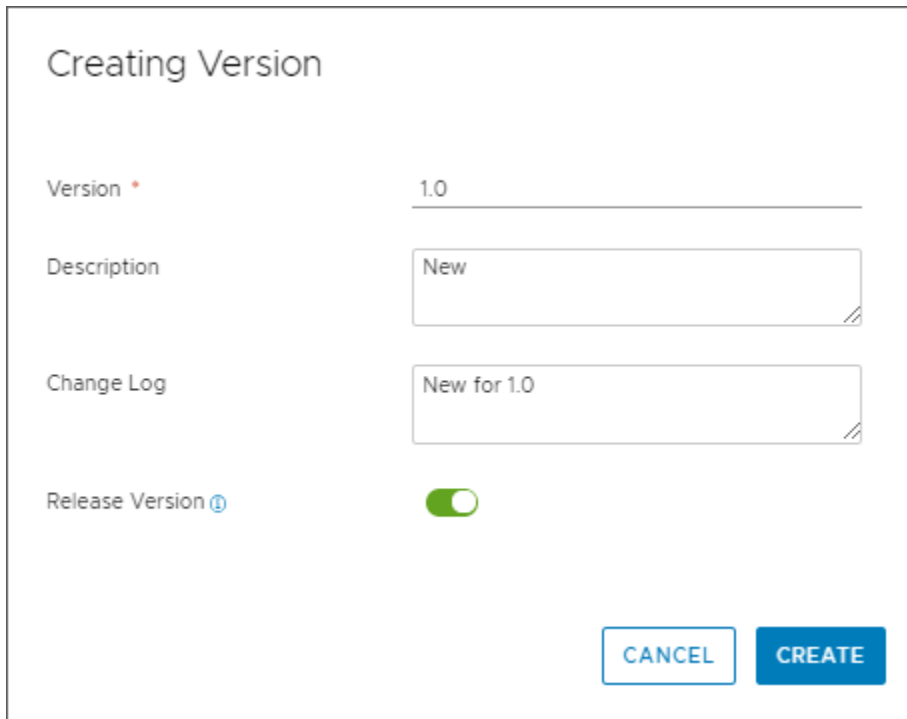
// Makes a https request and sets the output with statusCode which
```

```
// 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 Antes que você faça a versão e liberação de script de integração personalizado, faça o download do arquivo de contexto para Python ou Node.js e teste a lógica de negócios incluída no script.
  - a Coloque o ponteiro no script e, em seguida, clique no botão do arquivo de contexto na parte superior da tela. Por exemplo, se o script estiver em Python, clique em **CONTEXT.PY**.
  - b Modifique o arquivo e salve-o.
  - c No sistema de desenvolvimento, execute e teste o script personalizado com a ajuda do arquivo de contexto.
- 7 Aplique uma versão ao script de integração personalizado.
  - a Clique em **Versão**.
  - b Insira as informações de versão.

- c Clique em **Versão de Liberação** para poder selecionar o script na tarefa personalizada.
- d Para criar a versão, clique em **Criar**.



The image shows a 'Creating Version' dialog box. It has a title bar at the top. Below the title, there are four input fields: 'Version' with a red asterisk, 'Description', 'Change Log', and 'Release Version' with a blue information icon. The 'Version' field contains '1.0'. The 'Description' field contains 'New'. The 'Change Log' field contains 'New for 1.0'. The 'Release Version' field has a green toggle switch turned on. At the bottom right, there are two buttons: 'CANCEL' and 'CREATE'.

Creating Version

Version \* 1.0

Description New

Change Log New for 1.0

Release Version ⓘ ☒

CANCEL CREATE

- 8 Para salvar o script, clique em **Salvar**.
- 9 No pipeline, configure o espaço de trabalho.  
Neste exemplo, é usado um espaço de trabalho do Docker.

- a Clique na guia **Espaço de Trabalho**.
- b Selecione o host do Docker e o URL da imagem do compilador.

The screenshot shows the configuration interface for a pipeline named "Demo-customTask-nodejs", which is in an "Enabled" state. The interface has four tabs: "Workspace" (selected), "Input", "Model", and "Output". Under the "Workspace" tab, there are several configuration fields:

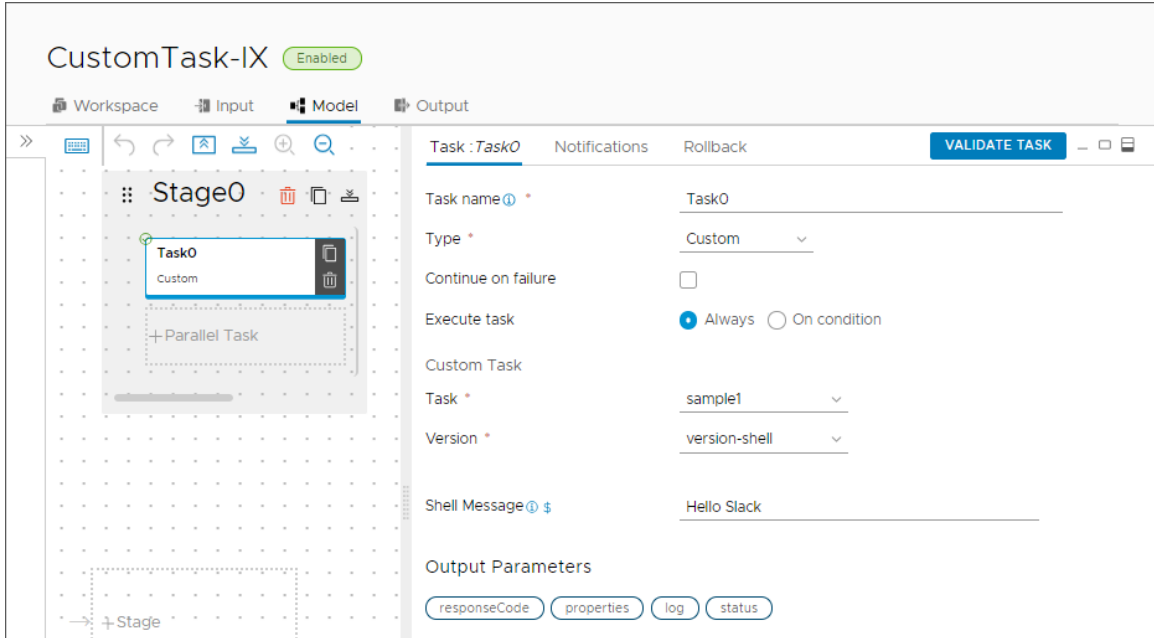
- Host**: A dropdown menu currently showing "Docker-saas".
- Builder image URL**: A text input field containing "node:latest".
- Image registry**: A dropdown menu showing "--Select Container Registry Endpoint--".
- Working directory**: An empty text input field.
- Cache**: An empty text input field with a blue plus icon to its right.
- Git clone**: A checkbox that is currently unchecked.

At the bottom of the configuration area, there is a note: "If this pipeline links to Git through a webhook, the pipeline triggers on Git events. For CI tasks, the linked Git repository, which receives details from the Git webhook, automatically clones the workspace."

- 10 Adicione uma tarefa personalizada ao pipeline e configure-a.
  - a Clique na guia **Modelo**.
  - b Adicione uma tarefa, selecione o tipo como **Personalizado** e digite um nome relevante.

- c Selecione a versão e o script da integração personalizada.
- d Para exibir uma mensagem personalizada no Slack, digite o texto da mensagem.

Qualquer texto digitado substituirá o `defaultValue` no script de integração personalizado.  
Por exemplo:



- 11 Salve e ative o pipeline.
  - a Clique em **Salvar**.
  - b Na guia Pipeline, clique em **Ativar Pipeline** para que o círculo se mova para a direita.
- 12 Execute o pipeline.
  - a Clique em **Executar**.
  - b Observe a execução do pipeline.



- c Confirme se a saída inclui o código de status esperado, o código de resposta, o status e a saída declarada.

Você definiu **statusCode** como uma propriedade de saída. Por exemplo, um **statusCode** de 200 pode indicar uma publicação de Slack bem-sucedida e um **responseCode** de 0 pode indicar que o script foi bem-sucedido, sem erros.

- d Para confirmar a saída nos logs de execução, clique em **Execuções**, clique no link do pipeline, clique na tarefa e examine os dados registrados. Por exemplo:

The screenshot displays the vRealize Automation interface for a pipeline named 'custom-int-demo #5'. The pipeline is in a 'COMPLETED' state. Below the pipeline name, a progress bar shows 'Stage0' completed, with 'Task0' and 'Task1' also marked as completed. The main section shows the details for 'Task1':

Task name	Task1 <a href="#">VIEW OUTPUT JSON</a>
Type	Custom
Status	<b>COMPLETED</b> Execution Completed.
Duration	6s (12/21/2018 3:04 AM - 12/21/2018 3:04 AM)
Continue on failure	<input type="checkbox"/>
Execute task	<input checked="" type="radio"/> Always <input type="radio"/> On condition
Output	
statusCode	200
Response code	0
Logs	<pre> 1 + node -r ./context.js app.js 2 3   </pre>

At the bottom of the logs section, there is a link to [View Full Log](#).

**13** Se ocorrer um erro, solucione o problema e execute o pipeline novamente.

Por exemplo, se um arquivo ou módulo na imagem de base estiver ausente, será necessário criar outra imagem de base que inclua o arquivo ausente. Em seguida, forneça o arquivo Docker e envie a imagem por meio do pipeline.

### Resultados

Parabéns! Você criou um script de integração personalizado que conecta o Code Stream à instância de Slack e publica uma mensagem em um canal de Slack.

### Próximo passo

Continue a criar integrações personalizadas para oferecer suporte ao uso de tarefas personalizadas nos pipelines, para poder ampliar o recurso do Code Stream na automação do ciclo de liberação do software.

## Como usar as propriedades de recursos de uma tarefa de modelo de nuvem na minha próxima tarefa

Quando você usa uma tarefa de modelo de nuvem no Code Stream, uma pergunta comum é como usar a saída dessa tarefa em uma tarefa subsequente no pipeline. Para usar a saída de uma tarefa de modelo de nuvem, como uma máquina de nuvem, você deve saber como encontrar as propriedades de recursos nos detalhes da implantação da tarefa de modelo de nuvem e o endereço IP da máquina de nuvem.

Por exemplo, os detalhes de implantação do Modelo de Nuvem VMware incluem o recurso de máquina de nuvem e seu endereço IP. No pipeline, você pode usar a máquina de nuvem e o endereço IP como uma variável para associar uma tarefa de modelo de nuvem a uma tarefa REST.

O método usado para encontrar o endereço IP para a máquina de nuvem não é típico, pois a implantação do Modelo de Nuvem VMware deve ser concluída antes que os detalhes da implantação estejam disponíveis. Em seguida, você pode usar os recursos da implantação do Modelo de Nuvem VMware para associar suas tarefas de pipeline.

- As propriedades de recursos que aparecem em uma tarefa de modelo de nuvem no pipeline são definidas no Modelo de Nuvem VMware no Cloud Assembly.
- Você pode não saber quando uma implantação desse modelo de nuvem foi concluída.
- Uma tarefa de modelo de nuvem no Code Stream só poderá exibir as propriedades de saída do Modelo de Nuvem VMware após a conclusão da implantação.

Esse exemplo pode ser especialmente útil quando você está implantando um aplicativo e chamando várias APIs. Por exemplo, se você usar uma tarefa de modelo de nuvem que chama um Modelo de Nuvem VMware, que implanta um aplicativo WordPress com uma REST API, poderá localizar o endereço IP da máquina implantada nos detalhes da implantação e usar a API para testá-lo.

A tarefa de modelo de nuvem oferece suporte ao uso da associação de variáveis, exibindo os detalhes de preenchimento automático de tipo antecipado. Cabe a você escolher a associação da variável.

Este exemplo mostra como:

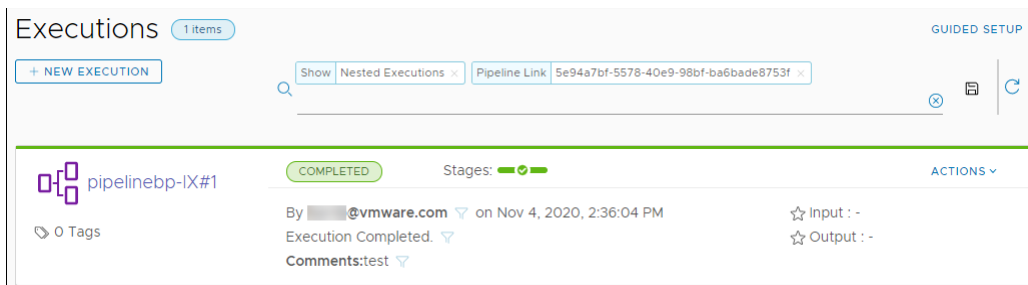
- Encontrar os detalhes de implantação e as propriedades de recursos para a tarefa de modelo de nuvem em um pipeline que foi executado e teve êxito.
- Encontrar o endereço IP da máquina de nuvem na seção de recursos dos detalhes da implantação.
- Adicionar uma tarefa REST subsequente à tarefa de modelo de nuvem no pipeline.
- Associar a tarefa de modelo de nuvem à tarefa REST usando o endereço IP da máquina de nuvem no URL da tarefa REST.
- Executar o pipeline e observar o trabalho de associação da tarefa de modelo de nuvem com a tarefa REST.

#### Pré-requisitos

- Verifique se você tem um Modelo de Nuvem VMware operacional e com controle de versão.
- Verifique se a implantação do Modelo de Nuvem VMware foi bem-sucedida no Cloud Assembly.
- Verifique se você tem um pipeline que inclui uma tarefa de modelo de nuvem que usa esse Modelo de Nuvem VMware.
- Verifique se o pipeline foi executado com êxito.

#### Procedimentos

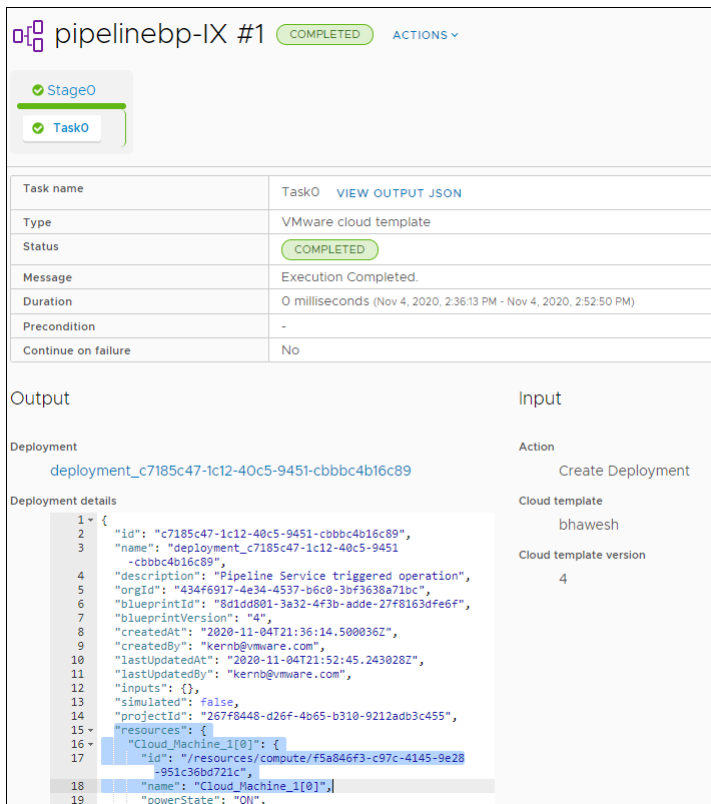
- 1 No seu pipeline, localize o endereço IP da máquina de nuvem na seção de recursos dos detalhes de implantação da sua tarefa de modelo de nuvem.
  - a Clique em **Ações > Exibir execuções**.
  - b Em um pipeline executado com êxito, clique no link para a execução do pipeline.



- c Sob o nome do pipeline, clique no link para a **Tarefa**.

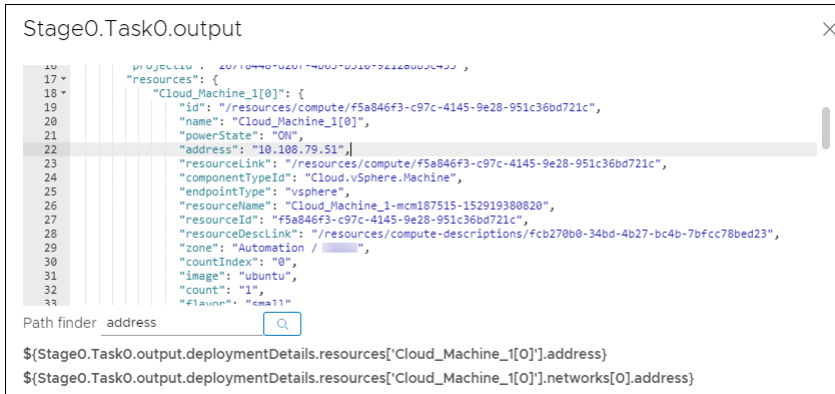


- d Na área Saída, localize os detalhes da Implantação.



- e Na seção de recursos dos detalhes da implantação, localize o nome da máquina de nuvem. Você incluirá a sintaxe do nome da máquina de nuvem na URL da sua tarefa REST.
- f Para encontrar a expressão de associação da propriedade de saída da tarefa de modelo de nuvem, clique em **EXIBIR JSON DE SAÍDA**, procure a propriedade de endereço e localize o endereço IP da máquina de nuvem.

A expressão de associação aparece abaixo da propriedade e do ícone de pesquisa na saída JSON.



A propriedade de recurso de endereço exibe o endereço IP da máquina de nuvem. Por exemplo:

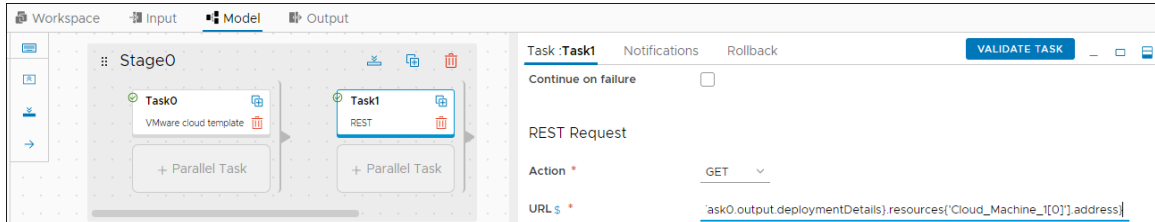
```
"resources": {
  "Cloud_Machine_1[0]": {
    "name": "Cloud_Machine_1[0]",
    "powerState": "ON",
    "address": "10.108.79.51",
    "resourceName": "Cloud_Machine_1-mcm187515-152919380820"
```

- 2 Retorne ao seu modelo de pipeline e insira a URL na sua tarefa REST.
  - a Clique em **Ações > Exibir Pipeline**.
  - b Clique na tarefa REST.

- c Na área URL da Solicitação REST, insira \$, selecione o **Estágio**, a **Tarefa**, a **saída**, **deploymentDetails** e insira **resources**.

A capacidade de digitar antecipadamente com preenchimento automático está disponível até o ponto em que você deve inserir **resources**.

- d Insira o restante do recurso de máquina de nuvem a partir dos detalhes da implantação como: `{ 'Cloud_Machine_1[0]' }.address`



Para a entrada da máquina de nuvem, você deve usar a notação de colchetes, conforme mostrado.

O formato completo da URL é: \$

```
{Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}
```

- 3 Execute o pipeline e observe a tarefa REST usar a máquina de nuvem e o endereço IP da saída da tarefa de modelo de nuvem como a URL a ser testada.

## Resultados

Parabéns! Você encontrou o nome e o endereço IP da máquina de nuvem nos detalhes da implantação e a saída JSON de uma tarefa de modelo de nuvem e os usou para associar a saída da tarefa do modelo de nuvem à entrada da URL da tarefa REST no seu pipeline.

## Próximo passo

Continue a explorar usando variáveis de associação de recursos na tarefa de modelo de nuvem com outras tarefas no seu pipeline.

# Como usar uma REST API para integrar o Code Stream a outros aplicativos

O Code Stream fornece um plug-in REST, que permite integrar o Code Stream com outros aplicativos que usam uma REST API para que você possa desenvolver e entregar continuamente aplicativos de software que devem interagir entre si. O plug-in REST invoca uma API, que envia e recebe informações entre o Code Stream e outro aplicativo.

Com o plug-in REST, é possível:

- Integrar sistemas externos com base em REST API em um pipeline do Code Stream.
- Integrar um pipeline do Code Stream como parte do fluxo de sistemas externos.

O plug-in REST funciona com qualquer REST API e oferece suporte aos métodos GET, POST, PUT, PATCH e DELETE para enviar ou receber informações entre o Code Stream e outros aplicativos.

**Tabela 5-7. Como preparar um pipeline para se comunicar pela REST API**

O que você faz	O que acontece como resultado
Adicione uma tarefa REST ao pipeline.	A tarefa REST comunica informações entre aplicativos e pode fornecer informações de status para uma tarefa sucessiva no estágio do pipeline.
Na tarefa REST, selecione a ação REST e inclua o URL.	A tarefa de pipeline chama o URL quando o pipeline é executado. Para ações de POST, PUT e PATCH, você deve incluir um payload. No payload, é possível associar as propriedades de pipeline e de tarefa quando o pipeline é executado.
Considere este exemplo.	Exemplo de uso do plug-in REST: É possível adicionar uma tarefa REST para criar uma tag em uma confirmação de Git para uma compilação e fazer com que a tarefa publique uma solicitação para obter o ID de verificação do repositório. A tarefa pode enviar um payload para o repositório e criar uma tag para a compilação, e o repositório pode retornar a resposta com a tag.

Semelhante a usar o plug-in REST para invocar uma API, é possível incluir uma tarefa de Sondagem no pipeline para invocar uma REST API e sondá-la até que seja concluída e a tarefa do pipeline atenda aos critérios de saída.

Também é possível usar as REST APIs para importar e exportar um pipeline, e usar os scripts de exemplo para executar um pipeline.

Este procedimento obtém um URL simples.

#### Procedimentos

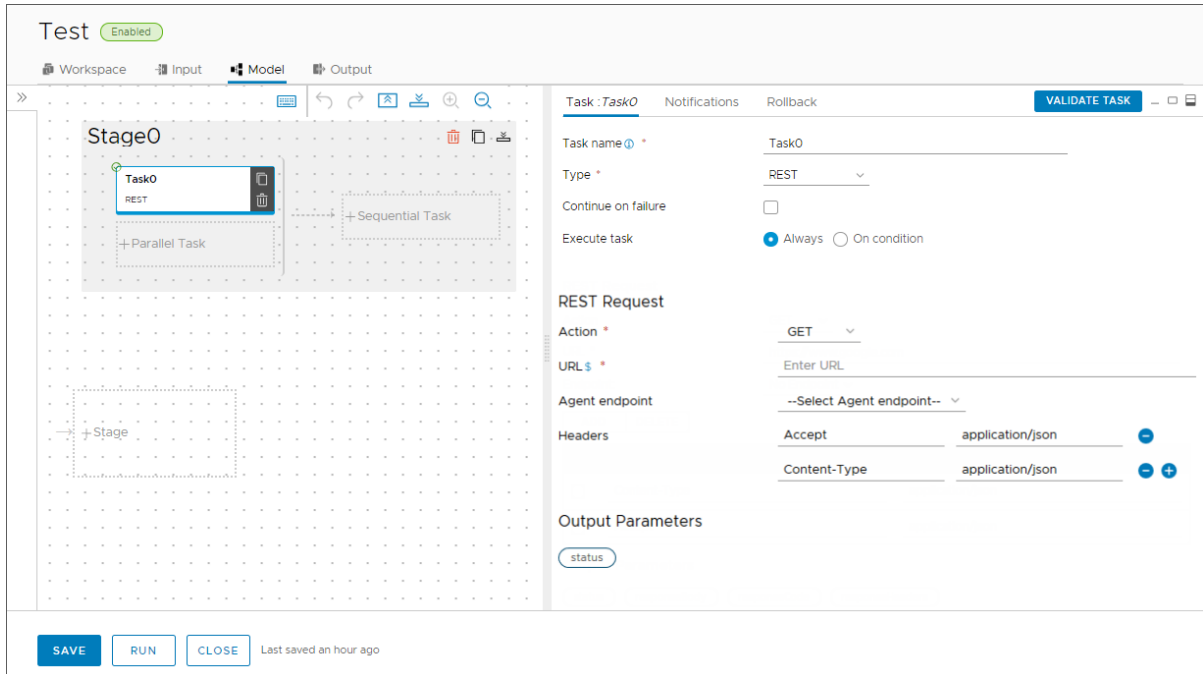
- 1 Para criar um pipeline, clique em **Pipelines > Novo Pipeline > Tela em Branco**.
- 2 No estágio do pipeline, clique em **+ Tarefa Sequencial**.
- 3 No painel de tarefas, adicione a tarefa REST:
  - a Digite um nome para a tarefa.
  - b No menu suspenso Tipo, selecione **REST**.
  - c Na área de Solicitação REST, selecione **GET**.

Para que a tarefa REST solicite dados de outro aplicativo, selecione o método GET. Para enviar dados para outro aplicativo, selecione o método POST.

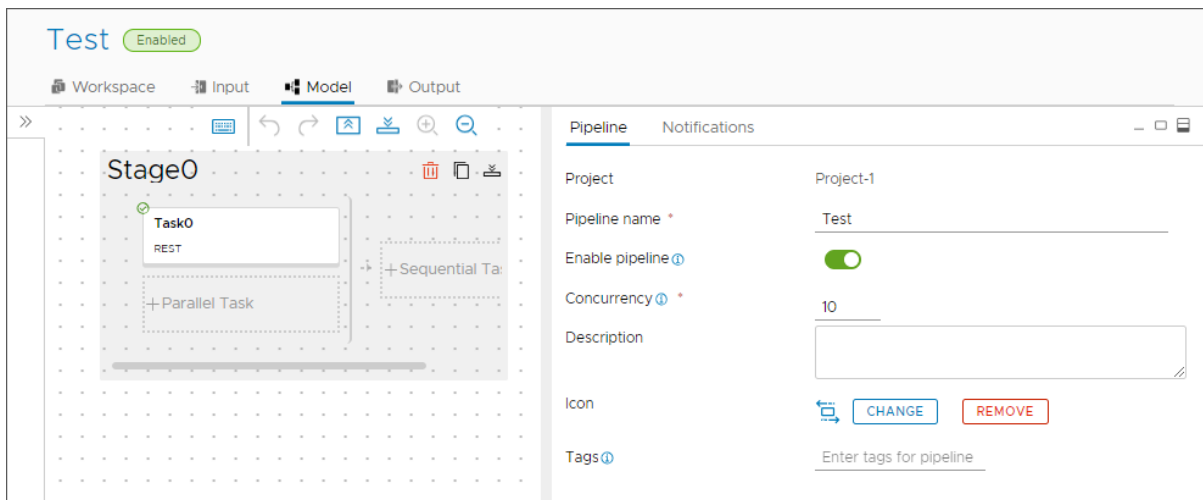
- d Digite o URL que identifique o endpoint da REST API. Por exemplo, `https://www.google.com`.

Para que uma tarefa REST importe dados de outro aplicativo, é possível incluir a variável de payload. Por exemplo, para uma ação de importação, é possível digitar `${Stage0.export.responseBody}`. Se o tamanho dos dados da resposta exceder 5 MB, a tarefa REST poderá falhar.

- e Para fornecer autorização para a tarefa, clique em **Adicionar Cabeçalhos** e digite uma chave de cabeçalho e um valor.



- 4 Para salvar o pipeline, clique em **Salvar**.
- 5 Na guia pipeline, clique em **Ativar pipeline**.



- 6 Clique em **Salvar**, depois clique em **Fechar**.



- 7 Clique em **Executar**.
- 8 Para observar a execução do pipeline, clique em **Execuções**.

The screenshot displays the 'Executions' interface in vRealize Automation. At the top, the title 'Executions' is followed by a badge indicating '10 items'. Below the title is a '+ NEW EXECUTION' button and a search bar. The main content area shows a table with one execution entry:

Icon	Name	Status	Stages	Actions
	Test#1	RUNNING	Stages: <div><div></div></div>	<a href="#">ACTIONS</a> ▾
		By system-user on 11/26/2018 3:11 PM		
		RUNNING		
		☆ Input : n/a ☆ Output : n/a		

- 9 Para verificar se a tarefa REST retorna as informações esperadas, examine a execução do pipeline e os resultados da tarefa.
  - a Após a conclusão do pipeline, para confirmar que o outro aplicativo retornou os dados solicitados, clique no link para a execução do pipeline.
  - b Clique na tarefa REST no pipeline.
  - c Na execução do pipeline, clique na tarefa, observe os detalhes da tarefa e verifique se a tarefa REST retornou os resultados esperados.

Os detalhes da tarefa exibem o código de resposta, o corpo, as chaves de cabeçalho e os valores.

[< BACK](#)

**Test #2** COMPLETED [ACTIONS](#)

Stage0

Task0

Task name	Task0 <a href="#">VIEW OUTPUT JSON</a>												
Type	REST												
Status	<span>COMPLETED</span> Execution Completed.												
Duration	1s (11/26/2018 3:45 PM - 11/26/2018 3:45 PM)												
Continue on failure	<input type="checkbox"/>												
Execute task	<input checked="" type="radio"/> Always <input type="radio"/> On condition												
Response													
Code	200												
Body	<pre> &lt;!doctype html&gt;&lt;html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"&gt;&lt;head&gt;&lt;meta content="text/html; charset=UTF-8" http-equiv="Content-Type"&gt;&lt;meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"&gt;&lt;title&gt;Google&lt;/title&gt;&lt;script nonce="aHwW/ydugkGr9CHU6QQGzg=="&gt;(function(){window.google={kEI:"cnf8W6KpJIEVkwXx-aLoDA",kEXPI:"0,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,6056,636,2239,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284,2,579,727,612,1820,58,2,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483,698,59,318,273,108,167,323,744,101,1119,38,363,557,438,135,145,155,497,2,718,383,978,487,47,1080,901,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,247,109,1049,14,758,7,127,179,9,21,261,1413,5977597,12,1861,681,134,43,5997424,90,2800095,4,1572,549,332,445,1,2,80,1,900,583,6,307,1,8,1,2,2132,1,1,1,1,1,1,748,141,297,169,301,24,2,8,96,50,2,47,22307501',authuser:0,kscs:'c9c918f0_cnf8W6KpJIEVkwXx-aLoDA',kGL:'IN';google.kHL='en-IN'}});google.time=function(){return(new Date).getTime()};(function(){google.lc=[];google.li=0;google.getEI=function(a){for(var b;a&amp;&amp;(a.getAttribute('')!==(b=a.getAttribute('eid'))));a=p </pre>												
Headers	<table> <thead> <tr> <th>Header Key</th> <th>Header Value</th> </tr> </thead> <tbody> <tr> <td>X-Frame-Options</td> <td>SAMEORIGIN</td> </tr> <tr> <td>Transfer-Encoding</td> <td>chunked</td> </tr> <tr> <td>Cache-Control</td> <td>private, max-age=0</td> </tr> <tr> <td>Server</td> <td>gws</td> </tr> <tr> <td>Alt-Svc</td> <td>quic="442";ma=350000;v="44,42,20,25"</td> </tr> </tbody> </table>	Header Key	Header Value	X-Frame-Options	SAMEORIGIN	Transfer-Encoding	chunked	Cache-Control	private, max-age=0	Server	gws	Alt-Svc	quic="442";ma=350000;v="44,42,20,25"
Header Key	Header Value												
X-Frame-Options	SAMEORIGIN												
Transfer-Encoding	chunked												
Cache-Control	private, max-age=0												
Server	gws												
Alt-Svc	quic="442";ma=350000;v="44,42,20,25"												

10 Para exibir a saída JSON, clique em **EXIBIR JSON DE SAÍDA**.

```

1  {
2    "responseHeaders": {
3      "X-Frame-Options": "SAMEORIGIN",
4      "Transfer-Encoding": "chunked",
5      "Cache-Control": "private, max-age=0",
6      "Server": "gws",
7      "Alt-Svc": "quic=\":443\"; ma=2592000; v=\":44,43,39,35\"",
8      "Set-Cookie": "NID=148
          =RTUkVjVhyg9KvAZR1S8yCCSEw8WosYf9mWdfQ1N5fnd5DavrXUM5B3J8PyKMX1Z_zRNp3usxttMpd7YiqRUOSfMkTC7cTERbd
          UmOnj3cTppHe3PHIXJPGHnTSZEweb3cxtjvIHv0LS85ezVXaTSRYFcG0B_XIHZ8kqB8uwl1aE; expires=Tue, 28-May-2019
          22:45:06 GMT; path=/; domain=.google.com; HttpOnly",
9      "Expires": "-1",
10     "P3P": "CP=\\\"This is not a P3P policy! See g.co/p3phelp for more info.\\\"",
11     "X-XSS-Protection": "1; mode=block",
12     "Date": "Mon, 26 Nov 2018 22:45:06 GMT",
13     "Content-Type": "text/html; charset=ISO-8859-1"
14   },
15   "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/google/1x/google_standard_color_128dp.png\\\" itemprop=\\\"image\\\"><title>Google</title><script
          nonce=\\\"aNww/ydugkGr9CHU6QQGz==\\\">(function(){window.google={keyI:'cnf8w6KpJIEVkwXx-aLoDA',keyPI:'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
          ,2,579,727,612,1820,58,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
          ,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
          "

```

## Resultados

Parabéns! Você configurou uma tarefa REST que invocou uma REST API e enviou informações entre o Code Stream e outro aplicativo usando o plug-in REST.

## Próximo passo

Continue a usar tarefas REST nos pipelines para executar comandos e integrar o Code Stream a outros aplicativos para poder desenvolver e fornecer seus aplicativos de software. Considere o uso de tarefas de sondagem que pesquisam a API até que ela seja concluída e a tarefa de pipeline atenda aos critérios de saída.

# Como alavancar o pipeline como código no Code Stream

Como administrador ou desenvolvedor de DevOps, você pode querer criar um pipeline no Code Stream usando o código YAML, em vez de usar a interface do usuário. Ao criar pipelines como código, você pode usar qualquer editor e inserir comentários no código do pipeline.

No seu código de pipeline, você pode fazer referência a configurações externas, como variáveis de ambiente e credenciais de segurança. Ao atualizar as variáveis que você usa no seu código de pipeline, a atualização pode ser feita sem a necessidade de atualizar o código do pipeline.

Você pode usar o código YAML do pipeline como modelo para clonar e criar outros pipelines e compartilhar os modelos com outros.

Você pode armazenar seus modelos de código de pipeline em um repositório de controle de origem, que cria versões desses modelos e rastreia atualizações. Usando um sistema de controle de origem, você pode fazer backup do seu código de pipeline facilmente e restaurá-lo se necessário.

#### Pré-requisitos

- Verifique se você tem um editor de código.
- Se você planeja armazenar seu código de pipeline em um repositório de controle de origem, verifique se é possível acessar uma instância operacional.

#### Procedimentos

- 1 No editor de códigos, crie um arquivo.
- 2 Copie e cole o código do pipeline de amostra e atualize-o para refletir as suas necessidades de pipeline específicas.
- 3 Para incluir um endpoint do no seu código de pipeline, copie e cole o exemplo de código do endpoint e atualize-o para refletir seu endpoint.

Ao usar um endpoint API do Kubernetes no espaço de trabalho do pipeline, o Code Stream cria os recursos do Kubernetes necessários, como ConfigMap, Segredo e Pod, para executar a tarefa de integração contínua (CI) ou personalizada. O Code Stream se comunica com o contêiner usando a NodePort.

O espaço de trabalho do pipeline do Code Stream é compatível com o Docker e o Kubernetes para tarefas de integração contínua e personalizadas.

Para obter mais informações sobre a configuração do espaço de trabalho, consulte [Como configurar o espaço de trabalho do pipeline](#).

- 4 Salve o código.
- 5 Para armazenar e criar uma versão do seu código de pipeline, verifique o código no repositório de controle de origem.
- 6 Ao criar um pipeline de integração e entrega contínuas, você deve importar o arquivo YAML do Kubernetes.

Para importar o arquivo YAML do Kubernetes, selecione-o na área Entrega Contínua do modelo de pipeline inteligente e clique em **Processar**. Se preferir, use a API.

#### Resultados

Usando os exemplos de código, você criou o código YAML que representa o pipeline e os endpoints.

### Exemplo: Exemplo de código YAML para um pipeline e endpoints

Este exemplo de código YAML inclui seções que representam o espaço de trabalho para a compilação nativa do Code Stream, estágios, tarefas, notificações e muito mais em um pipeline.

Para obter exemplos de códigos para plug-ins com suporte, consulte [Capítulo 6 Como conectar o Code Stream aos endpoints](#)

```
---
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}
---
```

Este código YAML representa um endpoint do Jenkins de exemplo.

```

---
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
---
```

Este código YAML representa um endpoint do Kubernetes de exemplo.

```

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

### Próximo passo

Execute o pipeline e faça os ajustes necessários. Consulte [Como executar um pipeline e ver os resultados](#).



# Como conectar o Code Stream aos endpoints

## 6

O Code Stream integra-se às ferramentas de desenvolvimento por meio de plug-ins. Os plug-ins com suporte incluem Jenkins, Bamboo, vRealize Operations, Bugzilla, Team Foundation Server, Git e muito mais.

Também é possível desenvolver seus próprios plug-ins que integram o Code Stream a outros aplicativos de desenvolvimento.

Para integrar o Code Stream com o Jira, você não precisa de um plug-in externo, pois o Code Stream inclui o recurso de criação de tíquetes do Jira como um tipo de notificação. Para criar tíquetes de Jira no status do pipeline, é necessário adicionar um endpoint do Jira.

Este capítulo inclui os seguintes tópicos:

- [O que são endpoints no Code Stream](#)
- [Como integrar o Code Stream ao Jenkins](#)
- [Como integrar o Code Stream ao Git](#)
- [Como integrar o Code Stream ao Gerrit](#)
- [Como integrar o Code Stream ao vRealize Orchestrator](#)

## O que são endpoints no Code Stream

Um endpoint é uma instância de um aplicativo DevOps que se conecta ao Code Stream e fornece dados para que seus pipelines sejam executados, como uma fonte de dados, repositório ou sistema de notificação.

Sua função no Code Stream determina como você usa os endpoints.

- Os administradores e desenvolvedores podem criar, atualizar, excluir e visualizar endpoints.
- Os administradores podem marcar um endpoint como restrito e executar pipelines que usem endpoints restritos.
- Os usuários que têm a função de visualizador podem ver os endpoints, mas não podem criá-los, atualizá-los ou excluí-los.

Para obter mais informações, consulte [Como gerenciar o acesso do usuário e as aprovações no Code Stream](#).

Para conectar o Code Stream a um endpoint, siga estas etapas.

- 1 Adicionar uma tarefa ao pipeline
- 2 Configure a tarefa para que ela se comunique com o endpoint.
- 3 Verifique se o Code Stream pode se conectar ao endpoint, clicando em **Validar**.
- 4 Em seguida, quando você executar o pipeline, a tarefa se conectará ao endpoint para que este possa executar a tarefa.

Para obter informações sobre os tipos de tarefas que usam esses endpoints, consulte [Que tipos de tarefas estão disponíveis no Code Stream](#).

**Tabela 6-1. Endpoints que o Code Stream suporta**

Endpoint	O que ele fornece	Versões com suporte	Requisitos
Bamboo	Cria planos de compilação.	6.9.*	
Docker	Compilações nativas podem usar hosts do Docker para implantação.		Quando um pipeline inclui uma imagem do Docker Hub, você deve garantir que essa imagem tenha <code>cURL</code> ou <code>wget</code> incorporado antes de executar o pipeline. Quando o pipeline é executado, o Code Stream baixa um arquivo binário que usa <code>cURL</code> ou <code>wget</code> para executar comandos.
Registro do Docker	Registra as imagens do contêiner para que um host de compilação do Docker possa receber imagens.	2.7.1	
Gerrit	Conecta-se a um servidor Gerrit para revisões e gatilho	2.14.*	
Git	Dispara pipelines quando os desenvolvedores atualizam o código e o verificam no repositório.	Git Hub Enterprise 2.1.8 Git Lab Enterprise 11.9.12-ee	
Jenkins	Compila artefatos de código.	1.6.* e 2.*	
Jira	Cria um tíquete do Jira quando uma tarefa de pipeline falha.	8.3.*	

Tabela 6-1. Endpoints que o Code Stream suporta (continuação)

Endpoint	O que ele fornece	Versões com suporte	Requisitos
Kubernetes	Automatiza as etapas que implantam, dimensionam e gerenciam aplicativos em contêiner.	Todas as versões com suporte para o Cloud Assembly 8.4 e posterior 1.18 para o Cloud Assembly 8.3 e versões anteriores	Ao usar um endpoint API do Kubernetes no espaço de trabalho do pipeline, o Code Stream cria os recursos do Kubernetes necessários, como ConfigMap, Segredo e Pod, para executar a tarefa de integração contínua (CI) ou personalizada. O Code Stream se comunica com o contêiner usando a NodePort. Para obter mais informações sobre a configuração do espaço de trabalho, consulte <a href="#">Como configurar o espaço de trabalho do pipeline</a> .
PowerShell	Crie tarefas que executam scripts do PowerShell em máquinas Windows ou Linux.	4 e 5	
SSH	Crie tarefas que executam scripts SSH em máquinas Windows ou Linux.	7.0	
TFS, Team Foundation Server	Gerencia código-fonte, compilações automatizadas, testes e atividades relacionadas.	2015 e 2017	
vRealize Orchestrator	Organiza e automatiza os fluxos de trabalho no seu processo de compilação.	7.* e 8.*	

## Exemplo de código YAML para um endpoint do GitHub

Este exemplo de código YAML define um endpoint do GitHub que pode ser consultado em uma tarefa Git.

```
---
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
---
```

## Como integrar o Code Stream ao Jenkins

O Code Stream fornece um plug-in Jenkins, que dispara trabalhos do Jenkins que compilam e testam seu código-fonte. O plug-in Jenkins executa casos de teste e pode usar scripts personalizados.

Para executar um trabalho do Jenkins no pipeline, use um servidor Jenkins e adicione o endpoint do Jenkins no Code Stream. Em seguida, crie um pipeline e adicione uma tarefa Jenkins a ele.

Ao usar a tarefa do Jenkins e um endpoint Jenkins no Code Stream, você pode criar um pipeline que oferece suporte a trabalhos de várias ramificações no Jenkins. O trabalho de várias ramificações inclui trabalhos individuais em cada ramificação de um repositório Git. Quando você cria pipelines no Code Stream que oferecem suporte a trabalhos de várias ramificações:

- A tarefa do Jenkins pode executar trabalhos do Jenkins que residem em várias pastas no servidor do Jenkins.
- Você pode substituir o caminho da pasta na configuração da tarefa do Jenkins para usar um caminho de pasta diferente, que substitui o caminho padrão definido no endpoint Jenkins no Code Stream.
- Pipelines de várias ramificações no Code Stream detectam arquivos de trabalho do Jenkins do tipo `.groovy` em um repositório Git ou em um repositório GitHub e começam a criar trabalhos para cada ramificação verificada no repositório.
- Você pode substituir o caminho padrão definido no endpoint Jenkins por um caminho fornecido na configuração da tarefa do Jenkins e executar um trabalho e pipeline que esteja associado a qualquer ramificação dentro de um trabalho do Jenkins principal.

### Pré-requisitos

- Configure um servidor Jenkins que execute a versão 1.561 ou mais recente.
- Verifique se você é membro de um projeto no Code Stream. Se você não for membro, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).
- Verifique se existe um trabalho no servidor Jenkins para que a tarefa do pipeline possa executá-lo.

## Procedimentos

- 1 Adicione e valide um endpoint do Jenkins.
  - a Clique em **Endpoints > Novo Endpoint**.
  - b Selecione um projeto e, para o tipo de endpoint, selecione **Jenkins**. Em seguida, insira um nome e uma descrição.
  - c Se esse endpoint for um componente crítico para os negócios em sua infraestrutura, ative **Marcar como restrito**.
  - d Insira o URL do servidor Jenkins.

- e Digite o nome do usuário e a senha para fazer login no servidor Jenkins. Em seguida, insira as informações restantes.

**Tabela 6-2. Informações restantes para o endpoint do Jenkins**

Entrada do endpoint	Descrição
Caminho da pasta	<p>Caminho para a pasta que agrupa seus trabalhos. O Jenkins pode executar todos os trabalhos na pasta. É possível criar subpastas. Por exemplo:</p> <ul style="list-style-type: none"> <li>■ <code>folder_1</code> pode incluir <code>job_1</code></li> <li>■ <code>folder_1</code> pode incluir <code>pasta_2</code>, que pode incluir <code>job_2</code></li> </ul> <p>Quando você cria um endpoint para <code>folder_1</code>, o caminho da pasta é <code>job/folder_1</code> e o endpoint lista apenas <code>job_1</code>.</p> <p>Para obter a lista de trabalhos na pasta filho chamada <code>folder_2</code>, você deve criar outro endpoint que use o caminho da pasta como <code>/job/folder_1/job/folder_2/</code>.</p>
Caminho da pasta para trabalhos do Jenkins de várias ramificações	<p>Para oferecer suporte a trabalhos do Jenkins de várias ramificações, na tarefa do Jenkins, insira o caminho completo que inclui a URL do servidor do Jenkins e o caminho completo do trabalho. Quando você inclui um caminho de pasta na tarefa do Jenkins, esse caminho substitui o caminho que aparece no endpoint Jenkins. Com o caminho da pasta personalizada na tarefa do Jenkins, o Code Stream exibe apenas os trabalhos que estão presentes nessa pasta.</p> <ul style="list-style-type: none"> <li>■ Por exemplo, <code>https://server.yourcompany.com/job/project</code></li> <li>■ Se o pipeline também tiver que acionar o trabalho do Jenkins principal, use: <code>https://server.yourcompany.com/job/project/job/main</code></li> </ul>
URL	<p>URL do host do servidor Jenkins. Insira a URL no formato <code>protocolo://host:porta</code>. Por exemplo: <code>http://192.10.121.13:8080</code></p>
Intervalo de Sondagem	<p>Duração do intervalo para o Code Stream pesquisar atualizações para o servidor Jenkins.</p>

Tabela 6-2. Informações restantes para o endpoint do Jenkins (continuação)

Entrada do endpoint	Descrição
Contagem de repetição de solicitação	Número de vezes para repetir a solicitação de compilação agendada para o servidor Jenkins.
Tempo de espera de repetição	Número de segundos a aguardar antes de tentar novamente a solicitação de compilação para o servidor Jenkins.

- f Clique em **Validar** e verifique se o endpoint se conecta ao Code Stream. Se ele não se conectar, corrija os erros e clique em **Salvar**.

## Edit Endpoint

Project

test1

Type

Jenkins

Name \*

aa

Description

Mark restricted

☐ non-restricted

URL \*


http(s)://<server\_url>:<port>

Username

username

Password

Enter password



CREATE VARIABLE

Folder Path

/job/DevFolder/

Poll Interval (sec) \*

15

Request Retries \*

5

Retry Wait Time (sec) \*

60

SAVE

VALIDATE

CANCEL

- 2 Para compilar seu código, crie um pipeline e adicione uma tarefa que use o endpoint do Jenkins.
  - a Clique em **Pipelines > Novo Pipeline > Tela em Branco**.
  - b Clique no estágio padrão.
  - c Na área de tarefas, digite um nome para a tarefa.
  - d Selecione o tipo de tarefa como **Jenkins**.
  - e Selecione o endpoint do Jenkins criado.
  - f No menu suspenso, selecione um trabalho no servidor Jenkins que será executado pelo pipeline.
  - g Digite os parâmetros do trabalho.
  - h Insira o token de autenticação para o trabalho do Jenkins.

The screenshot displays the vRealize Automation Code Stream interface for configuring a pipeline named 'Build and Deploy' (status: Enabled).

**Left Pane (Visual Workflow):**

- Stage0:** Contains two tasks: 'Build' and 'Test', both of type 'Jenkins'.
- Parallel Task:** A dashed box labeled '+ Parallel Task' is shown below the stage.
- Stage:** A dashed box labeled '+ Stage' is shown at the bottom of the workflow area.

**Right Pane (Task Configuration):**

- Task:** Build
- Task name:** Build
- Type:** Jenkins
- Continue On Failure:** ☐
- Execute Task:** ☒ Always ☐ On Condition
- Jenkins:**
  - Endpoint:** aa
  - Job:** add\_numbers
  - Num1:** 22
  - Num2:** 22
  - Token:** (empty field)
- Output Parameters:** status, job, jobId, jobResults, jobUrl


**Bottom Bar:**

- Buttons:** SAVE, RUN, CLOSE
- Status:** Last saved a month ago
- Chat:** A circular chat icon is located in the bottom right corner.



3 Ative e execute o pipeline e exiba a execução dele.

[< BACK](#)



# Build and Deploy #28

COMPLETED
0
ACTIONS

Stage0

✓ Build
✓ Test
✓ Approval for Deployment
✓ Deployment
✓ Wait for application to start

Task name

Build [VIEW OUTPUT JSON](#)

Type

Jenkins

Status

COMPLETED Execution Completed.
 

Duration

11s (08/06/2018 12:27 AM - 08/06/2018 12:27 AM)

Continue On Failure

☐

Execute Task

Always ☐ On Condition
 

Jenkins Job

Endpoint

aa

Job Name

add\_numbers

Job ID

1428

Job URL

[http://.../job/add\\_numbers/1428/](http://.../job/add_numbers/1428/)

Job Result

Key	Value
junitResponse.failCount	0
junitResponse.skipCount	0
junitResponse.totalCount	0
junitResponse.successCount	0
jacocoResponse.lineCoverage	0
jacocoResponse.classCoverage	0

4 Observe os detalhes da execução e o status no painel de pipeline.

É possível identificar quaisquer falhas e por que elas ocorreram. Também é possível ver tendências sobre as durações, conclusões e falhas de execução do pipeline.

The screenshot shows the 'Build and Deploy' interface. At the top, there are tabs for '1D', '7D', and '14D'. Below this is a 'Recent Executions' section with a list of execution numbers (#20 to #29) and a visual representation of their status (green for completed, red for failed). A legend indicates: green dot for Completed, red dot for Failed, blue dot for Running, and yellow dot for Waiting.

Below the 'Recent Executions' is the 'Execution Details' section, which contains a table with the following data:

Execution#	Status	Status Message	Duration	Updated On
#29	FAILED	Execution failed on task 'Stage0.Deployment'. namespaces "prod1" already exists	1m 32s	08/19 10:49PM
#28	COMPLETED	Execution Completed.	3m 42s	08/06 12:30AM
#27	COMPLETED	Execution Completed.	1m 45s	08/06 12:24AM
#26	FAILED	Execution failed on task 'Stage0.Deployment'. Conflict	1m 8s	08/06 12:19AM
#25	COMPLETED	Execution Completed.	2m 11s	08/06 12:07AM
#24	COMPLETED	Execution Completed.	58s	08/05 11:59PM
#23	FAILED	Execution failed on task 'Stage0.Approval for Deployment'. User Operation request has been	4m 55s	08/06 12:03AM

At the bottom right of the interface is a chat icon.

## Resultados

Parabéns! Você integrou o Code Stream ao Jenkins adicionando um endpoint, criando um pipeline e configurando uma tarefa Jenkins que compila seu código.

## Exemplo: Exemplo de YAML para uma tarefa de compilação Jenkins

Para o tipo de tarefa de compilação Jenkins usado neste exemplo, o YAML deve ser semelhante ao código a seguir, com notificações ativadas:

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

## Próximo passo

Revise as outras seções para saber mais. Consulte [Capítulo 6 Como conectar o Code Stream aos endpoints](#).

## Como integrar o Code Stream ao Git

O Code Stream fornece uma maneira de disparar um pipeline se uma alteração de código ocorrer no repositório GitHub, GitLab ou Bitbucket. O gatilho Git usa um endpoint do Git na ramificação do repositório que você deseja monitorar. O Code Stream conecta-se ao endpoint do Git por meio de um webhook.

Para definir um endpoint do Git no Code Stream, selecione um projeto e insira a ramificação do repositório Git em que o endpoint está localizado. O projeto agrupa o pipeline com o endpoint e outros objetos relacionados. Ao escolher o projeto na definição de webhook, selecione o endpoint e o pipeline para disparar.

---

**Observação** Se você definir um webhook com seu endpoint e, em seguida, editar o endpoint, não poderá alterar os detalhes do endpoint no webhook. Para alterar os detalhes do endpoint, você deve excluir e redefinir o webhook com o endpoint. Consulte [Como usar o gatilho Git no Code Stream para executar um pipeline](#).

---

Você pode criar vários webhooks para ramificações diferentes usando o mesmo endpoint Git e fornecendo diferentes valores para o nome da ramificação na página de configuração do webhook. Para criar outro webhook para outra ramificação no mesmo repositório Git, não é necessário clonar o endpoint Git várias vezes para várias ramificações. Em vez disso, forneça o nome da ramificação no webhook, o que permite reutilizar o endpoint Git. Se a ramificação no webhook Git for a mesma que a ramificação no endpoint, você não precisará fornecer o nome da ramificação na página do webhook Git.

### Pré-requisitos

- Verifique se é possível acessar o repositório GitHub, GitLab ou Bitbucket ao qual pretende se conectar.
- Verifique se você é membro de um projeto no Code Stream. Se não for, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).

### Procedimentos

- 1 Defina um endpoint do Git.
  - a Clique em **Endpoints > Novo Endpoint**.
  - b Selecione um projeto e para o tipo de endpoint e selecione **Git**. Em seguida, digite um nome e uma descrição.

- c Se esse endpoint for um componente crítico para os negócios em sua infraestrutura, ative **Marcar como restrito**.

Quando você usa um endpoint restrito em um pipeline, um administrador pode executar o pipeline e deve aprovar a execução desse pipeline. Se um endpoint ou uma variável estiver marcado como restrito, e um usuário não administrativo disparar o pipeline, este fará uma pausa nessa tarefa e esperará que um administrador a retome.

Um administrador de projeto poderá iniciar um pipeline que inclua endpoints ou variáveis restritos se esses recursos estiverem no projeto em que o usuário é um Administrador de projeto.

Quando um usuário que não é administrador tenta executar um pipeline que inclui um recurso restrito, o pipeline é interrompido na tarefa que usa esse recurso restrito. Em seguida, um administrador deve retomar o pipeline.

Para obter mais informações sobre recursos restritos e funções personalizadas que incluem a permissão chamada **Gerenciar Pipelines Restritos**, consulte:

- [Como gerenciar o acesso do usuário e as aprovações no Code Stream](#)
- [Capítulo 2 Como configurar o Code Stream para modelar meu processo de liberação](#)

- d Selecione um dos tipos de servidor Git compatíveis.

- e Insira a URL do repositório com o gateway de API para o servidor no caminho. Por exemplo:

Para o GitHub, insira: `https://api.github.com/vmware-example/repo-example`

Para o BitBucket, insira: `https://api.bitbucket.org/{user}/{repo name}` ou `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`

- f Insira a ramificação no repositório no qual o endpoint está localizado.

- g Selecione o tipo de autenticação e digite o nome de usuário para GitHub, GitLab ou BitBucket. Em seguida, insira o token privado que acompanha o nome do usuário.

- Senha. Para criar um webhook mais tarde, você deverá inserir o token privado para a senha. Webhooks para Git não oferecem suporte a endpoints criados usando autenticação básica.

Use variáveis secretas para ocultar e criptografar informações confidenciais. Use a variável restrita para strings, senhas e URLs que devem ser ocultadas e criptografadas e para restringir o uso em execuções. Por exemplo, use uma variável secreta para uma senha ou URL. Você pode usar variáveis secretas e restritas em qualquer tipo de tarefa no seu pipeline.

- Token privado. Esse token é específico para o Git e fornece acesso a uma ação específica. Consulte [https://docs.gitlab.com/ee/user/profile/personal\\_access\\_tokens.html](https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html). Você também pode criar uma variável para o token privado.

- 2 Clique em **Validar** e verifique se o endpoint se conecta ao Code Stream.

Se ele não se conectar, corrija os erros e clique em **Criar**.

**New endpoint**

Project \* test

Type \* GIT

Name \* DemoApp-Git

Description Git example branch

Mark restricted ☐ non-restricted

Git Server Type \* GitHub

Repo URL ⓘ \* https://api.github.com/vmware-example/repo-example

ACCEPT CERTIFICATE

Branch \* master

Authentication Type \* Password

Username \* ExampleUser

Password \* .....

CREATE VARIABLE

CREATE VALIDATE CANCEL

### Próximo passo

Para saber mais, leia as outras seções. Consulte [Como usar o gatilho Git no Code Stream para executar um pipeline](#).

## Como integrar o Code Stream ao Gerrit

O Code Stream permite disparar um pipeline quando ocorre uma revisão de código no seu projeto Gerrit. O trigger para a definição Gerrit inclui o projeto Gerrit e os pipelines que devem ser executados para diferentes tipos de eventos.

O trigger do Gerrit usa um ouvinte Gerrit no servidor Gerrit que você monitorará. Para definir um endpoint do Gerrit no Code Stream, selecione um projeto e insira a URL do servidor Gerrit. Em seguida, especifique o endpoint ao criar um ouvinte Gerrit nesse servidor.

Se você estiver usando um servidor Gerrit como um endpoint do Code Stream em uma instância do vRealize Automation que tenha o FIPS ativado, deverá verificar se o seu arquivo de configuração do Gerrit inclui as chaves de autenticação de mensagem corretas. Se o arquivo de configuração do servidor Gerrit não incluir as chaves de autenticação de mensagem corretas, o servidor não poderá ser exibido corretamente e mostrará esta mensagem: `PrivateKey/PassPhrase is incorrect`

### Pré-requisitos

- Verifique se é possível acessar o servidor Gerrit ao qual pretende se conectar.
- Verifique se você é membro de um projeto no Code Stream. Se você não for membro, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).

### Procedimentos

#### 1 Defina um endpoint do Gerrit.

- a Clique em **Configurar > Endpoints** e clique em **Novo Endpoint**.
- b Selecione um projeto e, para o tipo de endpoint, selecione **Gerrit**. Em seguida, insira um nome e uma descrição.
- c Se esse endpoint for um componente crítico para os negócios em sua infraestrutura, ative **Marcar como restrito**.
- d Digite o URL do servidor Gerrit.  
  
Para usar a porta padrão, você pode fornecer um número de porta com a URL ou deixar o valor em branco.
- e Digite o nome de usuário e a senha para o servidor Gerrit.

Se a senha tiver que ser criptografada, clique em **Criar Variável** e selecione o tipo:

- Segredo. A senha é resolvida quando um usuário com qualquer função executa o pipeline.
- Restrito. A senha é resolvida quando um usuário que tem a função Admin executa o pipeline.

Para o valor, insira a senha que deve ser segura, como a senha de um servidor do Jenkins.

- f Para a chave privada, digite a chave SSH usada para acessar o servidor Gerrit de forma segura.

Essa chave é a chave privada RSA que reside no diretório `.ssh`.

- g (Opcional) Se uma frase-chave estiver associada à chave privada, digite a frase-chave.

Para criptografar a senha, clique em **Criar Variável** e selecione o tipo:

- **Segredo.** A senha é resolvida quando um usuário com qualquer função executa o pipeline.
- **Restrito.** A senha é resolvida quando um usuário que tem a função Admin executa o pipeline.

Para o valor, insira a senha que deve ser segura, como a senha para um servidor SSH.

- 2 Clique em **Validar** e verifique se o endpoint do Gerrit no Code Stream se conecta ao servidor Gerrit.

Se ela não estabelecer a conexão, corrija os erros e clique em **Validar** novamente.

**New endpoint**

Project: test

Type: Gerrit

Name \*: Gerrit-Demo-Endpoint

Description:

Mark restricted: ☐ non-restricted

URL \*: http://example-gerrit.mycompany.com:8080

Username \*: CS\_user

Password \*:

Private Key \*:

Pass Phrase ⓘ:

- 3 Clique em **Criar**.

- 4 Verifique se o ambiente do vRealize Automation está com o FIPS ativado ou faça com que o seu trabalho do Jenkins crie o ambiente com o FIPS ativado usando a URL do Jenkins.
  - a Para executar o comando na linha de comando, conecte-se ao dispositivo do vRealize Automation 8.x via SSH e faça login como usuário root. Por exemplo, conecte-se à URL do nome de domínio totalmente qualificado, como `https://cava-1-234-567.yourcompanyFQDN.com` na porta 22, 5480 ou 443.
  - b Para verificar o FIPS no vRealize Automation, execute o comando `vracli security fips`.
  - c Verifique se o comando retorna `FIPS mode: strict`.
- 5 Se o servidor Gerrit for um endpoint em uma instância do vRealize Automation que tenha o FIPS ativado, certifique-se de que o arquivo de configuração do Gerrit inclua as chaves corretas de autenticação de mensagem (MAC).
  - a Abra o Gerrit e crie um par de chaves SSH.
  - b Localize o arquivo de configuração do servidor Gerrit em `'$site_path'/etc/gerrit.config`.
  - c Verifique se o arquivo de configuração do servidor Gerrit inclui uma ou mais chaves de código de autenticação de mensagem (MAC), com exceção de `hmac-MD5`.

---

**Observação** No modo FIPS, `hmac-MD5` não é um algoritmo MAC com suporte. Para garantir que o servidor Gerrit seja iniciado corretamente, o arquivo de configuração do servidor Gerrit deve excluir esse algoritmo. Se o servidor Gerrit não for inicializado corretamente, ele exibirá esta mensagem: `PrivateKey/PassPhrase is incorrect`

---

Nomes de chaves do código de autenticação de mensagem (MAC) com suporte que começam com um sinal de mais (+) estão habilitados. Os nomes de chave MAC que começam com um hífen (-) são removidos da lista de MACs padrão. Por padrão, estes MACs com suporte estão disponíveis no Code Stream para o servidor Gerrit:

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

#### Próximo passo

Para saber mais, leia as outras seções. Consulte [Como usar o gatilho Gerrit no Code Stream para executar um pipeline](#).



## Como integrar o Code Stream ao vRealize Orchestrator

O Code Stream pode se integrar ao vRealize Orchestrator (vRO) para estender seu recurso executando fluxos de trabalho do vRO. O vRealize Orchestrator inclui muitos fluxos de trabalho predefinidos que podem ser integrados a ferramentas de terceiros. Esses fluxos de trabalho ajudam a automatizar e gerenciar seus processos de DevOps, automatizar as operações em massa e muito mais.

Por exemplo, é possível usar um fluxo de trabalho em uma tarefa do vRO no pipeline para ativar um usuário, remover um usuário, mover VMs, integrar a estruturas de teste para testar o código enquanto o pipeline é executado, e muito mais. É possível procurar exemplos de código para os fluxos de trabalho do vRealize Orchestrator em [code.vmware.com](https://code.vmware.com).

Com um fluxo de trabalho do vRealize Orchestrator, seu pipeline pode executar uma ação enquanto compila, testa e implanta o aplicativo. É possível incluir fluxos de trabalho predefinidos no pipeline ou criar e usar fluxos de trabalho personalizados. Cada fluxo de trabalho inclui entradas, tarefas e saídas.

Para executar um fluxo de trabalho do vRO no pipeline, o fluxo de trabalho deve aparecer na lista de fluxos de trabalho disponíveis na tarefa do vRO incluída no pipeline.

Antes que o fluxo de trabalho possa aparecer na tarefa do vRO no pipeline, um administrador deve realizar as seguintes etapas no vRealize Orchestrator:

- 1 Aplique a tag CODESTREAM ao fluxo de trabalho do vRO.
- 2 Marque o fluxo de trabalho do vRO como global.

### Pré-requisitos

- Verifique se, como administrador, você pode acessar uma instância local do vRealize Orchestrator. Para obter ajuda, consulte seu próprio administrador e a [documentação do vRealize Orchestrator](#).
- Verifique se você é membro de um projeto no Code Stream. Se não for, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).
- No Code Stream, crie um pipeline e adicione um estágio.

### Procedimentos

- 1 Como administrador, prepare um fluxo de trabalho do vRealize Orchestrator para que seu pipeline seja executado.
  - a No vRealize Orchestrator, localize o fluxo de trabalho que você precisa usar no pipeline, como um fluxo de trabalho para habilitar um usuário.  
Se um fluxo de trabalho que não existe for necessário, é possível criá-lo.
  - b Na barra de pesquisa, insira **Fluxo de trabalho de tag** para encontrar o fluxo de trabalho denominado Fluxo de trabalho de tag.

- c No cartão denominado `Fluxo de trabalho de tag`, clique em **Executar**, o que exibe a área de configuração.
- d Na área de texto `Fluxo de trabalho marcado`, insira o nome do fluxo de trabalho a ser usado no pipeline do Code Stream e, em seguida, selecione-o na lista.
- e Nas áreas de texto `Tag` e `Valor` áreas de texto, insira `CODESTREAM` em letras maiúsculas.
- f Clique na caixa de seleção denominada **Tag global**.
- g Clique em **Executar**, o que anexa a tag chamada `CODESTREAM` ao fluxo de trabalho que você precisa selecionar no pipeline do Code Stream.
- h No painel de navegação, clique em **Fluxos de trabalho** e confirme se a tag chamada `CODESTREAM` aparece no cartão de fluxo de trabalho em que o pipeline será executado.

Depois de fazer login no Code Stream e adicionar uma tarefa do `vRO` ao pipeline, o fluxo de trabalho marcado aparecerá na lista de fluxos de trabalho.

- 2 No Code Stream, crie um endpoint para a instância do vRealize Orchestrator.
  - a Clique em **Endpoints > Novo Endpoint**.
  - b Selecione um projeto.
  - c Digite um nome relevante.

- d Insira a URL do endpoint do vRealize Orchestrator.

Use este formato: **https://vro-appliance.yourdomain.local:8281**

Não use este formato: **https://vro-appliance.yourdomain.local:8281/vco/api**

A URL para uma instância do vRealize Orchestrator que está integrada no dispositivo do vRealize Automation, é o FQDN do dispositivo sem uma porta. Por exemplo: **https://vro-appliance.yourdomain.local/vco**

Para dispositivos do vRealize Orchestrator externos que começam com o vRealize Automation 8.x, o FQDN do dispositivo é **https://vro-appliance.yourdomain.local**

Para dispositivos do vRealize Orchestrator externos incluídos com o vRealize Automation 7.x, o FQDN do dispositivo é **https://vro-appliance.yourdomain.local:8281/vco**

Se ocorrer um problema ao adicionar o endpoint, talvez você precise importar uma configuração do YAML com uma impressão digital de certificado SHA-256 com os dois pontos removidos. Por exemplo, **B0:01:A2:72...** se torna **B001A272...** O código YAML de amostra é semelhante a:

```

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

- e Clique em **Aceitar Certificado** caso a URL inserida precise de um certificado.
- f Digite o nome do usuário e a senha para o servidor vRealize Orchestrator.

Se você estiver usando um usuário não local para autenticação, deverá omitir a parte do domínio do nome de usuário. Por exemplo, para se autenticar com **svc\_vro@yourdomain.local**, você deve **svc\_vro** na área de texto **Nome de usuário**.

- 3 Prepare seu pipeline para executar a tarefa do vRO.
- a Adicione uma tarefa do vRO ao estágio do pipeline.
  - b Digite um nome relevante.
  - c Na área Propriedades do Fluxo de Trabalho, selecione o endpoint do vRealize Orchestrator.

- d Selecione o fluxo de trabalho que você marcou como CODESTREAM no vRealize Orchestrator.

Se selecionar um fluxo de trabalho personalizado seu, talvez seja necessário digitar os valores do parâmetro de entrada.

- e Para **Executar tarefa**, clique em **Na condição**.

The screenshot shows the configuration window for a vRO workflow task. The interface includes tabs for 'Task : vRO workflow', 'Notifications', and 'Rollback'. A 'VALIDATE TASK' button is located in the top right corner. The main configuration area contains the following fields:

- Task name**: vRO workflow
- Type**: vRO (dropdown menu)
- Duration**: NaNps ( - )
- Continue on failure**: ☐
- Execute task**: ☐ Always ☒ On condition
- Condition**:  (empty text box with an information icon on the right)

Below these fields is the **Workflow Properties** section:

- Endpoint**: vROEP (dropdown menu)
- Workflow**: Test (dropdown menu)
- Greeting**: Hello! (text field)

At the bottom is the **Output Parameters** section, which contains two buttons: 'status' and 'properties'.

- f Insira as condições aplicáveis à execução do pipeline.

Quando executar o pipeline...	Selecionar condições...
Na Condição	<p>Executa a tarefa de pipeline somente se a condição definida for avaliada como verdadeira. Se a condição for falsa, a tarefa será ignorada.</p> <p>A tarefa do vRO permite que você inclua uma expressão booleana que use os seguintes operandos e operadores.</p> <ul style="list-style-type: none"> <li>■ Variáveis de pipeline, como <code>\${pipeline.variableName}</code>. Use chaves somente ao inserir variáveis.</li> <li>■ Variáveis de saída da tarefa, como <code>{Stage1.task1.machines[0].value.hostIp[0]}</code>.</li> <li>■ Variáveis de associação de pipeline padrão, como <code>{releasePipelineName}</code>.</li> <li>■ Valores booleanos que não fazem distinção de maiúsculas e minúsculas, como <code>true</code>, <code>false</code>, <code>'true'</code>, <code>'false'</code>.</li> <li>■ Valores inteiros ou decimais sem aspas.</li> <li>■ Valores de cadeia de caracteres usados com aspas simples ou duplas, como <code>"test"</code>, <code>'test'</code>.</li> <li>■ Tipos numéricos e de cadeia de caracteres de valores, como <code>==</code>, <code>Equals</code> e <code>!= Not Equals</code>.</li> <li>■ Operadores relacionais, como <code>&gt;</code>, <code>&gt;=</code>, <code>&lt;</code> e <code>&lt;=</code>.</li> <li>■ Lógica booleana, como <code>&amp;&amp;</code> e <code>  </code>.</li> <li>■ Operadores aritméticos, como <code>+</code>, <code>-</code>, <code>*</code> e <code>/</code>.</li> <li>■ Expressões aninhadas usando parênteses.</li> <li>■ As cadeias de caracteres que incluem o valor literal ABCD são avaliadas como falso e a tarefa é ignorada.</li> <li>■ Não há suporte para operadores unários.</li> </ul> <p>Um exemplo de condição pode ser <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>
Sempre	Se você selecionar <b>Sempre</b> , o pipeline executará a tarefa sem condições.

- g Digite uma mensagem de saudação.
- h Clique em **Validar Tarefa** e corrija os erros que ocorrerem.
- 4 Salve, ative e execute o pipeline.
- 5 Após a execução do pipeline, examine os resultados.
- Clique em **Execuções**.
  - Clique no pipeline.
  - Clique na tarefa.
  - Examine os resultados, o valor de entrada e as propriedades.

É possível identificar o ID de execução do fluxo de trabalho, quem respondeu à tarefa e quando, bem como quaisquer comentários incluídos.

## Resultados

Parabéns! Você marcou um fluxo de trabalho do vRealize Orchestrator para uso no Code Stream e adicionou uma tarefa do vRO no pipeline do Code Stream para que ele execute um fluxo de trabalho que automatiza uma ação no seu ambiente do DevOps.

## Exemplo: Formato de saída da tarefa do vRO

O formato de saída para uma tarefa do vRO é semelhante a este exemplo.

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

## Próximo passo

Continue incluindo tarefas de fluxo de trabalho do vRO nos pipelines para poder automatizar tarefas em seus ambientes de desenvolvimento, teste e produção.

# Disparando pipelines no Code Stream

# 7

O Code Stream pode disparar um pipeline para você quando determinados eventos ocorrerem.

Por exemplo:

- O gatilho do Docker pode executar um pipeline quando um novo artefato é criado ou atualizado.
- O gatilho para Git pode acionar um pipeline quando os desenvolvedores atualizam o código.
- O gatilho para Gerrit pode acionar um pipeline quando os desenvolvedores revisam o código.

Este capítulo inclui os seguintes tópicos:

- [Como usar o gatilho do Docker no Code Stream para executar um pipeline de entrega contínua](#)
- [Como usar o gatilho Git no Code Stream para executar um pipeline](#)
- [Como usar o gatilho Gerrit no Code Stream para executar um pipeline](#)

## Como usar o gatilho do Docker no Code Stream para executar um pipeline de entrega contínua

Como administrador ou desenvolvedor do Code Stream, você pode usar o gatilho Docker no Code Stream. O gatilho do Docker executa um pipeline de entrega contínua (CD) independente sempre que um artefato de construção é criado ou atualizado. O gatilho do Docker executa o pipeline de CD, que envia o artefato novo ou atualizado como uma imagem de contêiner para um repositório do Docker Hub. O pipeline de CD pode ser executado como parte de suas compilações automatizadas.

Por exemplo, para implantar continuamente a imagem do contêiner atualizado por meio do seu pipeline de CD, use o gatilho do Docker. Quando a imagem do contêiner é verificada no registro do Docker, o webhook no Docker Hub notifica o Code Stream que a imagem foi alterada. Essa notificação aciona o pipeline de CD para ser executado com a imagem de contêiner atualizada e carrega a imagem no repositório do Docker Hub.

Para usar o gatilho do Docker, você executa várias etapas no Code Stream.

**Tabela 7-1. Como usar o gatilho do Docker**

O que fazer...	Mais informações sobre esta ação...
Criar um endpoint de registro do Docker.	<p>Para que o Code Stream dispare o pipeline, é necessário ter um endpoint de registro do Docker. Se o endpoint não existir, é possível selecionar uma opção que o crie ao adicionar o webhook para o gatilho do Docker.</p> <p>O endpoint de registro do Docker inclui o URL para o repositório do Docker Hub.</p>
Adicionar parâmetros de entrada ao pipeline que injeta parâmetros do Docker automaticamente quando o pipeline é executado.	<p>Você pode injetar parâmetros do Docker no pipeline. Os parâmetros podem incluir o nome do proprietário do evento do Docker, a imagem, o nome do repositório, o namespace do repositório e a tag.</p> <p>No seu pipeline de CD, inclua parâmetros de entrada que o webhook do Docker passa para o pipeline antes do pipeline disparar.</p>
Criar um webhook do Docker.	<p>Ao criar o webhook do Docker no Code Stream, ele também cria um webhook correspondente no Docker Hub. O webhook do Docker no Code Stream se conecta ao Docker Hub por meio do URL que você inclui no webhook.</p> <p>Os webhooks se comunicam uns com os outros e disparam o pipeline quando um artefato é criado ou atualizado no Docker Hub.</p> <p>Se você atualizar ou excluir o webhook do Docker no Code Stream, o webhook no Docker Hub também será atualizado ou excluído.</p>
Adicionar e configurar uma tarefa do Kubernetes no pipeline.	<p>Quando um artefato é criado ou atualizado no repositório do Docker Hub, o pipeline é acionado. Em seguida, ele implanta o artefato pelo pipeline no host do Docker no seu cluster do Kubernetes.</p>
Incluir uma definição de YAML local na tarefa.	<p>A definição de YAML que você aplica à tarefa de implantação inclui a imagem de contêiner do Docker. Se você precisar baixar uma imagem de um repositório de propriedade privada, o arquivo YAML deverá incluir uma seção com o Segredo de configuração do Docker. Consulte a parte sobre CD de <a href="#">Como planejar uma compilação nativa de CI/CD no Code Stream antes de usar o modelo de pipeline inteligente</a></p>

Quando um artefato é criado ou atualizado no repositório do Docker Hub, o webhook no Docker Hub notifica o webhook no Code Stream, que dispara o pipeline. As seguintes ações ocorrem:

- 1 O Docker Hub envia uma solicitação POST para o URL no webhook.
- 2 O Code Stream executa o gatilho do Docker.
- 3 O gatilho do Docker inicia seu pipeline de CD.
- 4 O pipeline de CD envia o artefato para o repositório do Docker Hub.



- 5 O Code Stream dispara o webhook do Docker, que executa um pipeline de CD que implanta o artefato no host do Docker.

Neste exemplo, é criado um endpoint do Docker e um webhook do Docker no Code Stream que implanta seu aplicativo no cluster do Kubernetes de desenvolvimento. As etapas incluem o código de exemplo para o payload que o Docker publica para o URL no webhook, o código de API que ele usa e o código de autenticação com o token seguro.

#### Pré-requisitos

- Verifique se existe um pipeline de entrega contínua (CD) na instância do Code Stream. Verifique também se ele inclui uma ou mais tarefas do Kubernetes que implantam seu aplicativo. Consulte [Capítulo 4 Planejamento para compilar, integrar e entregar seu código de forma nativa no Code Stream](#).
- Verifique se é possível acessar um cluster do Kubernetes existente no qual o seu pipeline de CD pode implantar seu aplicativo para desenvolvimento.
- Verifique se você é membro de um projeto no Code Stream. Se não for, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).

#### Procedimentos

- 1 Criar um endpoint de registro do Docker.
  - a Clique em **Endpoints**.
  - b Clique em **Novo Endpoint**.
  - c Comece a digitar o nome do projeto existente.
  - d Selecione o tipo como **Registro Docker**.
  - e Digite um nome relevante.
  - f Selecione o tipo de servidor como **DockerHub**.
  - g Digite o URL do repositório do Docker Hub.
  - h Insira o nome e a senha que podem acessar o repositório.

## New endpoint

Project *	<input type="text" value="Q AWS_PGProj"/>
Type *	<input type="text" value="Docker Registry"/>
Name *	<input type="text" value="dockerhub-endpoint"/>
Description	<div></div>
Mark restricted	<input type="checkbox"/> non-restricted
Server type *	<input type="text" value="DockerHub"/>
Repo URL *	<input type="text" value="https://hub.docker.com/repository/docker/automation/cs-builder"/> <input type="button" value="ACCEPT CERTIFICATE"/>
Username *	<input type="text" value="admin"/>
Password *	<input type="password" value="....."/> <input type="button" value="CREATE VARIABLE"/>

- 2 No seu pipeline de CD, defina as propriedades de entrada para injetar automaticamente os parâmetros do Docker quando o pipeline for executado.

sm-1 Enabled

Workspace **Input** Model Output

Input Parameters ⓘ

Auto inject parameters ☐ Gerrit ☐ Git ☒ Docker ☐ None

[ADD](#) [ADD/REMOVE INJECTED PARAMETERS](#)

Starred ⓘ	Name
⋮ ☆	DOCKER_EVENT_OWNER_NAME
⋮ ☆	DOCKER_IMAGE
⋮ ☆	DOCKER_REPO_NAME
⋮ ☆	DOCKER_REPO_NAMESPACE
⋮ ☆	DOCKER_TAG

- 3 Criar um webhook do Docker.
  - a Clique em **Gatilhos > Docker**.
  - b Clique em **Novo Webhook para Docker**.
  - c Selecione um projeto.
  - d Digite um nome relevante.
  - e Selecione o endpoint de registro do Docker.

Se o endpoint ainda não existir, clique em **Criar Endpoint** e crie-o.

  - f Selecione o pipeline com os parâmetros injetados do Docker para o webhook disparar. Consulte [Etapa 2](#).

Se o pipeline tiver sido configurado com os parâmetros de entrada adicionados personalizados, a lista de parâmetros de entrada exibirá parâmetros e valores. É possível digitar valores para os parâmetros de entrada que serão passados para o pipeline com o evento de gatilho. Também é possível deixar os valores em branco ou usar os valores padrão, se definidos.

Para obter mais informações sobre parâmetros na guia de entrada, consulte [Planejando uma compilação nativa de CI/CD no Code Stream antes de adicionar tarefas manualmente](#).

g Digite o Token de API.

O token de API do CSP autentica você para conexões de API externas com o Code Stream. Para obter o token de API:

- 1 Clique em **Gerar Token**.
- 2 Insira o endereço de e-mail associado ao seu nome de usuário e senha e clique em **Gerar**.

O token gerado será válido por seis meses. Ele também é conhecido como token de atualização.

- Para manter o token como uma variável para uso futuro, clique em **Criar Variável**, insira um nome para a variável e clique em **Salvar**.
- Para manter o token como um valor de texto para uso futuro, clique em **Copiar** e cole o token em um arquivo de texto para salvar localmente.

Você pode optar por criar uma variável e armazenar o token em um arquivo de texto para uso futuro.

- 3 Clique em **Fechar**.

h Insira a imagem de compilação.

- i Insira uma tag.

**Docker**

Activity **Webhooks for Docker**

Webhook URL <sup>?</sup> `https://[redacted]m/codestream/api/registry-webhook-listeners/54bd030d-`

Project `test`

Name \* `sm-1-Docker-WH`

Description `Docker webhook trigger for sm-1`

Docker Registry `Docker-Register-Endpoint`

Pipeline \* `sm-1` <sup>?</sup>

API token \* `.....` <sup>?</sup> **CREATE VARIABLE** **GENERATE TOKEN**

Image <sup>?</sup> `Image`

Tag <sup>?</sup> `Tags`

**SAVE** **CANCEL**

- j Clique em **Salvar**.

O cartão de webhook aparece com o webhook do Docker habilitado. Se você quiser fazer um envio fictício ao repositório do Docker Hub sem disparar o webhook do Docker e executar um pipeline, clique em **Desativar**.

- 4 No pipeline de CD, configure a tarefa de implantação do Kubernetes.
  - a Nas propriedades de tarefa do Kubernetes, selecione o cluster do Kubernetes de desenvolvimento.
  - b Selecione a ação **Criar**.

- c Selecione a **Definição Local** para a fonte de payload.
- d Em seguida, selecione seu arquivo YAML local.

Por exemplo, o Docker Hub pode publicar essa definição de YAML local como o payload para o URL no webhook:

```
{
  "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#\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\n\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"
  }
}
```

A API que cria o webhook no Docker Hub

usa o formulário: [https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook\\_pipeline/](https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/)

O corpo do código JSON é semelhante a:

```
{
  "name": "demo_webhook",
```

```
"webhooks": [
{
"name": "demo_webhook",
"hook_url": "http://www.google.com"
}
]
}
```

Para receber eventos do servidor Docker Hub, o esquema de autenticação do webhooker do Docker que você criar no Code Stream usará um mecanismo de autenticação de lista de permissões com um token de cadeia de caracteres aleatório no webhook. Ele filtra eventos com base no token seguro, que pode ser anexado a `hook_url`.

O Code Stream pode verificar qualquer solicitação do servidor Docker Hub usando o token seguro configurado. Por exemplo: `hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""`

- 5 Crie um artefato do Docker no repositório do Docker Hub. Ou atualize um artefato existente.
- 6 Para confirmar que o disparo ocorreu e ver a atividade no webhook do Docker, clique em **Gatilhos > Docker > Atividade**.

### Docker

GUIDED SETUP

Activity
Webhooks for Docker

Commit Time	Webhook	Image	Tag	Owner	Repository	Pipeline	Execution Status
01/09/2019 10:59 AM	dt11-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	fvxd-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	test-do-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	sm-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	t-token-Docker-WH	admin/repo:s1	s1	admin	repo		FAILED
01/09/2019 10:57 AM	dt11-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	sm-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	test-do-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	fvxd-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED

- 7 Clique em **Execuções** e observe seu pipeline enquanto ele é executado.

### Executions

417 items
GUIDED SETUP

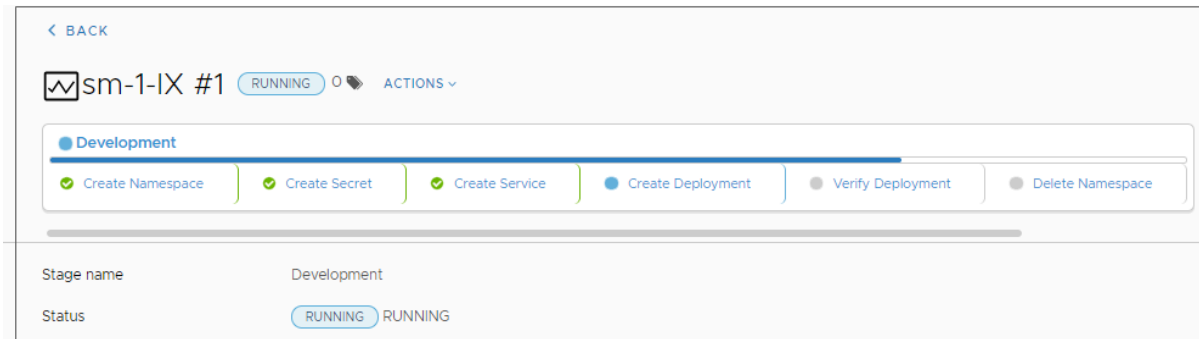
+ NEW EXECUTION
🔍
📄
🔍
🔄

sm-1-IX#1

RUNNING
Stages: 
ACTIONS ▾

By k on 01/09/2019 2:41 PM
☆ Input : n/a
☆ Output : n/a

- 8 Clique no estágio de execução e exiba as tarefas à medida que o pipeline é executado.



## Resultados

Parabéns! Configure o gatilho do Docker para executar o pipeline de CD continuamente. Seu pipeline agora pode carregar artefatos novos e atualizados do Docker para o repositório Docker Hub.

## Próximo passo

Verifique se o artefato novo ou atualizado está implantado no host do Docker no cluster do Kubernetes de desenvolvimento.

# Como usar o gatilho Git no Code Stream para executar um pipeline

Como administrador ou desenvolvedor do Code Stream, você pode integrar o Code Stream ao ciclo de vida do Git usando o gatilho Git. Quando você faz uma alteração de código no GitHub, GitLab ou Bitbucket Enterprise, o evento se comunica com o Code Stream por meio de um webhook e dispara um pipeline. O webhook funciona com versões empresariais locais do GitLab, do GitHub e Bitbucket quando o Cloud Assembly e a versão empresarial estão ambos acessíveis na mesma rede.

Ao adicionar o webhook para Git no Code Stream, ele também cria um webhook no repositório do GitHub, GitLab ou Bitbucket. Se você atualizar ou excluir o webhook posteriormente, essa ação também atualizará ou excluirá esse webhook no GitHub, GitLab ou Bitbucket.

Sua definição de webhook deve incluir um endpoint do Git na ramificação do repositório que será monitorado. Para criar o webhook, o Code Stream usa o endpoint do Git. Se o endpoint não existir, você poderá criá-lo ao adicionar o webhook. Este exemplo pressupõe que você tenha um endpoint predefinido do Git no GitHub.

**Observação** "Como criar um webhook, seu endpoint do Git deve usar um token privado para autenticação, mas ele não pode usar uma senha."



Você pode criar vários webhooks para ramificações diferentes usando o mesmo endpoint Git e fornecendo diferentes valores para o nome da ramificação na página de configuração do webhook. Para criar outro webhook para outra ramificação no mesmo repositório Git, não é necessário clonar o endpoint Git várias vezes para várias ramificações. Em vez disso, forneça o nome da ramificação no webhook, o que permite reutilizar o endpoint Git. Se a ramificação no webhook Git for a mesma que a ramificação no endpoint, você não precisará fornecer o nome da ramificação na página do webhook Git.

Este exemplo mostra como usar o gatilho Git com um repositório GitHub, mas os pré-requisitos incluem as preparações necessárias se outro tipo de servidor Git for usado.

#### Pré-requisitos

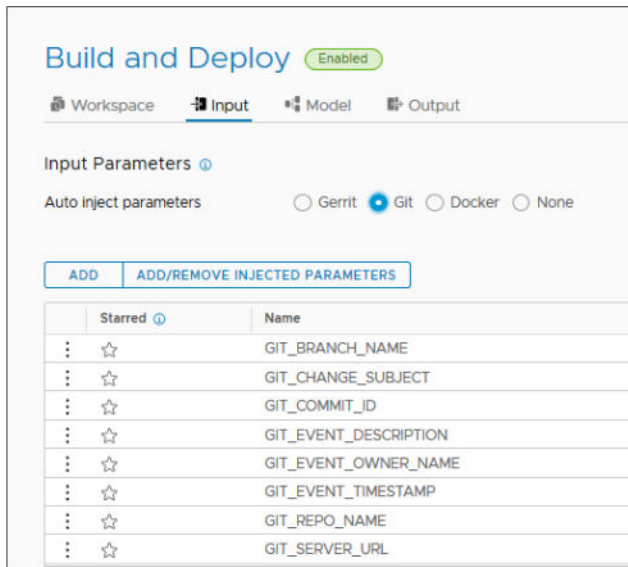
- Verifique se você é membro de um projeto no Code Stream. Se não for, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).
- Verifique a existência de um endpoint do Git na ramificação do GitHub que deseja monitorar. Consulte [Como integrar o Code Stream ao Git](#).
- Verifique se você tem direitos para criar um webhook no repositório Git.
- Se estiver configurando um webhook no GitLab, altere as configurações de rede padrão no GitLab Enterprise para habilitar solicitações de saída e permitir a criação de webhooks locais.

---

**Observação** Essa alteração só é necessária para o GitLab Enterprise. Essas configurações não se aplicam ao GitHub ou Bitbucket.

---

- a Faça login na instância do GitLab Enterprise como administrador.
- b Acesse as configurações de rede usando uma URL, como `http://{gitlab-server}/admin/application_settings/network`.
- c Expanda **Solicitações de saída** e clique em:
  - Permita solicitações para a rede local a partir de webhooks e serviços Web.
  - Permita solicitações para a rede local a partir de um gancho de sistema.
- Para os pipelines que você deseja disparar, verifique se você definiu as propriedades de entrada para injetar parâmetros Git quando o pipeline for executado.



Para obter informações sobre os parâmetros de entrada, consulte [Planejando uma compilação nativa de CI/CD no Code Stream antes de adicionar tarefas manualmente](#).

## Procedimentos

- 1 No Code Stream, clique em **Gatilhos > Git**.
- 2 Clique na guia **Webhooks para Git**, depois clique em **Novo Webhook para Git**.
  - a Selecione um projeto.
  - b Digite um nome significativo e uma descrição para o webhook.

- c Selecione um endpoint Git configurado para a ramificação que você deseja monitorar.

Quando você cria seu webhook, a definição do webhook inclui os detalhes atuais do endpoint.

- Mais tarde, se você alterar o tipo Git, o tipo de servidor Git ou a URL do repositório Git no endpoint, o webhook não poderá mais disparar um pipeline, pois ele tentará acessar o repositório Git usando os detalhes do endpoint original. Você deverá excluir o webhook e criá-lo novamente com o endpoint.
- Mais tarde, se você alterar o tipo de autenticação, o nome do usuário ou o token privado no endpoint, o webhook continuará a funcionar.
- Se você estiver usando um repositório do BitBucket, a URL do repositório deverá estar em um dos seguintes formatos: `https://api.bitbucket.org/{user}/{repo name}` ou `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`.

---

**Observação** Se você tiver criado anteriormente um webhook usando um endpoint Git que usa uma senha para autenticação básica, deverá excluir e redefinir esse webhook com um endpoint Git que usa um token privado para autenticação.

---

Consulte [Como integrar o Code Stream ao Git](#).

- d (Opcional) Insira a ramificação que você deseja que o webhook monitore.

Se você deixar a ramificação não especificada, o webhook monitorará a ramificação que você configurou para o endpoint do Git.

- e (Opcional) Gere um token secreto para o webhook.

Se você usar um token secreto, o Code Stream gerará um token de string aleatório para o webhook. Depois, quando o webhook recebe os dados de evento do Git, ele os envia com o token secreto. O Code Stream usa as informações para determinar se as chamadas são provenientes da origem esperada, como a instância do GitHub, o repositório e a ramificação configurados. O token secreto fornece uma camada extra de segurança que é usada para verificar se os dados de evento do Git são provenientes da origem correta.

f (Opcional) Forneça inclusões de arquivo ou exclusões como condições para o gatilho.

- **Inclusões de arquivos.** Se qualquer um dos arquivos em uma confirmação corresponder aos arquivos especificados nos caminhos de inclusão ou no regex, a confirmação vai disparar os pipelines. Quando um regex especificado, o Code Stream apenas dispara os pipelines quando os nomes de arquivo no conjunto de alterações correspondem à expressão fornecida. O filtro Regex é útil ao configurar um gatilho para vários pipelines em um único repositório.
- **Exclusões de arquivos.** Quando todos os arquivos em uma confirmação correspondem aos arquivos especificados nos caminhos de exclusão ou no Regex, os pipelines não são disparados.
- **Priorizar exclusões.** Quando ativada, a opção Priorizar Exclusão garante que os pipelines não sejam disparados, mesmo que qualquer um dos arquivos em uma confirmação corresponda aos arquivos especificados nos caminhos de exclusão ou no regex. A configuração padrão é desativada.

Se as condições atenderem tanto às inclusões quanto às exclusões de arquivos, os pipelines não serão disparados.

No exemplo a seguir, as inclusões e as exclusões de arquivo são condições para o gatilho.

The screenshot shows a configuration window titled "File" with a help icon. It contains two main sections: "Inclusions" and "Exclusions".

Section	Filter Type	Path/Regex	Actions
Inclusions	PLAIN	runtime/src/main/a.java	-
	REGEX	([a-z A-Z]+[/])[a-z A-Z]+	- +
Exclusions	PLAIN	runtime/pom.xml	-
	PLAIN	runtime/demo.yaml	- +

At the bottom, there is a "Prioritize Exclusion" toggle switch, which is currently turned off.

- Para inclusões de arquivo, uma confirmação com qualquer alteração em `runtime/src/main/a.java` ou qualquer arquivo Java disparará pipelines configurados na configuração de eventos.
  - Para exclusões de arquivos, uma confirmação com alterações somente em ambos os arquivos não disparará os pipelines configurados nas configurações de evento.
- g Para o evento do Git, selecione uma solicitação **Push** ou **Pull**.

## h Digite o Token de API.

O token de API do CSP autentica você para conexões de API externas com o Code Stream. Para obter o token de API:

1 Clique em **Gerar Token**.2 Insira o endereço de e-mail associado ao seu nome de usuário e senha e clique em **Gerar**.

O token gerado será válido por seis meses. Ele também é conhecido como token de atualização.

- Para manter o token como uma variável para uso futuro, clique em **Criar Variável**, insira um nome para a variável e clique em **Salvar**.
- Para manter o token como um valor de texto para uso futuro, clique em **Copiar** e cole o token em um arquivo de texto para salvar localmente.

Você pode optar por criar uma variável e armazenar o token em um arquivo de texto para uso futuro.

3 Clique em **Fechar**.

## i Selecione o pipeline para o webhook disparar.

Se o pipeline incluir parâmetros de entrada personalizados adicionados, a lista Parâmetros de Entrada exibirá parâmetros e valores. É possível digitar valores para os parâmetros de entrada que são passados para o pipeline com o evento de gatilho. Também é possível deixar os valores em branco ou usar os valores padrão, se definidos.

Para obter informações sobre a inserção automática de parâmetros de entrada para gatilhos do Git, consulte [Pré-requisitos](#).

j Clique em **Criar**.

O webhook aparece como um novo cartão.

## 3 Clique no cartão de webhook.

Quando o formulário de dados do webhook reaparecer, você verá um URL do webhook adicionado à parte superior do formulário. O Git webhook conecta-se ao repositório do GitHub por meio do URL do webhook.

# Git

Activity

Webhooks for Git

Webhook URL ⓘ

Project

Name \*

Description

Endpoint

Branch ⓘ

Secret token ⓘ \*

File ⓘ

Inclusions

Exclusions

Prioritize Exclusion

Trigger

For Git

API token \*

Pipeline \*

Comments

Execution trigger delay ⓘ

https://ca8001b1-414c-4389-8000-000000000000/codestream/api/git-webhook-listeners/963b2287-527f-4e9b-b000-000000000000

test

test-webhook

Description

DemoApp-Git

master

GYH0cBWZx4dUn47Y/KA8H/BOkts=

GENERATE

--Select-- ▾ Value +

--Select-- ▾ Value +

☐

☒ PUSH ☐ PULL REQUEST

.....

CREATE VARIABLE

GENERATE TOKEN

CICD-2

×

1

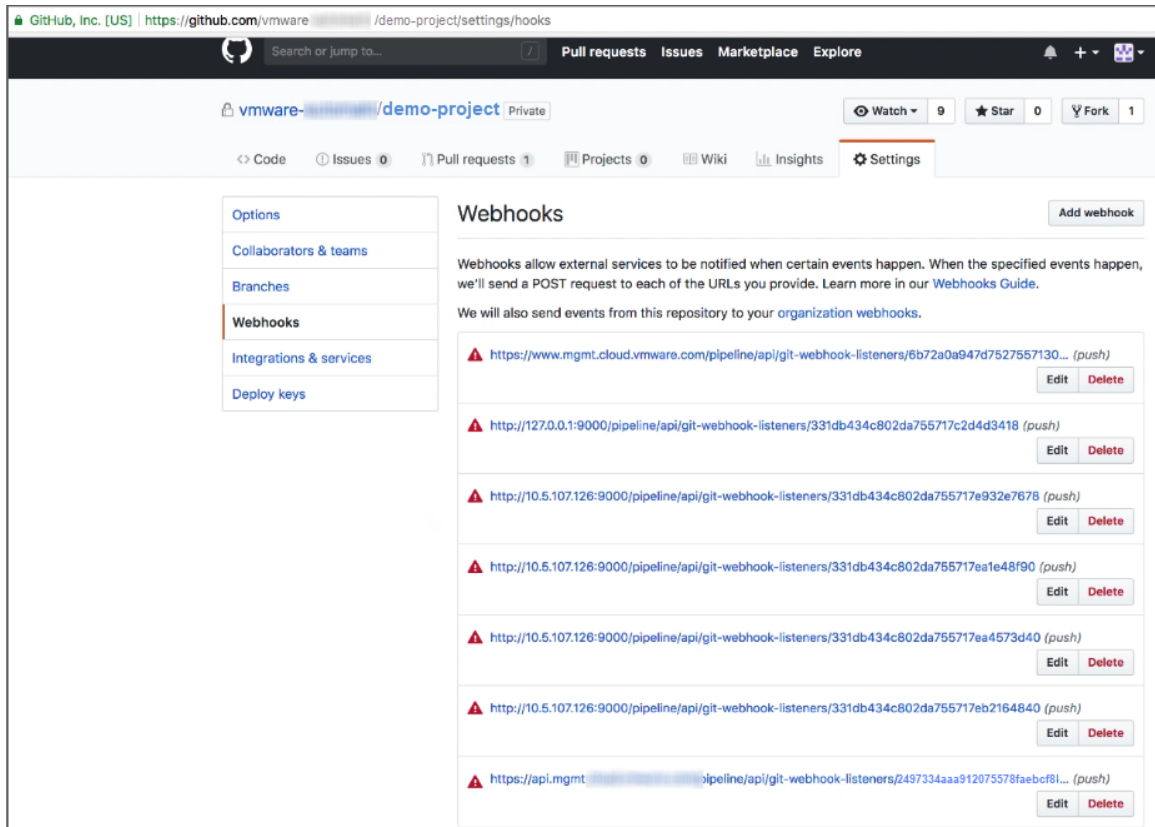
⌵

SAVE

CANCEL

- 4 Em uma nova janela do navegador, abra o repositório GitHub que se conecta por meio do webhook.
  - a Para ver o webhook adicionado por você no Code Stream, clique na guia **Configurações** e selecione **Webhooks**.

No final da lista de webhooks, você verá o mesmo URL do webhook.



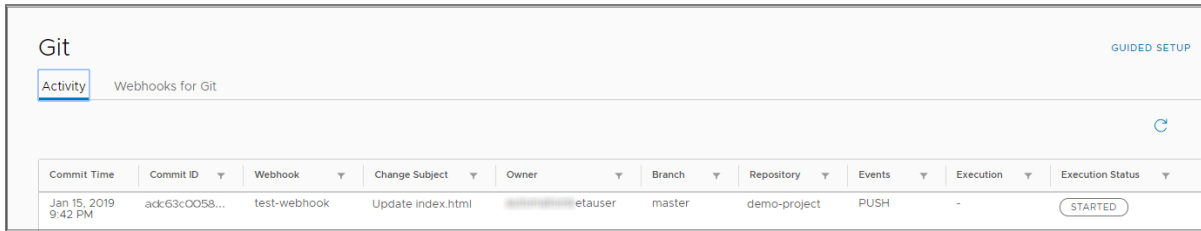
- b Para fazer uma alteração no código, clique na guia **Código** e selecione um arquivo na ramificação. Depois de editar o arquivo, confirme a alteração.
  - c Para verificar se o URL do webhook está funcionando, clique na guia **Configurações** e selecione **Webhooks** novamente.

"Na parte inferior da lista de webhooks, uma marca de seleção verde aparece ao lado da URL do webhook."



- 5 Volte para o Code Stream para visualizar a atividade no Git webhook. Clique em **Gatilhos > Git > Atividade**.

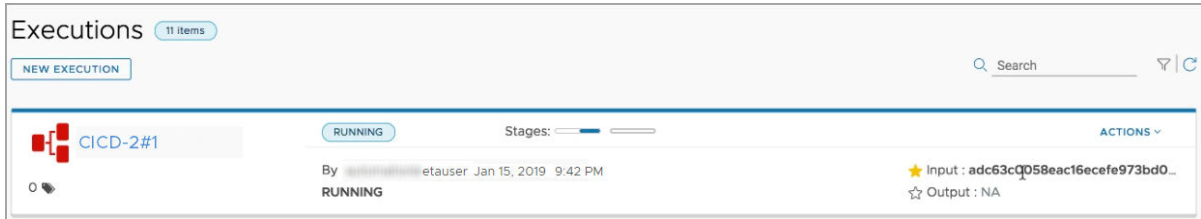
Em Status de Execução, verifique se a execução do pipeline foi iniciada.



Commit Time	Commit ID	Webhook	Change Subject	Owner	Branch	Repository	Events	Execution	Execution Status
Jan 15, 2019 9:42 PM	ack63c0058...	test-webhook	Update index.html	etauser	master	demo-project	PUSH	-	STARTED

6 Clique em **Execuções** e acompanhe seu pipeline à medida que ele é executado.

Para observar a execução do pipeline, você pode pressionar Atualizar.



Execution Name	Status	Stages	By	Time	Input	Output
CICD-2#1	RUNNING	0 / 1	etauser	Jan 15, 2019 9:42 PM	adc63c0058ac16ecef973bd0...	NA

## Resultados

Parabéns! Você usou com sucesso o gatilho para Git.

## Como usar o gatilho Gerrit no Code Stream para executar um pipeline

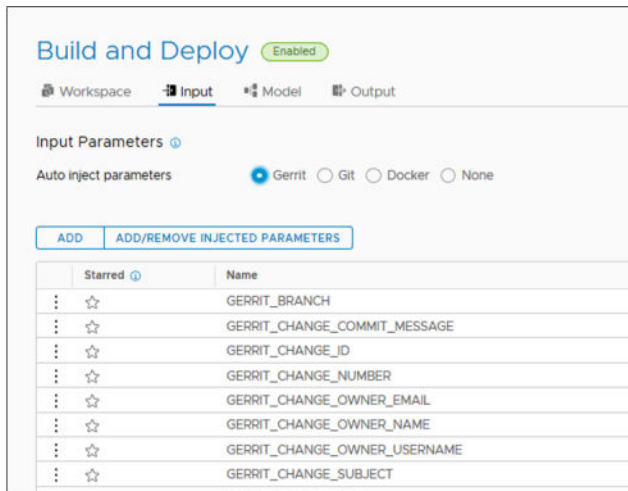
Como um administrador ou desenvolvedor do Code Stream, você pode integrar o Code Stream com o ciclo de vida de revisão de código Gerrit usando o gatilho Gerrit. O evento aciona um pipeline a ser executado quando você cria um conjunto de patches, publica rascunhos, mescla alterações de código no projeto Gerrit ou envia alterações diretamente na ramificação do Git.

Ao adicionar o gatilho Gerrit, você seleciona um ouvinte Gerrit, um projeto Gerrit no servidor Gerrit e configura eventos Gerrit. Neste exemplo, você configura primeiro um ouvinte Gerrit e, em seguida, usa esse ouvinte em um gatilho do Gerrit com dois eventos em três pipelines diferentes.

### Pré-requisitos

- Verifique se você é membro de um projeto no Code Stream. Se não for, peça a um administrador do Code Stream para adicioná-lo como membro de um projeto. Consulte [Como adicionar um projeto no Code Stream](#).
- Verifique se você tem um endpoint do Gerrit configurado no Code Stream. Consulte [Como integrar o Code Stream ao Gerrit](#).
- Você deve saber qual é a sua versão de lançamento do Gerrit.
- Para que os pipelines sejam disparados, verifique se você definiu as propriedades de entrada do pipeline como **Gerrit**, o que permite que o pipeline receba os parâmetros do Gerrit como entradas ao ser executado.





Para obter informações sobre os parâmetros de entrada, consulte [Planejando uma compilação nativa de CI/CD no Code Stream antes de adicionar tarefas manualmente](#).

## Procedimentos

- 1 No Code Stream, clique em **Gatilhos > Gerrit**.
- 2 (Opcional) Clique na guia **Ouvintes** e depois em **Novo Ouvinte**.

---

**Observação** Se o ouvinte Gerrit que você planeja usar para o gatilho Gerrit já estiver definido, pule esta etapa.

---

- a Selecione um projeto.
- b Insira um nome para o ouvinte Gerrit.
- c Selecione um endpoint do Gerrit.

- d Digite o Token de API.

O token de API do CSP autentica você para conexões de API externas com o Code Stream. Para obter o token de API:

- 1 Clique em **Gerar Token**.
- 2 Insira o endereço de e-mail associado ao seu nome de usuário e senha e clique em **Gerar**.

O token gerado será válido por seis meses. Ele também é conhecido como token de atualização.

- Para manter o token como uma variável para uso futuro, clique em **Criar Variável**, insira um nome para a variável e clique em **Salvar**.
- Para manter o token como um valor de texto para uso futuro, clique em **Copiar** e cole o token em um arquivo de texto para salvar localmente.

Você pode optar por criar uma variável e armazenar o token em um arquivo de texto para uso futuro.

- 3 Clique em **Fechar**.

Se você criou uma variável, o token de API exibirá o nome da variável que você inseriu usando uma associação com cifrão. Se você tiver copiado o token, o token de API exibirá o token mascarado.

The screenshot shows the 'Gerrit' configuration interface with the 'Listeners' tab selected. The form contains the following fields and controls:

- Project**: A text field containing 'test1' with a clear button (X).
- Name**: A text field containing 'Gerrit-Demo-Listener'.
- Endpoint**: A dropdown menu showing 'corporate-gerrit'.
- API token**: A text field containing the variable reference '\$var.CSuser API Token' with a green checkmark icon.
- Buttons**:
  - 'CREATE VARIABLE' and 'GENERATE TOKEN' are light blue buttons located to the right of the API token field.
  - 'CREATE', 'VALIDATE', and 'CANCEL' are blue buttons located at the bottom left of the form.

- e Para validar os detalhes do token e do endpoint, clique em **Validar**.

Seu token expira após 90 dias.

- f Clique em **Criar**.
- g No cartão do ouvinte, clique em **Conectar**.

O ouvinte começa a monitorar todas as atividades no servidor Gerrit e escuta todos os gatilhos habilitados nesse servidor. Para parar de escutar um gatilho nesse servidor. Desative o gatilho.

---

**Observação** Para atualizar um endpoint do Gerrit que está conectado a um ouvinte, você deve desconectar o ouvinte antes de atualizar o endpoint.

- Clique em **Configurar > Gatilhos > Gerrit**.
  - Clique na guia **Ouvintes**.
  - Clique em **Desconectar** no ouvinte que está conectado ao endpoint que você deseja atualizar.
- 

- 3 Clique na guia **Gatilhos** e depois em **Novo Gatilho**.

- 4 Selecione um projeto no servidor Gerrit.

- 5 Digite um nome.

O nome do gatilho Gerrit deve ser exclusivo.

- 6 Selecione um ouvinte Gerrit configurado.

Usando o ouvinte Gerrit, o Code Stream fornece uma lista de projetos Gerrit que estão disponíveis no servidor.

- 7 Selecione um projeto no servidor Gerrit.

- 8 Insira a ramificação no repositório que o ouvinte Gerrit monitorará.

- 9 (Opcional) Forneça inclusões de arquivo ou exclusões como condições para o gatilho.

- Você fornece inclusões de arquivo que disparam os pipelines. Quando qualquer um dos arquivos em uma confirmação corresponde aos arquivos especificados nos caminhos de inclusão ou no Regex, os pipelines são disparados. Com um Regex especificado, o Code Stream apenas disparará os pipelines com nomes de arquivo no conjunto de alterações que corresponderem à expressão fornecida. O filtro Regex é útil ao configurar um gatilho para vários pipelines em um único repositório.
- Você fornece exclusões de arquivos que evitam o disparo dos pipelines. Quando todos os arquivos em uma confirmação correspondem aos arquivos especificados nos caminhos de exclusão ou no Regex, os pipelines não são disparados.
- Quando ativada, a opção **Priorizar Exclusão** garante que os pipelines não sejam acionados. Os pipelines não serão disparados, mesmo que qualquer um dos arquivos em uma confirmação corresponda aos arquivos especificados nos caminhos de exclusão ou no regex. A configuração padrão para **Priorizar Exclusão** é Desativado.

Se as condições atenderem à inclusão e à exclusão de arquivos, os pipelines não serão disparados.

No exemplo a seguir, tanto as inclusões quanto as exclusões de arquivos são condições para o gatilho.

File ⓘ

Inclusions

PLAIN

runtime/src/main/a.java

-

REGEX

([a-z A-Z]+/[a-z A-Z])+

-

+

Exclusions

PLAIN

runtime/pom.xml

-

PLAIN

runtime/demo.yaml

-

+

Prioritize Exclusion
☐

- Para inclusões de arquivo, uma confirmação que tiver qualquer alteração em `runtime/src/main/a.java` ou qualquer arquivo Java disparará pipelines definidos na configuração de eventos.
- Para exclusões de arquivos, uma confirmação que tiver alterações somente em ambos os arquivos não disparará os pipelines definidos na configuração de eventos.

## 10 Clique em **Nova Configuração**.

- a Para um evento Gerrit, selecione **Conjunto de Patches Criado**, **Rascunho Publicado** ou **Alteração Mesclada**. Ou, para um push direto para o Git que ignore o Gerrit, selecione **Push Git direito**.

---

**Observação** A partir da versão de lançamento 2.15 do Gerrit, não há mais suporte para alterações de rascunho e conjuntos de alterações de rascunho. Portanto, se você tem a versão de lançamento 2.15 ou mais recente do Gerrit, **Rascunho Publicado** não é um evento permitido.

---

- b Selecione o pipeline que será disparado.

Se o pipeline incluir parâmetros de entrada personalizados adicionados, a lista Parâmetros de Entrada exibirá parâmetros e valores. É possível inserir valores para os parâmetros de entrada que serão transmitidos ao pipeline com o evento de gatilho. Também é possível deixar os valores em branco ou usar os valores padrão.

---

**Observação** Se valores padrão estiverem definidos:

- Qualquer valor inserido para os parâmetros de entrada substituirá os valores padrão definidos no modelo de pipeline.
  - Os valores padrão na configuração do gatilho não mudarão se os valores dos parâmetros no modelo de pipeline mudarem.
- 

Para obter informações sobre a inserção automática de parâmetros de entrada para gatilhos do Gerrit, consulte [Pré-requisitos](#).

- c Para **Conjunto de Patches Criado**, **Rascunho Publicado** e **Alterar Mesclados**, algumas ações aparecem com rótulos por padrão. Você pode alterar o rótulo ou adicionar comentários. Em seguida, quando o pipeline for executado, o rótulo ou comentário aparecerá na guia **Atividade** como a **Ação realizada** para o pipeline.

A configuração do Evento do Gerrit permite que você insira comentários usando uma variável para o comentário de Êxito ou Falha. Por exemplo, `${var.success}` e `${var.failure}`.

- d Clique em **Salvar**.

Para adicionar vários eventos de gatilho em vários pipelines, clique em **Nova Configuração** novamente.

No exemplo a seguir, é possível ver os eventos de três pipelines:

- Se um evento de **Alteração Mesclada** ocorrer no projeto Gerrit, o pipeline **Gerrit-Pipeline** será disparado.
- Se um evento de **Conjunto de Patches Criado** ocorrer no projeto Gerrit, os pipelines **Gerrit-Trigger-Pipeline** e **Gerrit-Demo-Pipeline** serão disparados.

# Gerrit

GUIDED SETUP

Activity
**Triggers**
Listeners

Project \*
test1

Name \*
Gerrit-Demo-Trigger

Gerrit Listener \*
Gerrit-Demo-Listener

Gerrit project \*
Gerrit-Demo-Project

Branch \*
master

File ⓘ

Inclusions
-- Select Type --
value

Exclusions
-- Select Type --
value

Prioritize Exclusion
☐

+ NEW CONFIGURATION

	Event Type	Pipeline	Label
⋮	Change Merged	Gerrit-Pipeline	Verified
⋮	Patchset Created	Gerrit-Trigger-Pipeline	Verified
⋮	Patchset Created	Gerrit-Demo-Pipeline	Verified

3 configurations

11 Clique em **Criar**.

O gatilho Gerrit aparece como um novo cartão na guia **Gatilhos** e é definido como **Desativado** por padrão.

12 No cartão de gatilho, clique em **Ativar**.

Depois que você ativar o gatilho, ele pode usar o ouvinte Gerrit, que começa a monitorar eventos que ocorrem na ramificação do projeto Gerrit.

Para criar um gatilho que tenha as mesmas condições de inclusão e de exclusão de arquivos, mas com um repositório diferente daquele que você incluiu quando criou o gatilho, clique no cartão de gatilho **Ações > Clonar**. Em seguida, no gatilho clonado, clique em **Abrir** e altere os parâmetros.

## Resultados

Parabéns! Você configurou com êxito um gatilho Gerrit com dois eventos em três pipelines diferentes.

## Próximo passo

Depois de confirmar uma alteração de código no projeto Gerrit, observe a guia **Atividade** do evento Gerrit no Code Stream. Verifique se a lista de atividades inclui entradas que correspondem a cada execução de pipeline na configuração do gatilho.

Quando um evento ocorrer, apenas os pipelines no gatilho Gerrit relacionados ao tipo específico de evento poderão ser executados. Neste exemplo, se um conjunto de patches for criado, somente o **Gerrit-Gatilho-Pipeline** e o **Gerrit-Demo-Pipeline** serão executados.

As informações nas colunas da guia **Atividade** descrevem cada evento do gatilho Gerrit. Você pode selecionar as colunas exibidas clicando no ícone da coluna que aparece abaixo da tabela.

- As colunas **Alterar Assunto** e **Execução** estarão vazias quando o disparador tiver sido um push Git direto.
- A coluna **Gatilho Gerrit** mostra o gatilho que criou o evento.
- A coluna **Ouvinte** está desativada por padrão. Quando selecionada, ela exibe o ouvinte Gerrit que recebeu o evento. Um único ouvinte pode aparecer como associado a vários gatilhos.
- A coluna **Tipo de Gatilho** está desativada por padrão. Quando selecionada, ela exibe o tipo de gatilho como AUTOMÁTICO ou MANUAL.
- Outras colunas incluem **Hora da Confirmação**, **Nº Alteração**, **Status**, **Mensagem**, **Ação realizada**, **Usuário**, **Projeto Gerrit**, **Ramificação** e **Evento**.

**Gerrit** GUIDED SETUP

Activity Triggers Listeners

TRIGGER MANUALLY

	Commit Time	Change#	Change Subject	Execution	Status	Message	Action taken	User	Gerrit project	Gerrit Trigger	Branch	Event
⋮	Nov 12, 2019, 12:47:53 PM	19570 /4	1111Dummy	Gerrit-Pipeline #1	COMPLETED	Execution Completed.	Verified +1	Drivank dpryan@gm	test1	Gerrit-Demo-Trigger	master	Change Merged
⋮	Nov 12, 2019, 12:50:04 PM	19570 /6	11111Dummy	Gerrit-Pipeline #2	WAITING	Stage0.Task0: Execution Waiting for User Action.		Drivank dpryan@gm	test1	Gerrit-Demo-Trigger	master	Change Merged
			11111Dummy	Gerrit-Demo-Pipeline #1	FAILED	Stage0.Task0: User Operation request has been rejected by Fritz.	Verified -1	Drivank dpryan@gm	test1	Gerrit-Demo-Trigger	master	Patchset created
			11111Dummy	Gerrit-Trigger-Pipeline #1	WAITING	Stage0.Task0: Execution Waiting for User Action.		Drivank dpryan@gm	test1	Gerrit-Demo-Trigger	master	Patchset created

Show columns

- ☐ Change
- ☒ Change Subject
- ☒ Execution
- ☒ Status
- ☒ Message
- ☒ Action taken
- ☒ User
- ☒ Gerrit project
- ☒ Gerrit Trigger
- ☐ Listener
- ☒ Branch
- ☒ Event
- ☐ Trigger Type

SELECT ALL

Items per page 20 1 - 4 of 4 items

Para controlar a atividade para uma execução de pipeline concluída ou que falhou, clique nos três pontos à esquerda de qualquer entrada na tela Atividade.

- Se a execução do pipeline falhar devido a um erro no modelo de pipeline ou outro problema, corrija o erro e selecione **Executar novamente**, o que repete a execução do pipeline.
- Se a execução do pipeline falhar devido a um problema de conectividade de rede ou outro problema, selecione **Retomar**, o que reinicia a mesma execução de pipeline e economiza tempo de execução.
- Use **Visualizar Execução**, o que abre a visualização de execução do pipeline. Consulte [Como executar um pipeline e ver os resultados](#).
- Use **Excluir** para excluir a entrada da tela Atividade.

Se um evento Gerrit falhar em disparar um pipeline, você poderá clicar em **Disparar Manualmente** e, em seguida, selecionar o gatilho Gerrit, inserir o ID da Alteração e clicar em **Executar**.



# Monitorando pipelines no Code Stream



Como administrador ou desenvolvedor do Code Stream, você precisa de informações sobre o desempenho dos pipelines no Code Stream. É necessário saber com que eficácia os pipelines liberam o código desde o desenvolvimento, passando por testes, até a produção.

Para obter insights, você usa painéis do Code Stream para monitorar as tendências e resultados de uma execução de pipeline. É possível usar os painéis de pipeline padrão para monitorar um único pipeline ou criar painéis personalizados para monitorar vários pipelines.

- As métricas do pipeline incluem estatísticas, como tempos médios, que estão disponíveis no painel de pipeline.
- Para ver as métricas em vários pipelines, use os painéis personalizados.

Este capítulo inclui os seguintes tópicos:

- [O que o painel do pipeline está mostrando no Code Stream](#)
- [Como usar painéis personalizados para rastrear indicadores-chave de desempenho para o meu pipeline no Code Stream](#)

## O que o painel do pipeline está mostrando no Code Stream

Um painel de pipeline é uma visualização dos resultados de um pipeline específico que foi executado, como tendências, principais falhas e alterações bem-sucedidas. O Code Stream cria o painel de pipeline quando você cria um pipeline.

O painel contém os widgets que exibem os resultados de execuções de pipelines.

### Widget de Contagens de Status de Execução de Pipeline

Você pode visualizar o número total de execuções de um pipeline em um período de tempo agrupado por status: Concluído, Com Falha ou Cancelado. Para ver como o status de execução do pipeline mudou em períodos mais longos ou mais curtos, altere a duração na tela.

### Widget de Estatísticas de Execução de Pipeline

As estatísticas de execuções de pipelines incluem os tempos médios de recuperação, entrega ou falha de um pipeline ao longo do tempo.

Os seguintes estados aplicam-se a todas as execuções de pipeline:

- Concluído
- Falhou
- Aguardando
- Em execução
- Cancelado
- Em Fila
- Não Iniciado
- Revertendo
- Reversão Concluído
- Falha na Reversão
- Em Pausa

**Tabela 8-1. Medição de tempos médios**

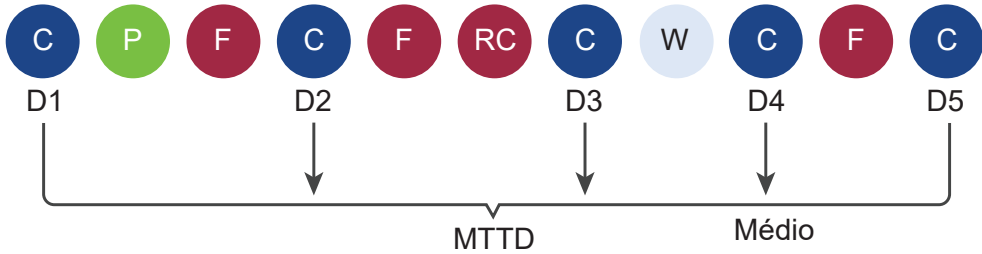
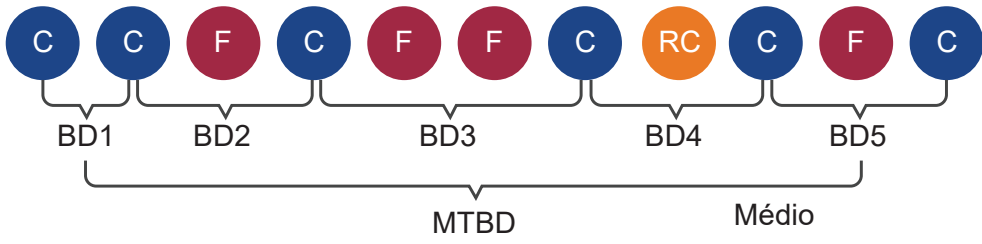
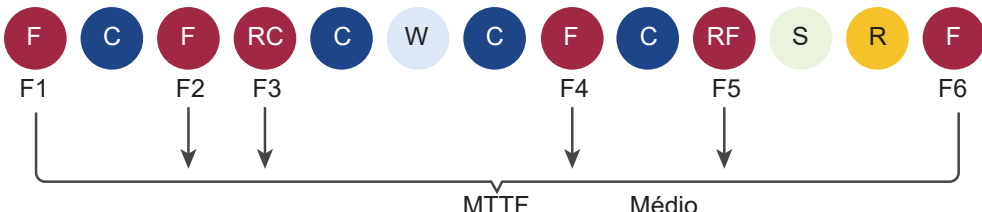
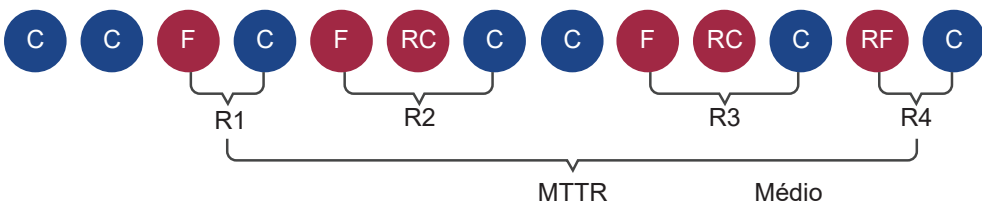
O que é avaliado...	O que significa...
CI média	Tempo médio gasto na fase de integração contínua, avaliado por tempo no tipo de tarefa de CI.
Tempo médio de entrega (MTTD)	<p>Duração média de todas as execuções com status COMPLETED ao longo de um período. D1, D2 e assim por diante é o tempo para entregar cada execução com status COMPLETED.</p> 
Tempo médio entre entregas (MTBD)	<p>Tempo médio decorrido entre as entregas bem-sucedidas ao longo de um período. O tempo decorrido entre duas execuções consecutivas com status COMPLETED é o tempo entre entregas bem-sucedidas, como BD1, BD2 e assim por diante. O MTBD indica com que frequência um ambiente de produção é atualizado.</p> 

Tabela 8-1. Medição de tempos médios (continuação)

O que é avaliado...	O que significa...
Tempo médio de falha (MTTF)	<p>Duração média de execuções que terminam nos estados FAILED, ROLLBACK_COMPLETED ou ROLLBACK_FAILED durante um período de tempo. F1, F2 e assim por diante é o tempo para uma execução terminar nos estados FAILURE, ROLLBACK_COMPLETED ou ROLLBACK_FAILED.</p> 
Tempo médio para recuperação (MTTR)	<p>O tempo médio para recuperação de uma falha ao longo de um período. O tempo para recuperação de uma falha é o tempo decorrido entre uma execução com um status final FAILED, ROLLBACK_COMPLETED ou ROLLBACK_FAILED e a próxima execução imediata bem-sucedida com um status COMPLETED. R1, R2 e assim por diante é o tempo para recuperação após cada execução com status FAILED ou ROLLBACK_FAILED.</p> 

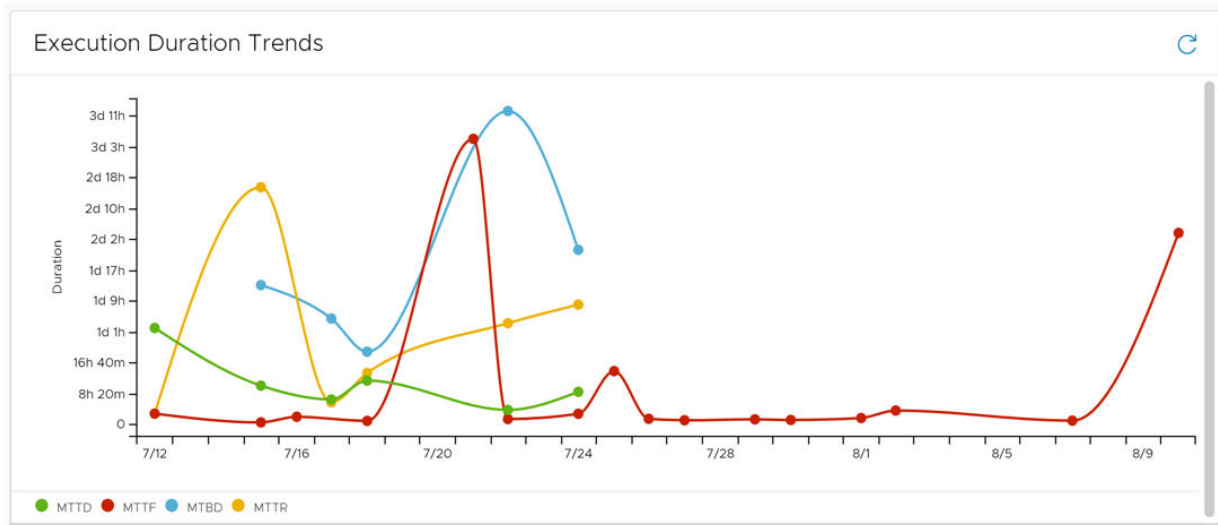
## Widgets de Principais Etapas e Tarefas com Falha

Dois widgets exibem os principais estágios e tarefas com falha em um pipeline. Cada medição registra o número e o percentual de falhas para ambientes de desenvolvimento e pós-desenvolvimento para cada pipeline e projeto, calculado por média sobre uma semana ou mês. Visualize as principais falhas para solucionar problemas no processo de automação da liberação.

Por exemplo, é possível configurar a exibição para uma duração específica, como os últimos sete dias, e anotar as principais tarefas que falharam durante esse período. Se você fizer uma alteração em seu ambiente ou pipeline e executar o pipeline novamente, verifique as principais tarefas que falharam em uma duração maior, como nos últimos 14 dias, as principais tarefas com falha podem ter mudado. Com esse resultado, é possível saber que a alteração no processo de automação da liberação melhorou a taxa de êxito da execução do pipeline.

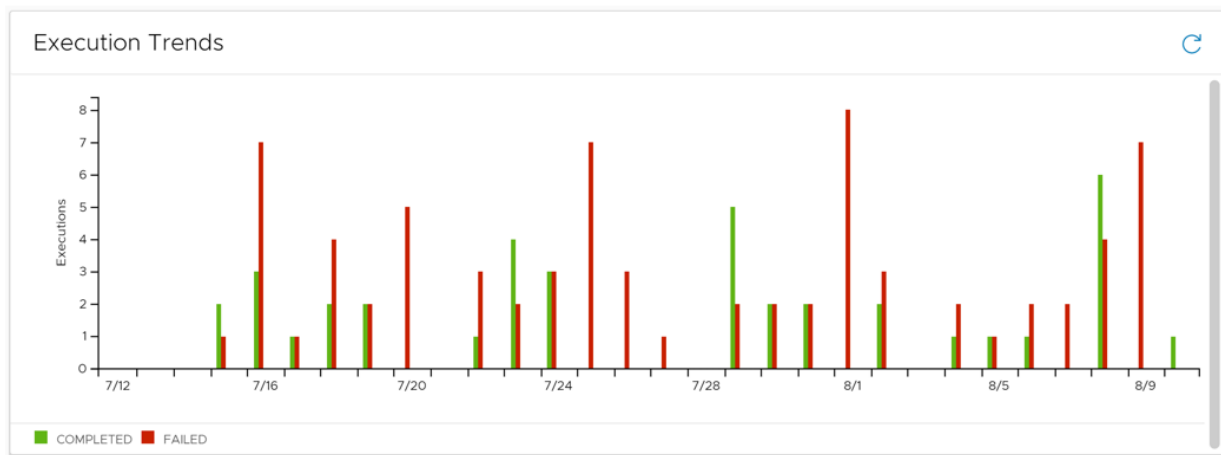
## Widget de Tendências de Duração de Execução de Pipeline

As tendências de duração de execuções de pipelines mostram os valores de MTTF, MTTR, MTBD e MTTR no decorrer de um período.



## Widget de Tendências de Execução de Pipeline

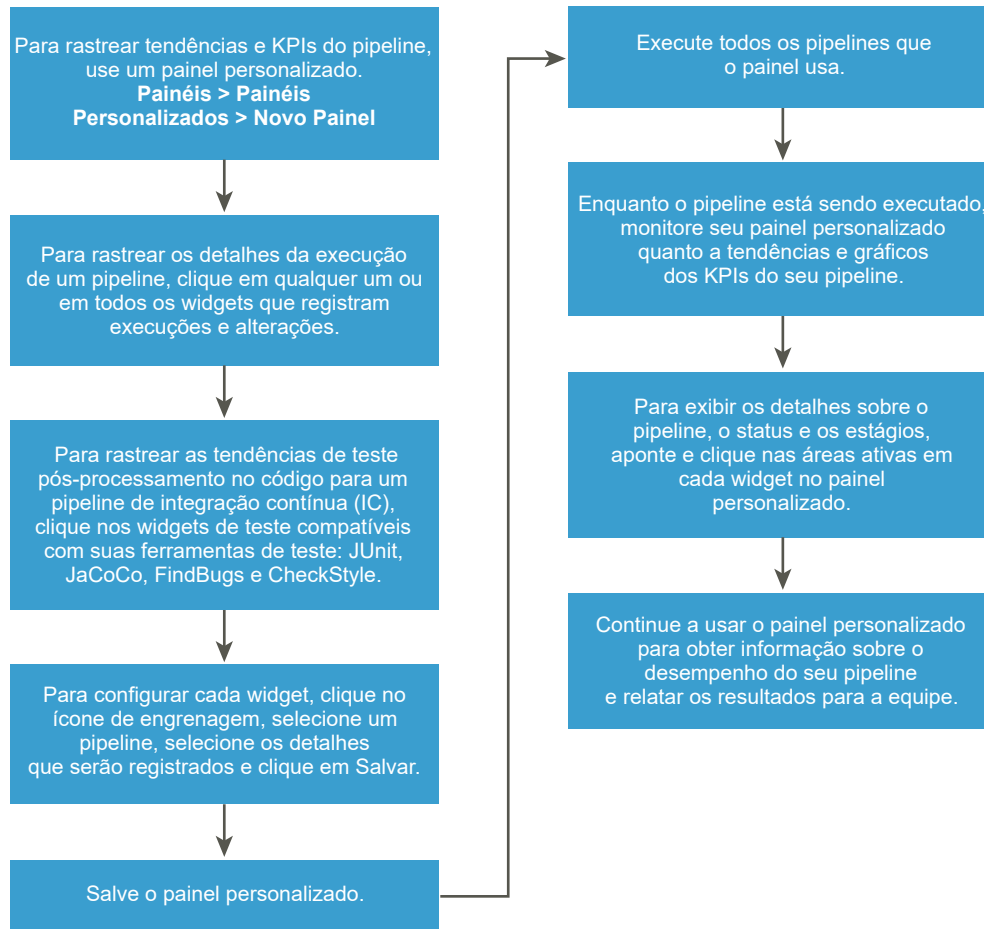
As tendências de execução do pipeline mostram o total de execuções diárias de um pipeline, agrupadas por status ao longo de um período. Exceto para o dia atual, a maioria das contagens de agregação diária mostra apenas execuções nos estados COMPLETED e FAILED.



## Como usar painéis personalizados para rastrear indicadores-chave de desempenho para o meu pipeline no Code Stream

Como administrador ou desenvolvedor do Code Stream, você cria o painel personalizado para exibir os resultados que deseja ver para um ou mais pipelines que foram executados. Por exemplo, é possível criar um painel de todo o projeto com KPIs e métricas coletadas de vários pipelines. Se um aviso ou falha de execução for relatado, você poderá usar o painel para solucionar a falha.

Para rastrear as tendências e os principais indicadores de desempenho dos pipelines usando um painel personalizado, adicione widgets ao painel e configure-os para registrar os pipelines.



### Pré-requisitos

- Verifique a existência de um ou mais pipelines. Na interface do usuário, clique em **Pipelines**.
- Verifique se os pipelines que você pretende monitorar foram executados com êxito. Clique em **Execuções**.

### Procedimentos

- 1 Para criar um painel personalizado, clique em **Painéis > Painéis Personalizados > Novo Painel**.

- 2 Para personalizar o painel de forma que ele registre tendências específicas e os principais indicadores de desempenho do pipeline, clique em um widget.

Por exemplo, para exibir detalhes sobre o status, estágios e tarefas do pipeline, por quanto tempo ele foi executado e quem o executou, clique no widget **Detalhes da Execução**.

Ou, para um pipeline de integração contínua (CI), é possível rastrear as tendências no pós-processo usando os widgets para JUnit, JaCoCo, FindBugs e CheckStyle.

Execution ID	Execution#	Status	Status Message	Stages	Tasks	Task0 (Stage0)	Duration
178f62eef...	#2	WAITING	Stage0.Task0 Execution Waiting for User Action.				15s
5503c1e51...	#1	COMPLETED	Execution Completed.				1h 28m 7s

- 3 Configure cada widget adicionado.
  - a No widget, clique no ícone de engrenagem.
  - b Selecione um pipeline, defina as opções disponíveis e selecione as colunas a serem exibidas.
  - c Para salvar a configuração do widget, clique em **Salvar**.
  - d Para salvar o painel personalizado, clique em **Salvar** e, depois, em **Fechar**.
- 4 Para exibir mais informações sobre o pipeline, clique nas áreas ativas nos widgets.
 

Por exemplo, no widget **Detalhes da Execução**, clique em uma entrada na coluna de Status para exibir mais informações sobre a execução do pipeline. Ou, no widget **Última Alteração Bem-Sucedida** para exibir um resumo do estágio e da tarefa do pipeline, clique no link ativo.

## Resultados

Parabéns! Você criou um painel personalizado que monitora tendências e KPIs dos pipelines.

## Próximo passo

Continue a monitorar o desempenho dos pipelines no Code Stream e compartilhe os resultados com o gerente e as equipes para melhorar o processo de liberação dos aplicativos.

# Saiba mais sobre o Code Stream

# 9

Há muitas maneiras para os administradores e desenvolvedores do Code Stream aprenderem mais sobre o Code Stream e o que ele pode fazer por você.

Você pode usar esta documentação para saber mais sobre pipelines e suas execuções, como adicionar endpoints, como adicionar projetos e muito mais.

Entenda as permissões que as funções fornecem. Saiba como usar recursos restritos e exigir aprovações nos pipelines. Consulte [Como gerenciar o acesso do usuário e as aprovações no Code Stream](#).

Consulte o valor da pesquisa ao descobrir onde os trabalhos ou componentes específicos estão localizados nos pipelines, execuções ou endpoints.

Este capítulo inclui os seguintes tópicos:

- [O que é a pesquisa no Code Stream](#)
- [Mais recursos para administradores e desenvolvedores do Code Stream](#)

## O que é a pesquisa no Code Stream

É possível usar a pesquisa para descobrir onde itens específicos ou outros componentes estão localizados. Por exemplo, talvez você queira procurar por pipelines ativados ou desativados. Porque, se um pipeline estiver desativado, ele não poderá ser executado.

## O que é possível pesquisar

Você pode pesquisar em:

- Projetos
- Endpoints
- Pipelines
- Execuções
- Painéis de pipeline, painéis personalizados
- Servidores e gatilhos Gerrit
- Git Webhooks

- Webhooks do Docker

Você pode realizar a pesquisa de filtro com base em coluna em:

- Operações do Usuário
- Variáveis
- Atividade de gatilho para Gerrit, Git e Docker

É possível executar a pesquisa de filtro com base em grade na página **Atividade** para cada gatilho.

## Como funciona a pesquisa

Os critérios para a pesquisa variam de acordo com a página em que você está. Cada página tem diferentes critérios de pesquisa.

Onde você pesquisa	Crítérios a serem usados para pesquisa
Painéis de Pipeline	Projeto, Nome, Descrição, Tags, Link
Painéis Personalizados	Projeto, Nome, Descrição, Link (UUID de um item no painel)
Execuções	Nome, Comentários, Motivo, Tags, Índice, Status, Projeto, Mostrar, Executado por, Executado por mim, Link (UUID da execução) e Parâmetros de entrada, Parâmetros de saída ou Mensagem de status usando este formato: <key>:<value>
Pipelines	Nome, Descrição, Estado, Tags, Criado por, Criado por mim, Atualizado por, Atualizado por mim, Projeto
Projetos	Nome, Descrição
Endpoints	Nome, Descrição, Tipo, Atualizado por, Projeto
Gatilhos Gerrit	Nome, Status, Projeto
Servidores Gerrit	Nome, URL do Servidor, Projeto
Git Webhooks	Nome, Tipo de Servidor, Repositório, Ramificação, Projeto

Onde:

- O link é o UUID de um pipeline, execução ou widget em um painel.
- A notação e exemplos de parâmetros de entrada, parâmetros de saída e mensagens de status incluem:
  - Notação: `input.<inputKey>:<inputValue>`  
Exemplo: `input.GERRIT_CHANGE_OWNER_EMAIL:joe_user`
  - Notação: `output.<outputKey>:<outputValue>`  
Exemplo: `output.BuildNo:29`

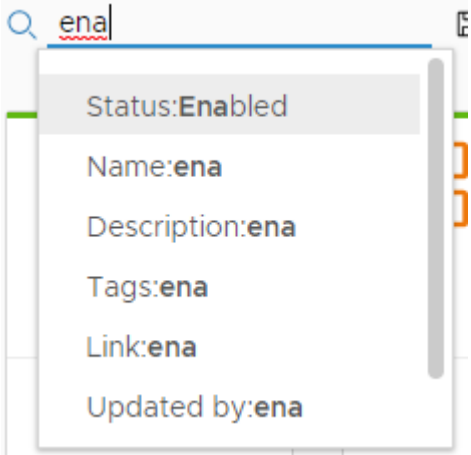
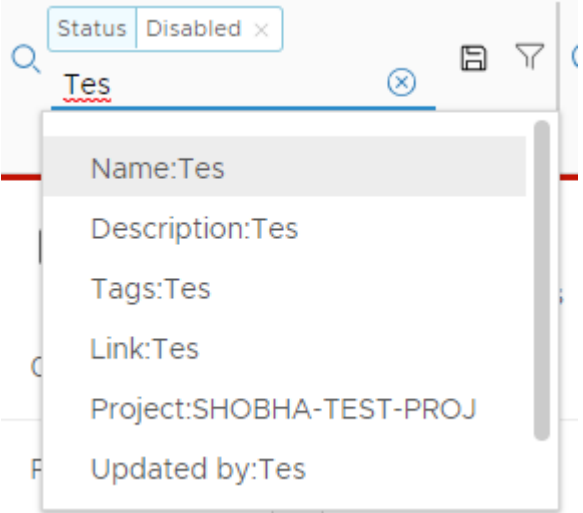


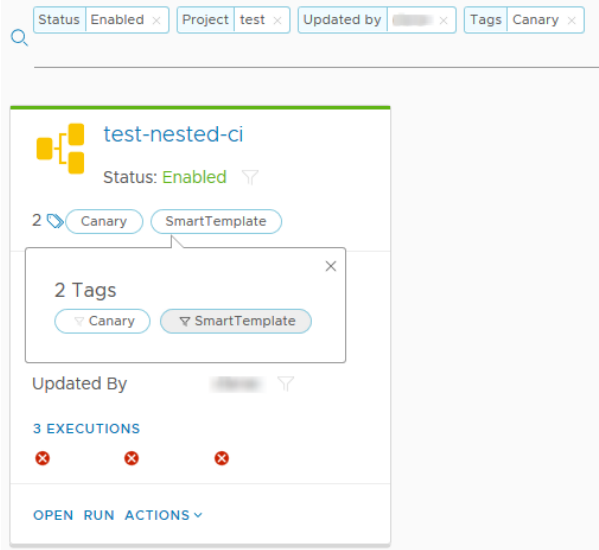
- **Notação:** `statusMessage:<value>`

Exemplo: **`statusMessage:Execution failed`**

- O status ou estado depende da página de pesquisa.
  - Para execuções, os valores possíveis incluem: concluído, com falha, reversão\_falhou ou cancelado.
  - Para pipelines, os possíveis valores de estado incluem: habilitado, desabilitado ou liberado.
  - Para gatilhos, os possíveis valores de status incluem: habilitado ou desabilitado.
- Executado, Criado ou Atualizado por Mim se referem a mim, o usuário conectado.

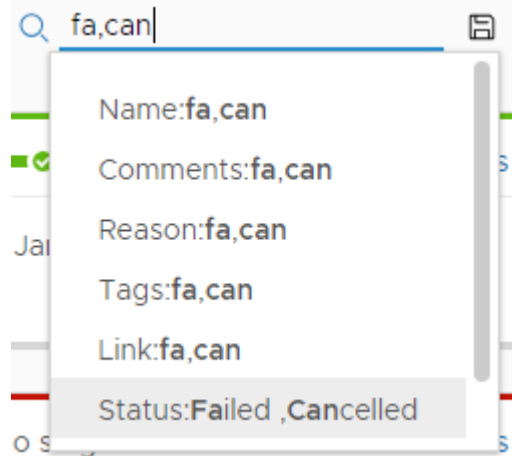
A pesquisa aparece no canto superior direito de cada página válida. Quando você começa a digitar na pesquisa em branco, o Code Stream reconhece o contexto da página e sugere opções para a pesquisa.

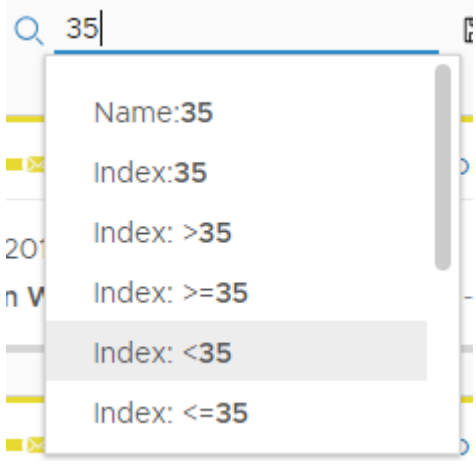
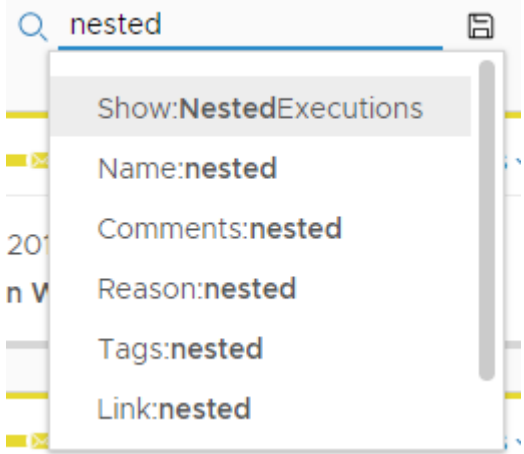
Métodos que podem ser usados para pesquisar	Como digitar
<p>Digite uma parte do parâmetro de pesquisa.</p> <p>Por exemplo, para adicionar um filtro de status que liste todos os pipelines ativados, digite <b>ena</b>.</p>	
<p>Para reduzir o número de itens encontrados, adicione um filtro.</p> <p>Por exemplo, digite <b>Tes</b> para adicionar um filtro de nome. O filtro funciona como E com o filtro existente de <b>Status: Desativado</b> para mostrar apenas os pipelines desativados com <b>Tes</b> no nome.</p> <p>Quando você adiciona outro filtro, as opções restantes são exibidas: <b>Nome, Descrição, Tag, Link, Projeto e Atualizado por</b>.</p>	

Métodos que podem ser usados para pesquisar	Como digitar
<p>Para reduzir o número de itens exibidos, clique no ícone de filtro nas propriedades de um pipeline ou em uma execução de pipeline.</p> <ul style="list-style-type: none"> <li>■ Para pipelines, <b>Status</b>, <b>Tags</b>, <b>Projeto</b> e <b>Atualizado por</b> têm, cada um, um ícone de filtro.</li> <li>■ Para execuções, <b>Tags</b>, <b>Executado por</b> e <b>Mensagem de status</b> têm, cada um, um ícone de filtro.</li> </ul> <p>Por exemplo, no cartão de pipeline, clique no ícone para adicionar o filtro para a tag <b>SmartTemplate</b> aos filtros existentes para: <b>Status:Enabled</b>, <b>Project:test</b>, <b>Updated by:user</b> e <b>Tags:Canary</b>.</p>	

Use um separador de vírgula para incluir todos os itens em dois estados de execução.

Por exemplo, digite **fa,can** para criar um filtro de status que funcione como um **OR** para listar todas as execuções que falharam ou cancelamento.

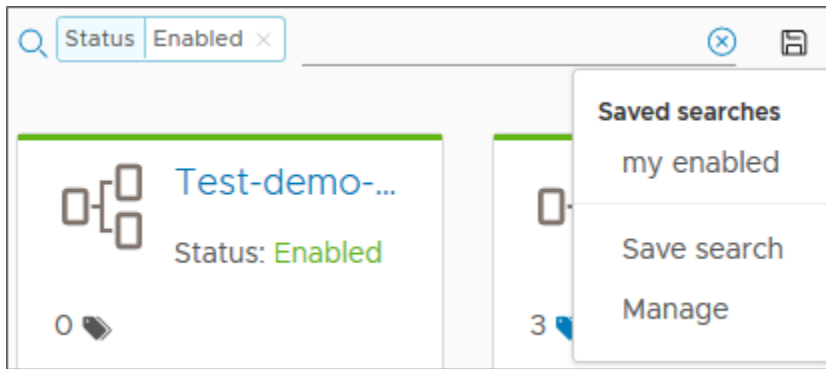


Métodos que podem ser usados para pesquisar	Como digitar
<p>Digite um número para incluir todos os itens em um intervalo de índice.</p> <p>Por exemplo, digite <b>35</b> e selecione <b>&lt;</b> para listar todas as execuções com um número de índice inferior a 35.</p>	
<p>Os pipelines são modelados conforme as tarefas se tornam execuções aninhadas e não são listadas com todas as execuções por padrão.</p> <p>Para mostrar execuções aninhadas, digite <b>nested</b> e selecione o filtro <b>Exibir</b>.</p>	

## Como salvar uma pesquisa favorita

É possível salvar as pesquisas favoritas para serem usadas em todas as páginas clicando no ícone de disco ao lado da área de pesquisa.

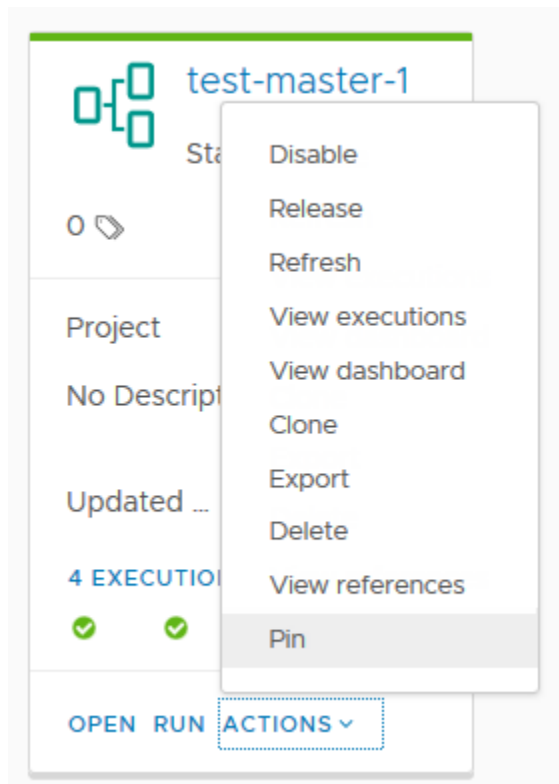
- Salve uma pesquisa digitando os parâmetros para pesquisa e clicando no ícone para dar um nome de pesquisa como **ativado por mim**.
- Depois de salvar uma pesquisa, clique no ícone para acessar a pesquisa. Também é possível selecionar **Gerenciar** para renomear, excluir ou mover a pesquisa na lista de pesquisas salvas.



As pesquisas são ligadas ao seu nome do usuário e aparecem apenas nas páginas às quais a pesquisa se aplica. Por exemplo, se você tiver salvo uma pesquisa chamada **ativado por mim** para **Status: ativado** na página Pipelines, a pesquisa **ativado por mim** não está disponível na página de gatilhos Gerrit, mesmo que o **Status: ativado** seja uma pesquisa válida para um gatilho.

## Posso salvar um pipeline favorito?

Se você tiver um pipeline ou um painel favorito, poderá fixá-los para que sempre apareçam na parte superior da sua página de pipelines ou painéis. No cartão de pipeline, clique em **Ações > Fixar**.



# Mais recursos para administradores e desenvolvedores do Code Stream

Como administrador ou desenvolvedor do Code Stream, você pode saber mais sobre o Code Stream.

**Tabela 9-1. Mais recursos para administradores**

Para saber mais sobre...	Consulte estes recursos...
<p>Outras maneiras em que os administradores podem usar o Code Stream:</p> <ul style="list-style-type: none"> <li>■ Configurar os pipelines para automatizar o teste e a liberação dos aplicativos nativos da nuvem.</li> <li>■ Automatizar e testar o código-fonte do desenvolvedor, por meio de testes até a produção.</li> <li>■ Configurar os pipelines para os desenvolvedores testarem as alterações antes que eles as confirmem para a ramificação primária.</li> <li>■ Rastrear as principais métricas de pipeline.</li> </ul>	<p>Code Stream</p> <ul style="list-style-type: none"> <li>■ <a href="#">Documentação do vRealize Automation</a></li> <li>■ <a href="#">Site do produto vRealize Automation</a></li> </ul> <p>VMware Hands On</p> <ul style="list-style-type: none"> <li>■ Use a <a href="#">Comunidade do vRealize Automation</a>.</li> <li>■ Use a <a href="#">Zona de Aprendizagem do VMware</a>.</li> <li>■ Pesquise nos <a href="#">Blogs de VMware</a>.</li> <li>■ Experimente os <a href="#">Laboratórios do VMware Hands On</a>.</li> </ul>

**Tabela 9-2. Mais recursos para desenvolvedores**

Para saber mais sobre...	Consulte estes recursos...
<p>Outras maneiras em que os desenvolvedores podem usar o Code Stream:</p> <ul style="list-style-type: none"> <li>■ Usar imagens de registro públicas e privadas para compilar ambientes para novos aplicativos ou serviços.</li> <li>■ Definir ambientes de desenvolvimento para que você possa criar ramificações a partir da compilação estável mais recente.</li> <li>■ Atualizar os ambientes de desenvolvimento com as alterações mais recentes de código e artefatos.</li> <li>■ Testar as alterações de código não confirmadas nas compilações estáveis mais recentes de outros serviços dependentes.</li> <li>■ Receber uma notificação quando uma alteração confirmada em um pipeline de ICEC primário interromper outros serviços.</li> </ul>	<p>Code Stream</p> <ul style="list-style-type: none"> <li>■ <a href="#">Documentação do vRealize Automation</a></li> <li>■ <a href="#">Site do produto vRealize Automation</a></li> </ul> <p>VMware Hands On</p> <ul style="list-style-type: none"> <li>■ Use a <a href="#">Comunidade do vRealize Automation</a>.</li> <li>■ Use a <a href="#">Zona de Aprendizagem do VMware</a>.</li> <li>■ Pesquise nos <a href="#">Blogs de VMware</a>.</li> <li>■ Experimente os <a href="#">Laboratórios do VMware Hands On</a>.</li> </ul>