

Utilizzo e gestione di vRealize Automation Code Stream

14 dicembre 2022

vRealize Automation 8.7

È possibile trovare la documentazione tecnica più aggiornata sul sito Web di VMware all'indirizzo:

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

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

VMware, Inc.
Centro Leoni Palazzo A
Via Spadolini 5
Ground Floor
Milan, MI 20121
tel: +39 02 30412700
fax: +39 02 30412701
www.vmware.com/it

Copyright © 2022 VMware, Inc. Tutti i diritti sono riservati. [Informazioni sul copyright e sui marchi](#)

Sommario

- 1 Che cos'è Code Stream e come funziona 5**
- 2 Impostazione per modellare il processo di rilascio 10**
 - Come aggiungere un progetto 15
 - Come gestire l'accesso utente e le approvazioni 16
 - Che cosa sono le operazioni utente e le approvazioni 25
- 3 Creazione e utilizzo di pipeline 27**
 - Come eseguire una pipeline e visualizzare i risultati 30
 - Tipi di attività disponibili 35
 - Come utilizzare i binding delle variabili nelle pipeline 40
 - Come utilizzare le associazioni di variabili in un'attività di condizione per eseguire o arrestare una pipeline 50
 - Variabili ed espressioni che è possibile utilizzare quando si associano le attività della pipeline 53
 - Come inviare notifiche su una pipeline 71
 - Come creare un ticket Jira quando un'attività della pipeline non riesce 74
 - Come eseguire il rollback di una distribuzione 76
- 4 Pianificazione di creazione, integrazione e consegna native del codice 83**
 - Configurazione dell'area di lavoro della pipeline 83
 - Pianificazione di una creazione CI/CD nativa prima di utilizzare il modello di pipeline smart 87
 - Pianificazione di una creazione CI nativa prima di utilizzare il modello di pipeline smart 94
 - Pianificazione di una creazione CD nativa prima di utilizzare il modello di pipeline smart 95
 - Pianificazione di una creazione nativa di CI/CD prima dell'aggiunta manuale di attività 97
 - Pianificazione del rollback 103
- 5 Tutorial 106**
 - Come integrare in modo continuativo il codice dal repository GitHub o GitLab nella pipeline 107
 - Come automatizzare il rilascio di un'applicazione da distribuire da un modello cloud YAML 112
 - Come automatizzare il rilascio di un'applicazione in un cluster Kubernetes 119
 - Come distribuire un'applicazione nella distribuzione blu-verde 127
 - Come integrare gli strumenti di creazione, test e distribuzione 132
 - Come utilizzare le proprietà delle risorse di un'attività del modello cloud nell'attività successiva 143
 - Come utilizzare una REST API da integrare con altre applicazioni 147
 - Come sfruttare la pipeline come codice 152
- 6 Connessione agli endpoint 158**

- Che cosa sono gli endpoint 158
- Come eseguire l'integrazione con Jenkins 161
- Come eseguire l'integrazione 167
- Come eseguire l'integrazione con Gerrit 169
- Come eseguire l'integrazione con vRealize Orchestrator 173

7 Attivazione delle pipeline 179

- Come utilizzare il trigger Docker per eseguire una pipeline di consegna continua 179
- Come utilizzare il trigger Git per eseguire una pipeline 188
- Come utilizzare il trigger Gerrit per eseguire una pipeline 196

8 Monitoraggio delle pipeline 205

- Elementi visualizzati nel dashboard della pipeline 205
- Come utilizzare dashboard personalizzati per tenere traccia degli indicatori di prestazioni chiave 209

9 Ulteriori informazioni 211

- Che cos'è la ricerca 211
- Altre risorse per amministratori e sviluppatori 217

Che cos'è Code Stream e come funziona

1

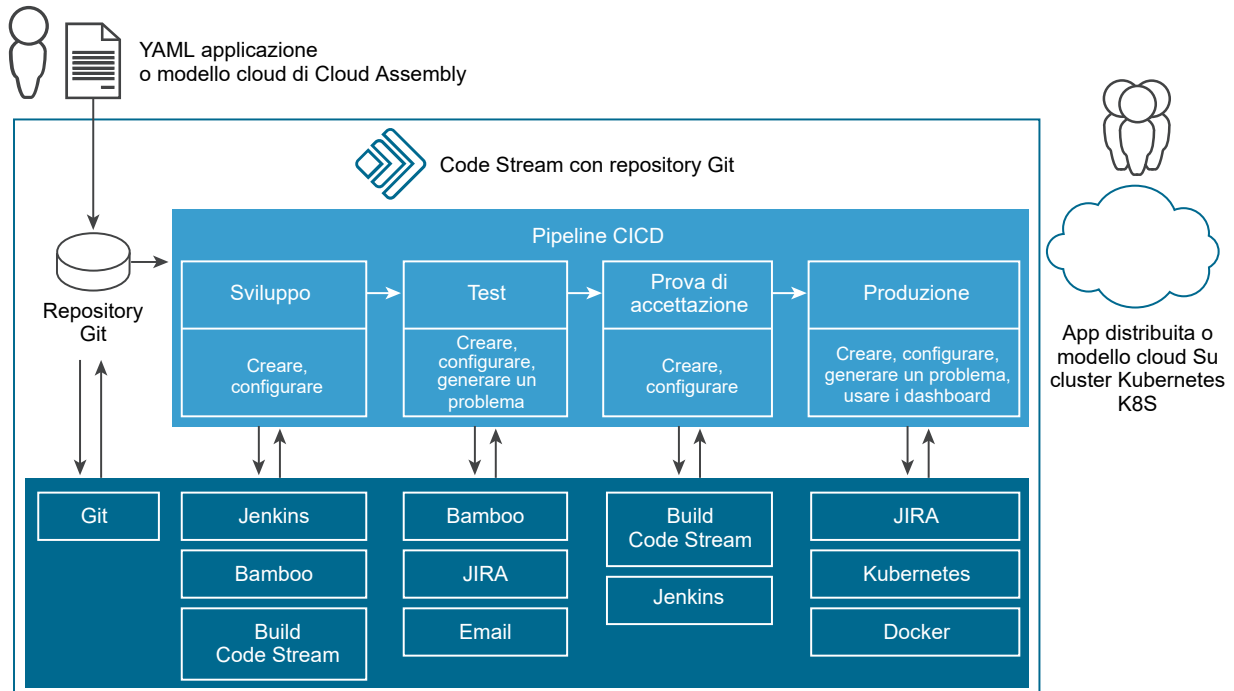
vRealize Automation Code Stream™ è uno strumento di integrazione e consegna continue (CI/CD). Creando pipeline che modellano il processo di rilascio del software nel ciclo di vita di DevOps, si crea l'infrastruttura di codice che consegna il software in modo rapido e continuativo.



Quando si utilizza Code Stream per consegnare il software, si integrano due delle parti più importanti del ciclo di vita di DevOps: il processo di rilascio e gli strumenti di sviluppo. Dopo la configurazione iniziale, che integra Code Stream con gli strumenti di sviluppo esistenti, le pipeline automatizzano l'intero ciclo di vita di DevOps.

A partire da vRealize Automation 8.2, i blueprint vengono chiamati VMware Cloud Templates.

Si crea una pipeline che crea, testa e rilascia il software. Code Stream utilizza tale pipeline per far procedere il software dal repository del codice di origine attraverso i livelli di test e distribuirlo alla produzione.



Ulteriori informazioni sulla pianificazione delle pipeline di integrazione e consegna continue sono disponibili qui: [Capitolo 4 Pianificazione di creazione, integrazione e consegna native del codice in Code Stream](#).

Come gli amministratori di Code Stream utilizzano Code Stream

Gli amministratori possono creare endpoint e assicurarsi che gli sviluppatori abbiano a disposizione istanze funzionanti. Inoltre si creano, si attivano e si gestiscono le pipeline e molto altro. Si dispone del ruolo di *Administrator*, come descritto in [Come gestire accesso utente e approvazioni in Code Stream](#).

Tabella 1-1. Come gli amministratori di Code Stream supportano gli sviluppatori

Per supportare gli sviluppatori...	Procedere come segue...
Fornire e gestire gli ambienti.	<p>Creare ambienti in cui gli sviluppatori possano testare e distribuire il codice.</p> <ul style="list-style-type: none"> ■ Tenere traccia dello stato e inviare notifiche tramite e-mail. ■ Garantire la produttività degli sviluppatori assicurandosi che gli ambienti che utilizzano funzionino sempre. <p>Per ulteriori informazioni, vedere Altre risorse per amministratori e sviluppatori di Code Stream.</p> <p>Vedere anche Capitolo 5 Tutorial sull'utilizzo di Code Stream.</p>
Fornire gli endpoint.	Assicurarsi che gli sviluppatori dispongano di istanze funzionanti di endpoint in grado di connettersi alle rispettive pipeline.
Fornire integrazioni con altri servizi.	<p>Assicurarsi che le integrazioni con gli altri servizi funzionino.</p> <p>Per ulteriori informazioni, vedere la documentazione di VMware Cloud Services.</p>

Tabella 1-1. Come gli amministratori di Code Stream supportano gli sviluppatori (continua)

Per supportare gli sviluppatori...	Procedere come segue...
Creare pipeline.	<p>Creare pipeline che modellino i processi di rilascio.</p> <p>Per ulteriori informazioni, vedere Capitolo 3 Creazione e utilizzo di pipeline in Code Stream.</p>
Attivare le pipeline.	<p>Assicurarsi che le pipeline vengano eseguite quando si verificano eventi.</p> <ul style="list-style-type: none"> ■ Per attivare una pipeline di consegna continua (CD) indipendente ogni volta che viene creato o aggiornato un artefatto di creazione, utilizzare il trigger Docker. ■ Per attivare una pipeline quando uno sviluppatore conferma le modifiche al codice, utilizzare il trigger Git. ■ Per attivare una pipeline quando gli sviluppatori esaminano il codice, uniscono modifiche o eseguono altre operazioni, utilizzare il trigger Gerrit. ■ Per eseguire una pipeline di consegna continua (CD) standalone ogni volta che viene creato o aggiornato un artefatto di creazione utilizzare il trigger Docker. <p>Per ulteriori informazioni, vedere Capitolo 7 Attivazione di pipeline in Code Stream.</p>
Gestire le pipeline e le approvazioni.	<p>Ricevere aggiornamenti continui sulle pipeline.</p> <ul style="list-style-type: none"> ■ Visualizzare lo stato della pipeline e chi ha eseguito le pipeline. ■ Visualizzare le approvazioni per le esecuzioni delle pipeline e gestire le approvazioni per le esecuzioni di pipeline attive e inattive. <p>Per ulteriori informazioni, vedere Che cosa sono le operazioni utente e le approvazioni in Code Stream.</p> <p>Vedere anche Come utilizzare dashboard personalizzati per tenere traccia degli indicatori di prestazioni chiave per la pipeline in Code Stream.</p>
Monitorare gli ambienti di sviluppo.	<p>Creare dashboard personalizzati in grado di monitorare lo stato, le tendenze, le metriche e gli indicatori principali della pipeline. Utilizzare i dashboard personalizzati per monitorare le pipeline che riescono o non riescono negli ambienti di sviluppo. È inoltre possibile identificare e segnalare le risorse sottoutilizzate, nonché liberare risorse.</p> <p>È inoltre possibile:</p> <ul style="list-style-type: none"> ■ Per quanto tempo è stata eseguita una pipeline prima di essere completata. ■ Per quanto tempo una pipeline ha atteso l'approvazione e inviare notifica all'utente che deve approvarla. ■ Quali sono le fasi e le attività che più spesso non riescono. ■ Quali sono le fasi e le attività che impiegano più tempo per l'esecuzione. ■ Quali sono le versioni a cui attualmente lavorano i team di sviluppo. ■ Quali sono le applicazioni che sono state distribuite e rilasciate correttamente. <p>Per ulteriori informazioni, vedere Capitolo 8 Monitoraggio delle pipeline in Code Stream.</p>
Risolvere i problemi.	<p>Risolvere i problemi e gli errori della pipeline negli ambienti di sviluppo.</p> <ul style="list-style-type: none"> ■ Identificare e risolvere i problemi che si verificano negli ambienti di integrazione e consegna continue (CICD). ■ Utilizzare i dashboard della pipeline e creare dashboard personalizzati per ulteriori dettagli. Vedere Capitolo 8 Monitoraggio delle pipeline in Code Stream. <p>Vedere anche Capitolo 2 Impostazione di Code Stream per modellare il processo di rilascio.</p>

Code Stream fa parte di VMware Cloud Services.

- Utilizzare Cloud Assembly per distribuire modelli di cloud.
- Utilizzare Service Broker per ottenere modelli di cloud dal catalogo.

Per conoscere le altre operazioni che è possibile eseguire, vedere la [documentazione di VMware vRealize Automation](#).

Come gli sviluppatori utilizzano Code Stream

Gli sviluppatori possono utilizzare Code Stream per creare ed eseguire pipeline e per monitorare l'attività delle pipeline sui dashboard. Si dispone del ruolo di `user`, come descritto in [Come gestire accesso utente e approvazioni in Code Stream](#).

Dopo aver eseguito una pipeline, si desidererà sapere:

- Se il codice ha avuto esito positivo in tutte le fasi della pipeline. Per saperlo, osservare i risultati nelle esecuzioni della pipeline.
- Come comportarsi se la pipeline non è riuscita e come sapere qual è stata la causa dell'esito negativo. Per scoprirlo, osservare gli errori principali nei dashboard della pipeline.

Tabella 1-2. Sviluppatori che utilizzano Code Stream

Per integrare e rilasciare il codice	Procedere come segue
Creare pipeline.	Testare e distribuire il codice. Aggiornare il codice quando una pipeline non riesce.
Connettere la pipeline agli endpoint.	Connettere le attività della pipeline agli endpoint, ad esempio un repository GitHub.
Eseguire pipeline.	Aggiungere un'attività di approvazione dell'operazione utente in modo che un altro utente possa approvare la pipeline in punti specifici.
Visualizzare dashboard.	Visualizzare i risultati nel dashboard della pipeline. È possibile visualizzare le tendenze, la cronologia, gli errori e molto altro.

Per ulteriori informazioni su come iniziare, vedere [Guida introduttiva a VMware Code Stream](#).

Ulteriori informazioni sono disponibili nel pannello di supporto all'interno del prodotto

Se non si trovano le informazioni necessarie, è possibile ottenere ulteriore assistenza nel prodotto.



- Fare clic e leggere le indicazioni e le descrizioni comandi nell'interfaccia utente per ottenere le informazioni specifiche del contesto dove e quando sono necessarie.

- Aprire il pannello di supporto all'interno del prodotto e leggere gli argomenti visualizzati per la pagina dell'interfaccia utente attiva. È inoltre possibile cercare nel pannello per ottenere risposte alle domande.

Ulteriori informazioni sui webhook

È possibile creare più webhook per rami diversi utilizzando lo stesso endpoint Git e specificando valori diversi per il nome del ramo nella pagina di configurazione del webhook. Per creare un altro webhook per un altro ramo nello stesso repository Git, non è necessario clonare l'endpoint Git più volte per più rami. È infatti sufficiente specificare il nome del ramo nel webhook in modo da poter riutilizzare l'endpoint Git. Se il ramo nel webhook Git è lo stesso ramo dell'endpoint, non è necessario specificare il nome del ramo nella pagina del webhook Git.

Impostazione di Code Stream per modellare il processo di rilascio

2

Per modellare il processo di rilascio, si crea una pipeline che rappresenta le fasi, le attività e le approvazioni normalmente utilizzate per rilasciare il software. Code Stream automatizza quindi il processo che crea, testa, approva e distribuisce il codice.

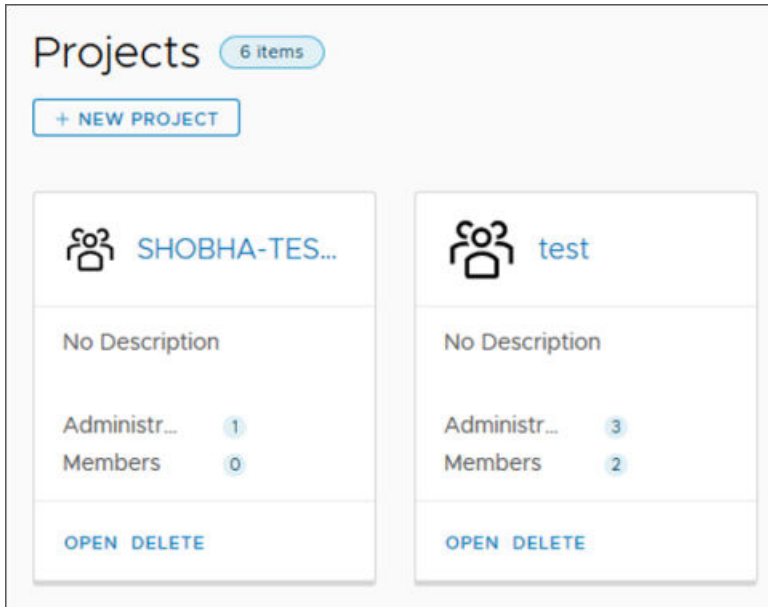
Ora che tutte le condizioni per modellare il processo di rilascio del software sono soddisfatte, vediamo come procedere in Code Stream.

Prerequisiti

- Verificare se sono già disponibili endpoint. In Code Stream fare clic su **Endpoint**.
- Informazioni sulle modalità native per la creazione e la distribuzione del codice. Vedere [Capitolo 4 Pianificazione di creazione, integrazione e consegna native del codice in Code Stream](#).
- Stabilire se alcune delle risorse che verranno utilizzate nella pipeline devono essere contrassegnate come limitate. Vedere [Come gestire accesso utente e approvazioni in Code Stream](#).
- Se non si dispone del ruolo di amministratore ma si dispone di quello di utente o di visualizzatore, stabilire chi è l'amministratore dell'istanza di Code Stream.

Procedura

- 1 Esaminare i progetti disponibili in Code Stream e selezionarne uno appropriato.
 - Se non sono visualizzati progetti, chiedere di creare un progetto e farsi attribuire il ruolo di membro del progetto a un amministratore di Code Stream dotato dei privilegi necessari. Vedere [Come aggiungere un progetto in Code Stream](#).
 - Se non si è membri di alcun progetto elencato, chiedere di essere aggiunti come membri di un progetto a un amministratore di Code Stream dotato dei privilegi necessari.



- 2 Aggiungere tutti i nuovi endpoint necessari per la pipeline.

Ad esempio, potrebbero essere necessari Git, Jenkins, Code Stream Build, Kubernetes e JIRA.

- 3 Creare variabili in modo da poter riutilizzare i valori nelle attività della pipeline.

Per vincolare le risorse utilizzate nelle pipeline, ad esempio una macchina host, utilizzare variabili limitate. È possibile impedire che la pipeline continui a essere eseguita finché un altro utente non la approva esplicitamente.

Gli amministratori possono creare variabili segrete e variabili limitate. Gli utenti possono creare variabili segrete.

È possibile riutilizzare una variabile in più pipeline tutte le volte che si desidera. Ad esempio, una variabile che definisce una macchina host può essere `HostIPAddress`. Per utilizzare la variabile in un'attività della pipeline, immettere `${var.HostIPAddress}`.

The screenshot shows the 'Variables' section of the vRealize Automation interface. It has a '3 items' badge and a 'NEW' button. Below is a table with the following data:

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

- 4 Se si è un amministratore, contrassegnare gli endpoint e le variabili che sono di vitale importanza per la propria azienda come risorse limitate.

Quando un utente che non è un amministratore tenta di eseguire una pipeline che include una risorsa limitata, la pipeline si arresta in corrispondenza dell'attività che utilizza la risorsa limitata. Un amministratore deve quindi riprendere la pipeline.

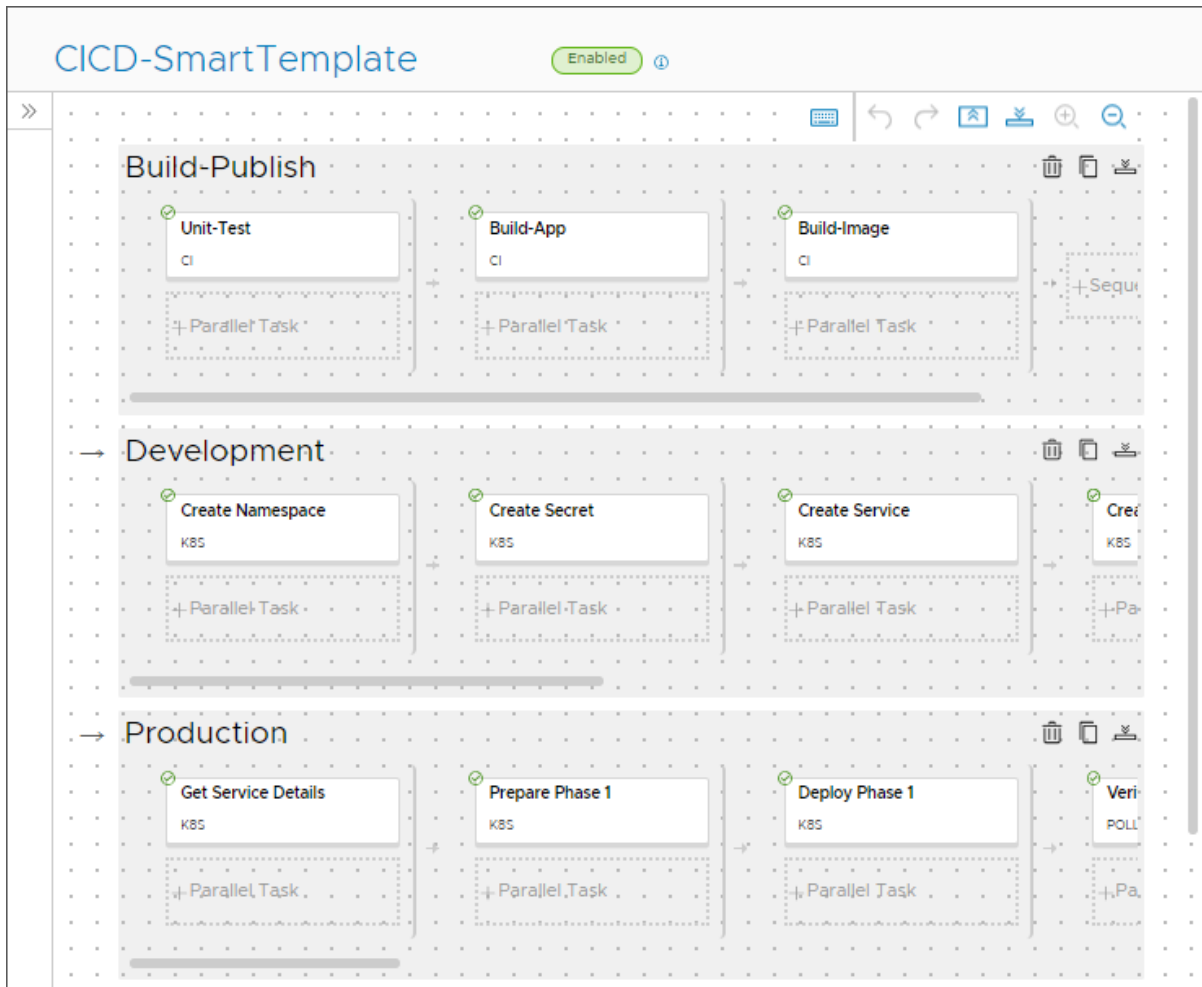
- 5 Pianificare la strategia di creazione per la pipeline CICD, CI o CD nativa.

Prima di creare una pipeline che integri (CI) e consegna (CD) in modo continuativo il codice, pianificare la strategia di creazione. Il piano di creazione consente di determinare quanto necessario a Code Stream affinché possa creare, integrare, testare e distribuire il codice in modo nativo.

Come creare una build nativa di Code Stream	Risultati in questa strategia di creazione
Utilizzare uno dei modelli di pipeline smart.	<ul style="list-style-type: none"> ■ Crea automaticamente tutte le fasi e le attività. ■ Clona il repository di origine. ■ Crea e testa il codice. ■ Inserisce il codice in contenitori per la distribuzione. ■ Compila i passaggi delle attività della pipeline in base alle selezioni.
Aggiungere manualmente fasi e attività.	Aggiungere fasi, aggiungere attività e immettere le informazioni necessarie per compilarle.

- 6 Creare la pipeline utilizzando un modello di pipeline smart oppure aggiungere manualmente fasi e attività alla pipeline.

Quindi, contrassegnare le risorse come limitate. Aggiungere approvazioni, se necessario. Applicare tutte le variabili regolari, limitate o segrete. Aggiungere eventuali associazioni tra le attività.



7 Convalidare, abilitare ed eseguire la pipeline.

8 Visualizzare le esecuzioni della pipeline.

Executions 280 items

[NEW EXECUTION](#)

Icon	Name	Status	Stages	Actions	User	Time	Input	Output
	Demo-Jenkins... #95	COMPLETED	████████	ACTIONS	kr	09/11/2018 10:32 AM	8df0d9a1d365299f2...	NA
	Demo-Jenkins... #94	COMPLETED	████████	ACTIONS	kr	09/11/2018 9:17 AM	6d82d079a8b8921a9...	NA
	Demo-CICD-S... #51	COMPLETED	████████	ACTIONS	dk	09/11/2018 7:13 AM	NA	NA
	Demo-CICD-S... #50	FAILED	██████████	ACTIONS	dk	09/11/2018 5:51 AM	NA	NA

Note: Execution #50 failed on task 'Production.Deploy Phase 1'. deployments....'

9 Per tenere traccia dello stato e degli indicatori di prestazioni chiave (KPI), utilizzare i dashboard della pipeline e creare dashboard personalizzati.

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

Risultati

È stata creata una pipeline utilizzabile nel progetto selezionato.

È possibile anche esportare il file YAML della pipeline per poi importarlo e riutilizzarlo in altri progetti.

Operazioni successive

Ulteriori informazioni sui casi d'uso che potrebbero essere applicati nell'ambiente in uso. Vedere [Capitolo 5 Tutorial sull'utilizzo di Code Stream](#).

Come aggiungere un progetto in Code Stream

È possibile creare un progetto e aggiungerci amministratori e membri. I membri del progetto possono utilizzare funzionalità come la creazione di una pipeline e l'aggiunta di un endpoint. Per creare, eliminare o aggiornare un progetto per un team di sviluppo, è necessario essere un amministratore di Code Stream.

Deve esistere un progetto prima che sia possibile creare una pipeline. Quando si crea una pipeline, si seleziona un progetto che raggruppa tutte le informazioni sulla pipeline. Anche le definizioni per gli endpoint e le variabili dipendono da un progetto esistente.

Prerequisiti

- Verificare di disporre del ruolo di amministratore di Code Stream. Vedere [Quali sono i ruoli di Code Stream](#).

Se non si dispone del ruolo di amministratore di Code Stream, ma si è un amministratore in Cloud Assembly, è possibile utilizzare l'interfaccia utente di Cloud Assembly per creare, aggiornare o eliminare i progetti. Vedere [Come aggiungere un progetto per il team di sviluppo di Cloud Assembly](#)

- Se si aggiungono gruppi di Active Directory ai progetti, verificare di aver configurato i gruppi di Active Directory per l'organizzazione. Vedere [Come abilitare i gruppi di Active Directory in vRealize Automation per i progetti](#). Se i gruppi non sono sincronizzati, non sono disponibili quando si tenta di aggiungerli a un progetto.

Procedura

- 1 Selezionare **Progetti**, quindi fare clic su **Nuovo progetto**.
- 2 Immettere il nome del progetto.
- 3 Fare clic su **Crea**.
- 4 Selezionare la scheda per il progetto appena creato, quindi fare clic su **Apri**.
- 5 Fare clic sulla scheda **Utenti**, aggiungere utenti e assegnare ruoli.
 - L'amministratore del progetto può aggiungere membri.
 - Il membro del progetto che dispone di un ruolo di servizio può utilizzare i servizi.

- Il visualizzatore del progetto può visualizzare i progetti ma non può crearli, aggiornarli o eliminarli.

Per ulteriori informazioni sui ruoli dei progetti, vedere [Come gestire accesso utente e approvazioni in Code Stream](#).

6 Fare clic su **Salva**.

Operazioni successive

Aggiungere gli endpoint e le pipeline che utilizzano il progetto. Vedere [Capitolo 6 Connessione di Code Stream agli endpoint](#) e [Capitolo 3 Creazione e utilizzo di pipeline in Code Stream](#).

Dopo aver creato una pipeline, il nome del progetto che raggruppa tutte le informazioni della pipeline viene visualizzato nelle schede della pipeline e nelle schede di esecuzione della pipeline.

Come gestire accesso utente e approvazioni in Code Stream

Code Stream offre diversi modi per assicurarsi che gli utenti dispongano dell'autorizzazione e del consenso appropriati per lavorare con le pipeline che rilasciano le applicazioni software.

A ciascun membro di un team è assegnato un ruolo, che fornisce autorizzazioni specifiche su pipeline, endpoint e dashboard e sulla possibilità di contrassegnare le risorse come limitate.

Le operazioni e le approvazioni utente consentono di controllare quando una pipeline viene eseguita e quando deve essere interrotta per l'approvazione. Il ruolo determina se è possibile riprendere una pipeline ed eseguire le pipeline che includono variabili o endpoint limitati.

Utilizzare le variabili segrete per nascondere e crittografare le informazioni riservate. Utilizzare una variabile limitata per le stringhe, le password e gli URL che devono essere nascosti e crittografati, nonché per limitare l'uso nelle esecuzioni. Ad esempio, utilizzare una variabile segreta per una password o un URL. È possibile utilizzare variabili segrete e limitate in qualsiasi tipo di attività nella pipeline.

Quali sono i ruoli di Code Stream

In base al ruolo di Code Stream, è possibile eseguire determinate azioni e accedere ad alcune aree. Ad esempio, un ruolo potrebbe consentire di creare, aggiornare ed eseguire pipeline, mentre un altro solo di visualizzarle.

Tutte le azioni ad eccezione delle limitazioni indica che questo ruolo dispone dell'autorizzazione a eseguire azioni di creazione, lettura, aggiornamento ed eliminazione su entità, ad eccezione di variabili ed endpoint limitati.

Tabella 2-1. Autorizzazioni di accesso a livello di progetto e servizio in Code Stream

Ruoli di Code Stream					
Livelli di accesso	Amministratore di Code Stream	Sviluppatore Code Stream	Esecutore Code Stream	Visualizzatore di Code Stream	Utente di Code Stream
Accesso a livello di servizio di Code Stream	Tutte le azioni	Tutte le azioni ad eccezione delle limitazioni	Azioni di esecuzione	Sola lettura	Nessuna
Accesso a livello di progetto: amministratore del progetto	Tutte le azioni	Tutte le azioni	Tutte le azioni	Tutte le azioni	Tutte le azioni
Accesso a livello di progetto: membro del progetto	Tutte le azioni	Tutte le azioni ad eccezione delle limitazioni	Tutte le azioni ad eccezione delle limitazioni	Tutte le azioni ad eccezione delle limitazioni	Tutte le azioni ad eccezione delle limitazioni
Accesso a livello di progetto: visualizzatore del progetto	Tutte le azioni	Tutte le azioni ad eccezione delle limitazioni	Azioni di esecuzione	Sola lettura	Sola lettura

Gli utenti che dispongono del ruolo di amministratore di progetto possono eseguire tutte le azioni sui progetti in cui sono amministratori di progetto.

Un amministratore del progetto può creare, leggere, aggiornare ed eliminare pipeline, variabili, endpoint, dashboard, trigger e avviare una pipeline che includa endpoint o variabili con limitazioni se queste risorse si trovano nel progetto in cui l'utente è un amministratore del progetto.

Gli utenti con il ruolo di visualizzatore del servizio possono visualizzare tutte le informazioni disponibili per l'amministratore. Non possono eseguire alcuna azione a meno che un amministratore non li renda amministratori del progetto o membri del progetto. Se l'utente è associato a un progetto, dispone delle autorizzazioni correlate al ruolo. Il visualizzatore del progetto non estenderebbe le proprie autorizzazioni nel modo in cui avviene per il ruolo di amministratore o membro. Questo ruolo è di sola lettura in tutti i progetti.

Se si dispone delle autorizzazioni di lettura in un progetto, è comunque possibile visualizzare le risorse limitate.

- Per visualizzare gli endpoint limitati, ovvero quelli con un'icona di blocco sulla scheda dell'endpoint, fare clic su **Configura > Endpoint**.
- Per visualizzare le variabili limitate e segrete, che vengono visualizzate come LIMITATE o SEGRETE nella colonna **Tipo**, fare clic su **Configura > Variabili**.

Tabella 2-2. Funzionalità dei ruoli di servizio di Code Stream

Contesto dell'interfaccia utente	Funzionalità	Ruolo di amministratore di Code Stream	Ruolo di sviluppatore di Code Stream	Ruolo di esecutore di Code Stream	Ruolo di visualizzatore di Code Stream	Ruolo di utente di Code Stream
Pipeline						
	Visualizzare pipeline	Sì	Sì	Sì	Sì	
	Creare pipeline	Sì	Sì			
	Eseguire pipeline	Sì	Sì	Sì		
	Eseguire le pipeline che includono variabili o endpoint limitati	Sì				
	Aggiornare pipeline	Sì	Sì			
	Eliminare pipeline	Sì	Sì			
Esecuzioni di pipeline						
	Visualizzare esecuzioni di pipeline	Sì	Sì	Sì	Sì	
	Riprendere, sospendere e annullare le esecuzioni delle pipeline	Sì	Sì	Sì		
	Riprendere le pipeline che vengono interrotte per l'approvazione su risorse limitate	Sì				
Integrazioni personalizzate						
	Creare integrazioni personalizzate	Sì	Sì			
	Leggere integrazioni personalizzate	Sì	Sì	Sì	Sì	

Tabella 2-2. Funzionalità dei ruoli di servizio di Code Stream (continua)

Contesto dell'interfaccia utente	Funzionalità	Ruolo di amministratore di Code Stream	Ruolo di sviluppatore di Code Stream	Ruolo di esecutore di Code Stream	Ruolo di visualizzatore di Code Stream	Ruolo di utente di Code Stream
	Aggiornare integrazioni personalizzate	Sì	Sì			
Endpoint						
	Visualizzare esecuzioni	Sì	Sì	Sì	Sì	
	Creare esecuzioni	Sì	Sì			
	Aggiornare esecuzioni	Sì	Sì			
	Eliminare esecuzioni	Sì	Sì			
Contrassegnare le risorse come limitate						
	Contrassegnare una variabile o un endpoint come limitati	Sì				
Dashboard						
	Visualizzare dashboard	Sì	Sì	Sì	Sì	
	Creare dashboard	Sì	Sì			
	Aggiornare dashboard	Sì	Sì			
	Eliminare dashboard	Sì	Sì			

Ruoli personalizzati e autorizzazioni in Code Stream

È possibile creare ruoli personalizzati in Cloud Assembly che estendono i privilegi degli utenti che utilizzano le pipeline. Quando si crea un ruolo personalizzato per le pipeline di Code Stream, si selezionano una o più autorizzazioni **Pipeline**.

Selezionare il numero minimo di autorizzazioni **Pipeline** necessarie per gli utenti a cui verrà assegnato questo ruolo personalizzato.

Quando un utente è assegnato a un progetto e ha un ruolo in tale progetto, se a tale utente viene assegnato un ruolo personalizzato che include una o più autorizzazioni **Pipeline**, può eseguire tutte le azioni consentite dalle autorizzazioni. Ad esempio, può creare variabili limitate, gestire pipeline limitate, creare e gestire integrazioni personalizzate e molto altro.

Tabella 2-3. Autorizzazioni della pipeline che è possibile assegnare a ruoli personalizzati

Autorizzazione pipeline	Amministratore di Code Stream	Sviluppatore Code Stream	Esecutore Code Stream	Visualizzatore di Code Stream	Utente di Code Stream	Amministratore del progetto	Membro del progetto	Visualizzatore del progetto
Gestisci pipeline	Sì	Sì				Sì	Sì	
Gestisci pipeline limitate	Sì					Sì		
Gestisci integrazioni personalizzate	Sì	Sì						
Esegui pipeline	Sì	Sì	Sì			Sì	Sì	
Esegui pipeline limitate	Sì					Sì		
Gestisci esecuzioni	Sì					Sì		
Lettura. Questa autorizzazione non è visibile.	Sì	Sì	Sì	Sì		Sì	Sì	Sì

Tabella 2-4. Come utilizzare le autorizzazioni Pipeline con i ruoli personalizzati

Autorizzazione	Operazioni che è possibile eseguire
Gestisci pipeline	<ul style="list-style-type: none"> ■ Creare, aggiornare, eliminare e clonare pipeline. ■ Rilasciare e annullare il rilascio di pipeline in VMware Service Broker. ■ Creare, aggiornare ed eliminare endpoint. ■ Creare, aggiornare ed eliminare le variabili regolari e segrete. ■ Creare, clonare, aggiornare ed eliminare un listener Gerrit. ■ Connettere e disconnettere un listener Gerrit. ■ Creare, clonare, aggiornare ed eliminare un trigger Gerrit. ■ Creare, aggiornare ed eliminare un webhook Git. ■ Creare, aggiornare ed eliminare un webhook Docker. ■ Utilizzare modelli di pipeline smart per creare pipeline. ■ Importare pipeline da YAML ed esportarle in YAML. ■ Creare, aggiornare ed eliminare dashboard personalizzati. ■ Leggere tutte le integrazioni personalizzate. ■ Leggere tutti gli endpoint e le variabili con limitazioni, ma non visualizzarne i valori.
Gestisci pipeline limitate	<ul style="list-style-type: none"> ■ Creare, aggiornare ed eliminare endpoint. ■ Contrassegnare gli endpoint come endpoint con restrizioni, aggiornare gli endpoint con restrizioni ed eliminarli. ■ Creare, aggiornare ed eliminare le variabili regolari e segrete. ■ Creare, aggiornare ed eliminare variabili con restrizioni. ■ Tutte le autorizzazioni che è possibile eseguire con Gestisci pipeline.
Gestisci integrazioni personalizzate	<ul style="list-style-type: none"> ■ Creare e aggiornare integrazioni personalizzate. ■ Assegnare la versione e rilasciare integrazioni personalizzate. ■ Eliminare e rendere obsolete le versioni delle integrazioni personalizzate. ■ Eliminare integrazioni personalizzate.
Esegui pipeline	<ul style="list-style-type: none"> ■ Eseguire pipeline. ■ Sospendere, riprendere e annullare esecuzioni delle pipeline. ■ Eseguire nuovamente esecuzioni delle pipeline. ■ Riprendere, eseguire nuovamente e attivare manualmente un evento di attivazione Gerrit. ■ Approvare un'operazione dell'utente ed eseguire l'approvazione in batch delle operazioni utente.

Tabella 2-4. Come utilizzare le autorizzazioni Pipeline con i ruoli personalizzati (continua)

Autorizzazione	Operazioni che è possibile eseguire
Esegui pipeline limitate	<ul style="list-style-type: none"> ■ Eseguire pipeline. ■ Sospendere, riprendere, annullare ed eliminare le esecuzioni di pipeline. ■ Eseguire nuovamente esecuzioni delle pipeline. ■ Sincronizzare l'esecuzione di una pipeline in esecuzione. ■ Forzare l'eliminazione dell'esecuzione di una pipeline in esecuzione. ■ Riprendere, eseguire nuovamente, eliminare e attivare manualmente un evento di attivazione Gerrit. ■ Risolvere gli elementi con restrizioni e continuare l'esecuzione della pipeline. ■ Cambiare il contesto dell'utente e continuare l'esecuzione della pipeline dopo l'approvazione dell'attività dell'operazione utente. ■ Tutte le autorizzazioni che è possibile eseguire con Esegui pipeline.
Gestisci esecuzioni	<ul style="list-style-type: none"> ■ Eseguire pipeline. ■ Sospendere, riprendere, annullare ed eliminare le esecuzioni di pipeline. ■ Eseguire nuovamente esecuzioni delle pipeline. ■ Riprendere, eseguire nuovamente, eliminare e attivare manualmente un evento di attivazione Gerrit. ■ Tutte le autorizzazioni che è possibile eseguire con Esegui pipeline.

I ruoli personalizzati possono includere combinazioni di autorizzazioni. Queste autorizzazioni sono organizzate in gruppi di funzionalità che permettono agli utenti di gestire o eseguire pipeline, con e senza risorse limitate. Queste autorizzazioni rappresentano tutte le funzionalità che ogni ruolo può eseguire in Code Stream.

Ad esempio, se si crea un ruolo personalizzato e si include l'autorizzazione denominata **Gestisci pipeline limitate**, gli utenti che dispongono del ruolo di sviluppatore di Code Stream possono:

- Creare, aggiornare ed eliminare endpoint.
- Contrassegnare gli endpoint come endpoint con restrizioni, aggiornare gli endpoint con restrizioni ed eliminarli.
- Creare, aggiornare ed eliminare le variabili regolari e segrete.
- Creare, aggiornare ed eliminare variabili con restrizioni.

Tabella 2-5. Combinazioni di esempio di autorizzazioni della pipeline nei ruoli personalizzati

Numero di autorizzazioni assegnate al ruolo personalizzato	Esempi di autorizzazioni combinate	Come utilizzare questa combinazione
Autorizzazione singola	Esegui pipeline	
Due autorizzazioni	Gestisci pipeline ed Esegui pipeline	
Tre autorizzazioni	Gestisci pipeline ed Esegui pipeline ed Esegui pipeline limitate	

Tabella 2-5. Combinazioni di esempio di autorizzazioni della pipeline nei ruoli personalizzati (continua)

Numero di autorizzazioni assegnate al ruolo personalizzato	Esempi di autorizzazioni combinate	Come utilizzare questa combinazione
	Gestisci pipeline e Gestisci integrazioni personalizzate ed Esegui pipeline limitate	Questa combinazione può essere applicata a un ruolo di sviluppatore di Code Stream, ma può essere limitata ai progetti di cui l'utente è membro.
	Gestisci pipeline e Gestisci integrazioni personalizzate e Gestisci esecuzioni	Questa combinazione può essere applicata a un amministratore di Code Stream, ma può essere limitata ai progetti di cui l'utente è membro.
	Gestisci pipeline, Gestisci pipeline limitate e Gestisci integrazioni personalizzate	Con questa combinazione, un utente dispone di autorizzazioni complete e può creare ed eliminare qualsiasi elemento in Code Stream.

Se si dispone del ruolo di amministratore

In qualità di amministratore, è possibile creare integrazioni, endpoint, variabili, trigger, pipeline e dashboard personalizzati.

I progetti consentono alle pipeline di accedere alle risorse dell'infrastruttura. Gli amministratori creano progetti in modo che gli utenti possano raggruppare pipeline, endpoint e dashboard. Gli utenti selezionano quindi il progetto nelle loro pipeline. Ogni progetto include un amministratore e utenti con ruoli assegnati.

Con il ruolo di amministratore, è possibile contrassegnare gli endpoint e le variabili come risorse limitate, nonché eseguire pipeline che utilizzano risorse limitate. Se un utente non amministratore esegue la pipeline che include una variabile o un endpoint con limitazioni, la pipeline si arresta in corrispondenza dell'attività in cui viene utilizzata la variabile limitata e un amministratore deve riprendere la pipeline.

Come amministratore, è possibile anche richiedere che le pipeline vengano pubblicate in vRealize Automation Service Broker.

Se si dispone del ruolo di sviluppatore

È possibile eseguire le stesse operazioni di un amministratore con le pipeline, tranne quelle che coinvolgono variabili o endpoint limitati.

Se si esegue una pipeline che utilizza variabili o endpoint limitati, la pipeline viene eseguita solo fino all'attività che utilizza la risorsa limitata. Quindi, si arresta e un amministratore del progetto o un amministratore di Code Stream deve riprendere la pipeline.

Se si dispone del ruolo di utente

È possibile accedere a Code Stream, ma non si dispone dei privilegi forniti dagli altri ruoli.

Se si dispone del ruolo di visualizzatore

È possibile visualizzare le stesse risorse visualizzate da un amministratore, ad esempio pipeline, endpoint, esecuzioni di pipeline, dashboard, integrazioni personalizzate e trigger, ma non è possibile crearle, aggiornarle o eliminarle. Per eseguire le azioni, al ruolo Visualizzatore deve essere assegnato anche il ruolo di amministratore del progetto o membro del progetto.

Gli utenti che dispongono del ruolo Visualizzatore possono visualizzare i progetti. Possono inoltre visualizzare endpoint limitati e variabili limitate, ma non possono visualizzare informazioni dettagliate su endpoint e variabili.

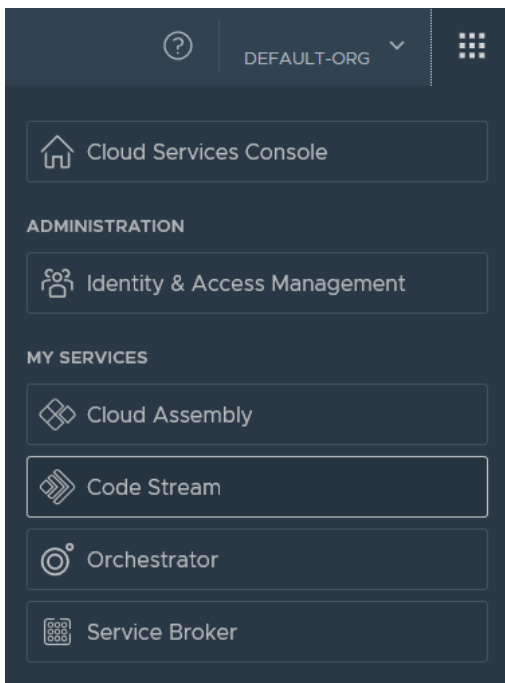
Se si dispone del ruolo di esecutore

È possibile eseguire pipeline e azioni per le attività Operazione utente. È inoltre possibile riprendere, sospendere e annullare le esecuzioni della pipeline. Tuttavia, non è possibile modificare le pipeline.

Come assegnare e aggiornare i ruoli

Per assegnare e aggiornare i ruoli per gli altri utenti, è necessario essere un amministratore.

- 1 Per visualizzare gli utenti attivi e i loro ruoli, in vRealize Automation, fare clic sui nove punti nella parte superiore destra.
- 2 Fare clic su **Gestione identità e accessi**.



- 3 Per visualizzare ruoli e nomi utente, fare clic su **Utenti attivi**.



- 4 Per aggiungere ruoli per un utente o modificarne i ruoli, fare clic sulla casella di controllo accanto al nome utente e quindi su **Modifica ruoli**.
- 5 Quando si aggiungono o si modificano i ruoli utente, è possibile aggiungere anche l'accesso ai servizi.
- 6 Per salvare le modifiche, fare clic su **Salva**.

Che cosa sono le operazioni utente e le approvazioni in Code Stream

L'area Operazioni utente visualizza le esecuzioni della pipeline che richiedono l'approvazione. L'approvatore richiesto può approvare o rifiutare l'esecuzione della pipeline.

Quando si crea una pipeline, potrebbe essere necessario aggiungervi un'approvazione se:

- Un membro del team deve esaminare il codice.
- Un altro utente deve confermare un artefatto di creazione.
- È necessario assicurarsi che tutti i test siano stati completati.
- Un'attività utilizza una risorsa contrassegnata come limitata da un amministratore e l'attività richiede l'approvazione.
- La pipeline rilascerà il software alla produzione.

Per determinare se approvare un'attività della pipeline, l'approvatore richiesto deve disporre dell'autorizzazione e dell'esperienza necessari.

Quando si aggiunge un'attività Operazione utente, è possibile impostare il timeout di scadenza in termini di giorni, ore o minuti. Ad esempio, potrebbe essere richiesta l'approvazione della pipeline da parte dell'utente entro 30 minuti. Se l'approvazione non giunge entro 30 minuti, la pipeline non riuscirà come previsto.

Se si abilita l'invio delle notifiche tramite e-mail, l'attività Operazione utente invia notifiche solo agli approvatori con indirizzi e-mail completi e non ai nomi di approvatori che non sono in formato e-mail.

Dopo che l'utente necessario ha approvato l'attività:

- L'esecuzione della pipeline in sospeso può continuare.

- Quando la pipeline continua, tutte le richieste in sospeso precedenti per l'approvazione della stessa attività Operazione utente vengono annullate.

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[redacted]om
<input type="checkbox"/>	a0a990	Demo2-Jenkins-K8s#6	Testing	fritz	Nov 11, 2019, 1:34:11 PM	k[redacted]om, f[redacted]m
<input checked="" type="checkbox"/>	User Operation #8f1728 Request Details Execution: Demo-Jenkins-K8s #5 Summary: Testing Approvers: k[redacted]om, f[redacted]om Requested By: fritz Requested On: Nov 11, 2019, 1:22:21 PM Expires On: Nov 14, 2019, 1:22:21 PM					

1

Items per page 20 1 - 7 of 7 items

Nell'area Operazioni utente, gli elementi da approvare o rifiutare vengono visualizzati come elementi attivi o inattivi. Ogni elemento è mappato a un'attività Operazione utente in una pipeline.

- Gli **elementi attivi** attendono l'approvatore che deve rivedere l'attività, affinché la approvino o la rifiutino. Se si è un utente presente nell'elenco di approvatori, è possibile espandere la riga Operazione utente e fare clic su **Accetta** o **Rifiuta**.
- Gli **elementi inattivi** sono stati approvati o rifiutati. Se un utente ha rifiutato l'operazione utente o se l'approvazione dell'attività è scaduta, tale attività non può più essere approvata.

N. indice è una stringa alfanumerica di sei caratteri univoca che è possibile utilizzare come filtro per cercare una particolare approvazione.

Le approvazioni della pipeline vengono visualizzate anche nell'area **Esecuzioni**.

- Lo stato delle pipeline che richiedono l'approvazione è in attesa.
- Gli altri stati sono in coda, completata e non riuscita.
- Se la pipeline è in uno stato di attesa, l'approvatore necessario deve approvare l'attività della pipeline.

Creazione e utilizzo di pipeline in Code Stream

3

È possibile utilizzare vRealize Automation Code Stream per modellare il processo di creazione, test e distribuzione. Con vRealize Automation Code Stream, è possibile configurare l'infrastruttura che supporta il ciclo di rilascio e creare pipeline che modellano le attività di rilascio del software. vRealize Automation Code Stream fornisce il software dal codice di sviluppo al testing e lo distribuisce alle istanze di produzione.

Ogni pipeline include fasi e attività. Le fasi rappresentano le fasi di sviluppo e le attività eseguono le azioni necessarie per consegnare l'applicazione software attraverso le fasi.

Che cosa sono le pipeline in vRealize Automation Code Stream

Una pipeline è un modello di integrazione e consegna continue del processo di rilascio del software. Rilascia il software dal codice sorgente, attraverso il testing, alla produzione. Include una sequenza di fasi con attività che rappresentano le attività nel ciclo di rilascio del software. L'applicazione software passa da una fase all'altra attraverso la pipeline.

È possibile aggiungere endpoint in modo che le attività della pipeline possano connettersi a origini dati, repository o sistemi di notifica.

Creazione di pipeline

È possibile creare una pipeline iniziando con una tela vuota, utilizzando un modello di pipeline smart o importando codice YAML.

- Utilizzare la tela vuota. Per un esempio, vedere [Pianificazione di una creazione nativa di integrazione e consegna continue in Code Stream prima dell'aggiunta manuale di attività](#).
- Utilizzare un modello di pipeline smart. Per un esempio, vedere [Capitolo 4 Pianificazione di creazione, integrazione e consegna native del codice in Code Stream](#).
- Importare il codice YAML. Fare clic su **Pipeline > Importa**. Nella finestra di dialogo **Importa**, selezionare il file YAML o immettere il codice YAML, quindi fare clic su **Importa**.

Quando si utilizza la tela vuota per creare una pipeline, è possibile aggiungere fasi, attività e approvazioni. La pipeline automatizza il processo di che crea, testa, distribuisce e rilascia l'applicazione. Le attività in ogni fase eseguono azioni che creano, testano e rilasciano il codice in ogni fase.

Tabella 3-1. Fasi e utilizzi delle pipeline di esempio

Fase di esempio	Esempi di operazioni che è possibile eseguire
Sviluppo	<p>In una fase di sviluppo, è possibile eseguire il provisioning di una macchina, recuperare un artefatto o aggiungere un'attività di creazione per creare un host Docker per l'integrazione continua del codice e molto altro.</p> <p>Ad esempio:</p> <ul style="list-style-type: none"> ■ Per pianificare e creare una build di integrazione continua (CI), che fornisca il codice utilizzando la funzionalità di creazione nativa in vRealize Automation Code Stream, vedere Pianificazione di una creazione nativa di integrazione continua in Code Stream prima di utilizzare il modello di pipeline smart.
Test	<p>In una fase di test, è possibile aggiungere un'attività Jenkins per testare l'applicazione software, nonché includere strumenti di test post-elaborazione come JUnit e JaCoCo e molto altro.</p> <p>Ad esempio:</p> <ul style="list-style-type: none"> ■ Integrare vRealize Automation Code Stream con Jenkins ed eseguire un processo Jenkins nella pipeline per creare e testare il codice sorgente. Vedere Come integrare Code Stream con Jenkins. ■ Creare script personalizzati che estendano la capacità di vRealize Automation Code Stream di integrarsi con gli strumenti di creazione, test e distribuzione. Vedere Integrazione degli strumenti di creazione, test e distribuzione con Code Stream. ■ Monitorare le tendenze nella post-elaborazione per una pipeline di integrazione continua (CI). Vedere Come utilizzare dashboard personalizzati per tenere traccia degli indicatori di prestazioni chiave per la pipeline in Code Stream.
Produzione	<p>In una fase di produzione è possibile eseguire l'integrazione con il modello cloud in Cloud Assembly che esegue il provisioning dell'infrastruttura, distribuisce il software in un cluster Kubernetes e molto altro.</p> <p>Ad esempio:</p> <ul style="list-style-type: none"> ■ Per visualizzare le fasi di esempio per lo sviluppo e la produzione, che possono distribuire l'applicazione software nel modello di distribuzione blu-verde, vedere Come distribuire un'applicazione in Code Stream nella distribuzione blu-verde. ■ Per integrare un modello cloud nella pipeline, vedere Come automatizzare il rilascio di un'applicazione da distribuire da un modello cloud YAML in Code Stream. È inoltre possibile aggiungere un'attività di distribuzione che esegua uno script per distribuire l'applicazione. ■ Per automatizzare la distribuzione delle applicazioni software a un cluster Kubernetes, Come automatizzare il rilascio di un'applicazione in Code Stream in un cluster Kubernetes. ■ Per integrare il codice nella pipeline e distribuire l'immagine della build, vedere Come integrare il codice in modo continuativo dal repository GitHub o GitLab nella pipeline in Code Stream.

È possibile esportare la pipeline come file YAML. Fare clic su **Pipeline**, fare clic su una scheda di pipeline, quindi fare clic su **Azioni > Esporta**.

Approvazione delle pipeline

È possibile ottenere un'approvazione da un altro membro del team in punti specifici della pipeline.

- Per richiedere l'approvazione in una pipeline includendo un'attività Operazione utente in una pipeline, vedere [Come eseguire una pipeline e visualizzare i risultati](#). Questa attività invia una notifica e-mail all'utente che deve rivederla. Prima che sia possibile continuare l'esecuzione della pipeline, il revisore deve concedere l'approvazione o rifiutarla. Se l'attività Operazione utente ha un timeout di scadenza impostato in giorni, ore o minuti, l'utente richiesto deve approvare la pipeline prima che l'attività scada. In caso contrario, la pipeline non riuscirà come previsto.
- In qualsiasi fase di una pipeline, se un'attività o una fase non riesce, è possibile creare un ticket Jira con vRealize Automation Code Stream. Vedere [Come creare un ticket Jira in Code Stream quando un'attività della pipeline non riesce](#).

Attivazione delle pipeline

Le pipeline possono essere attivate quando gli sviluppatori archiviano il proprio codice nel repository o rivedono il codice o quando rilevano un artefatto della build nuovo o aggiornato.

- Per integrare vRealize Automation Code Stream con il ciclo di vita Git e attivare una pipeline quando gli sviluppatori aggiornano il proprio codice, utilizzare il trigger Git. Vedere [Come utilizzare il trigger Git in Code Stream per eseguire una pipeline](#).
- Per integrare vRealize Automation Code Stream con il ciclo di vita di revisione del codice Gerrit e attivare una pipeline nelle revisioni del codice, utilizzare il trigger Gerrit. Vedere [Come utilizzare il trigger Gerrit in Code Stream per eseguire una pipeline](#).
- Per attivare una pipeline quando viene creato o aggiornato un artefatto della build Docker, utilizzare il trigger Docker. Vedere [Come utilizzare il trigger Docker in Code Stream per eseguire una pipeline di consegna continua](#).

Per ulteriori informazioni sui trigger supportati da vRealize Automation Code Stream, vedere [Capitolo 7 Attivazione di pipeline in Code Stream](#).

Questo capitolo include i seguenti argomenti:

- [Come eseguire una pipeline e visualizzare i risultati](#)
- [Tipi di attività disponibili in Code Stream](#)
- [Come utilizzare i binding delle variabili nelle pipeline di Code Stream](#)
- [Come utilizzare le associazioni di variabili in un'attività di condizione per eseguire o arrestare una pipeline in Code Stream](#)
- [Variabili ed espressioni che è possibile utilizzare quando si associano le attività della pipeline in Code Stream](#)
- [Come inviare notifiche sulla pipeline in Code Stream](#)

- [Come creare un ticket Jira in Code Stream quando un'attività della pipeline non riesce](#)
- [Come eseguire il rollback di una distribuzione in Code Stream](#)

Come eseguire una pipeline e visualizzare i risultati

È possibile eseguire una pipeline dalla scheda della pipeline, in modalità di modifica della pipeline, e dall'esecuzione della pipeline. È inoltre possibile utilizzare i trigger disponibili affinché Code Stream esegua una pipeline quando si verificano determinati eventi.

Quando tutte le fasi e le attività della pipeline sono valide, la pipeline è pronta per il rilascio, l'esecuzione o l'attivazione.

Per eseguire o attivare la pipeline utilizzando Code Stream, è possibile abilitare ed eseguire la pipeline dalla scheda della pipeline o mentre si è nella pipeline. Quindi, è possibile visualizzare l'esecuzione della pipeline per verificare che la pipeline abbia creato, testato e distribuito il codice.

Quando è in corso, l'esecuzione di una pipeline può essere eliminata sia da un amministratore sia da un utente non amministratore.

- Amministratore: per eliminare l'esecuzione di una pipeline quando è in corso, fare clic su **Esecuzioni**. Nell'esecuzione da eliminare, fare clic su **Azioni > Elimina**.
- Utente non amministratore: per eliminare l'esecuzione di una pipeline in corso, fare clic su **Esecuzioni** e premere **Alt Shift d**.

Quando l'esecuzione di una pipeline è in corso e sembra essere bloccata, un amministratore può aggiornare l'esecuzione dalla pagina Esecuzioni o dalla pagina Dettagli esecuzione.

- Pagina Esecuzioni: fare clic su **Esecuzioni**. Nell'esecuzione da aggiornare, fare clic su **Azioni > Sincronizza**.
- Pagina Dettagli esecuzione: fare clic su **Esecuzioni**, fare clic sul collegamento ai dettagli dell'esecuzione e scegliere **Azioni > Sincronizza**.

Per eseguire una pipeline al verificarsi di un evento specifico, utilizzare i trigger.

- Il trigger Git può eseguire una pipeline quando gli sviluppatori aggiornano il codice.
- Il trigger Gerrit può eseguire una pipeline quando si verificano revisioni del codice.
- Il trigger Docker può eseguire una pipeline quando viene creato un artefatto in un registro di Docker.
- Il comando `curl` o il comando `wget` possono fare in modo che Jenkins esegua una pipeline al termine di una creazione di Jenkins.

Per ulteriori informazioni sull'utilizzo dei trigger, vedere [Capitolo 7 Attivazione di pipeline in Code Stream](#).

La procedura seguente illustra come eseguire una pipeline dalla scheda della pipeline, visualizzare le esecuzioni, visualizzare i dettagli delle esecuzioni e utilizzare le azioni. Viene inoltre illustrato come rilasciare una pipeline in modo che sia possibile aggiungerla a vRealize Automation Service Broker.

Prerequisiti

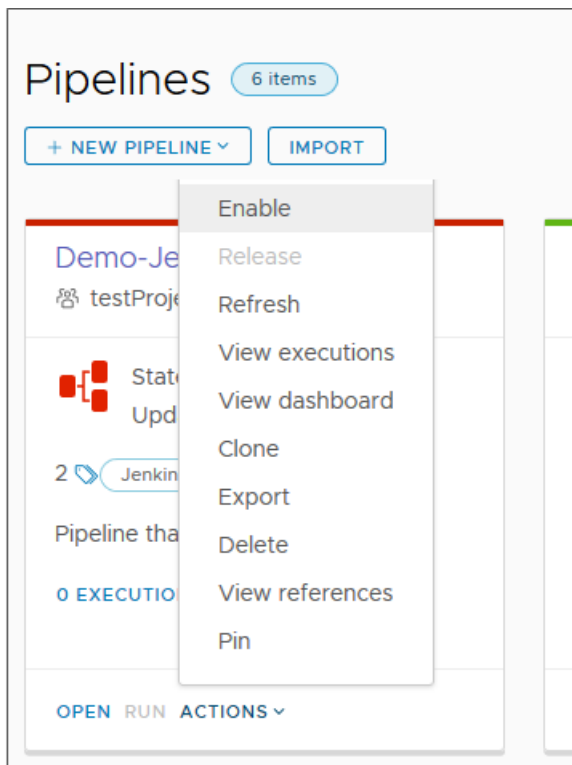
- Verificare che vengano create una o più pipeline. Vedere gli esempi in [Capitolo 5 Tutorial sull'utilizzo di Code Stream](#).

Procedura

- 1 Abilitare la pipeline.

Per eseguire o rilasciare una pipeline, è innanzitutto necessario abilitarla.

- a Fare clic su **Pipeline**.
- b Nella scheda della pipeline, fare clic su **Azioni > Abilita**.



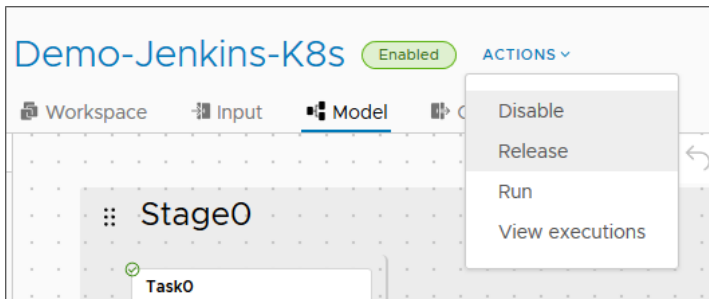
È inoltre possibile abilitare la pipeline quando si è nella pipeline. Se la pipeline è già abilitata, il comando **Esegui** è attivo e il menu **Azioni** include il comando **Disabilita**.

2 (Facoltativo) Rilasciare la pipeline.

Se si desidera rendere la pipeline disponibile come elemento del catalogo in vRealize Automation Service Broker, è necessario rilasciarla in Code Stream.

- a Fare clic su **Pipeline**.
- b Nella scheda della pipeline, fare clic su **Azioni > Rilascia**.

È inoltre possibile rilasciare la pipeline quando si è nella pipeline.



Dopo aver rilasciato la pipeline, aprire Service Broker per aggiungere la pipeline come elemento del catalogo ed eseguirla. Vedere [Aggiunta di pipeline di Code Stream al catalogo di Service Broker](#).

Nota Se l'esecuzione della pipeline richiede più di 120 minuti, specificare un tempo di esecuzione approssimativo come valore di timeout della richiesta. Per impostare o rivedere il timeout della richiesta per un progetto, aprire Service Broker come amministratore e selezionare **Infrastruttura > Progetti**. Fare clic sul nome del progetto e quindi su **Provisioning**.

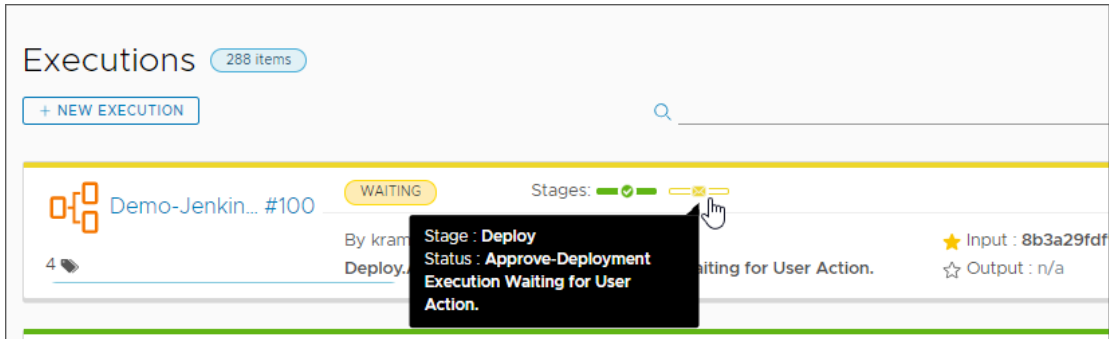
Se il valore di timeout della richiesta non è impostato, un'esecuzione che richiede più di 120 minuti viene visualizzata come non riuscita con un errore di timeout della richiesta di callback. Ciò non influisce tuttavia sull'esecuzione della pipeline.

- 3 Nella scheda della pipeline fare clic su **Esegui**.
- 4 Per visualizzare la pipeline durante l'esecuzione, fare clic su **Esecuzioni**.

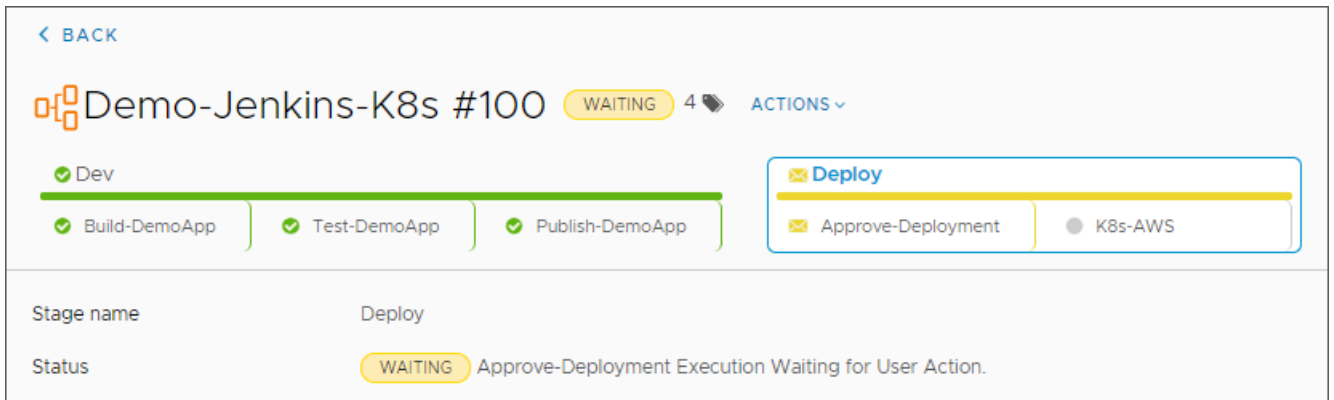
La pipeline esegue ogni fase in sequenza e l'esecuzione della pipeline mostra un'icona di stato per ogni fase. Se la pipeline include un'attività Operazione utente, l'utente deve approvare l'attività affinché l'esecuzione della pipeline possa continuare. Quando viene utilizzata un'attività Operazione utente, la pipeline viene interrotta e attende che l'utente richiesto approvi l'attività.

Ad esempio, è possibile utilizzare l'attività Operazione utente per approvare la distribuzione del codice in un ambiente di produzione.

Se l'attività Operazione utente ha un timeout di scadenza impostato in giorni, ore o minuti, l'utente richiesto deve approvare la pipeline prima che l'attività scada. In caso contrario, la pipeline non riuscirà come previsto.



- 5 Per visualizzare la fase della pipeline in attesa di approvazione dell'utente, fare clic sull'icona di stato della fase.

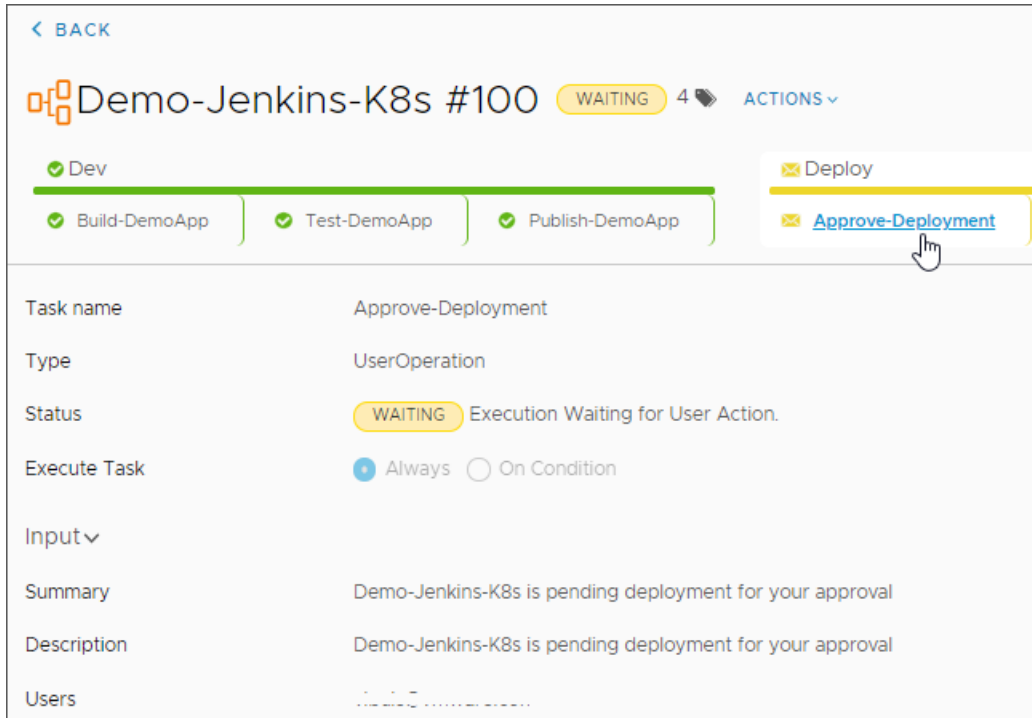


- 6 Per visualizzare i dettagli dell'attività, fare clic sull'attività.

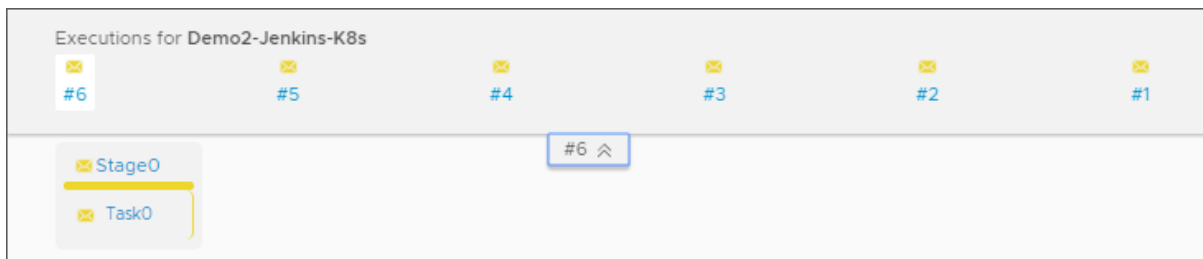
Dopo che l'utente richiesto approva l'attività, un utente con il ruolo appropriato deve riprendere la pipeline. Per i ruoli necessari, vedere [Come gestire accesso utente e approvazioni in Code Stream](#).

Se un'esecuzione non riesce, è necessario valutare e correggere la causa dell'errore. Passare quindi all'esecuzione e fare clic su **Azioni > Esegui nuovamente**.

È possibile riprendere le esecuzioni primarie delle pipeline e le esecuzioni nidificate.



- 7 Dall'esecuzione della pipeline, è possibile fare clic su **Azioni** per visualizzare la pipeline e selezionare un'azione come **Sospendi**, **Annulla** e molte altre. Quando l'esecuzione di una pipeline è in corso, un amministratore può eliminare o sincronizzare l'esecuzione della pipeline. Un utente non amministratore può eliminare una pipeline in esecuzione.
- 8 Per passare facilmente da un'esecuzione all'altra e visualizzare i dettagli di un'attività, fare clic su **Esecuzioni**, quindi fare clic su un'esecuzione di pipeline. Quindi, fare clic sulla scheda nella parte superiore e selezionare l'esecuzione della pipeline.



Risultati

Congratulazioni! È stata eseguita una pipeline, è stata esaminata l'esecuzione della pipeline ed è stata visualizzata un'attività Operazione utente che richiedeva l'approvazione per continuare l'esecuzione della pipeline. È stato inoltre utilizzato il menu **Azioni** nell'esecuzione della pipeline per tornare al modello della pipeline e apportare le modifiche necessarie.

Operazioni successive

Per ulteriori informazioni sull'utilizzo di Code Stream per automatizzare il ciclo di rilascio del software, vedere [Capitolo 5 Tutorial sull'utilizzo di Code Stream](#).

Tipi di attività disponibili in Code Stream

Quando si configura la pipeline, si aggiungono tipi di attività specifici che vengono eseguiti per le azioni necessarie. Ogni tipo di attività si integra con un'altra applicazione e abilita la pipeline quando crea, testa e consegna le applicazioni.

Per eseguire la pipeline, quando è necessario estrarre gli elementi da un repository per la distribuzione, eseguire uno script remoto o richiedere l'approvazione per un'operazione utente da un membro del team, Code Stream offre il tipo di attività richiesta.

Code Stream supporta l'annullamento dell'esecuzione di una pipeline su vari tipi di attività. Quando si fa clic su **Annulla** durante l'esecuzione di una pipeline, l'attività, la fase o l'intera pipeline passa allo stato di annullamento e annulla l'esecuzione della pipeline.

Code Stream consente di annullare l'esecuzione della pipeline in un'attività, in staging o nell'intera pipeline quando si utilizzano queste attività:

- Jenkins
- SSH
- PowerShell
- Operazione utente
- Pipeline
- Modello cloud
- vRO
- Polling

Code Stream non propaga il comportamento di annullamento ai sistemi di terze parti per queste attività: CI, Integrazione personalizzata o Kubernetes. Code Stream contrassegna l'attività come annullata e interrompe immediatamente il recupero dello stato senza attendere il completamento dell'attività. L'attività potrebbe essere completata o non riuscire nel sistema di terze parti, ma smette immediatamente di funzionare in Code Stream quando si fa clic su **Annulla**.

Prima di utilizzare un'attività nella pipeline, verificare che sia disponibile l'endpoint corrispondente.

Tabella 3-2. Ottenere un'approvazione o impostare un punto di decisione

Tipo di attività	Funzioni	Esempi e dettagli
Operazione utente	Un'attività Operazione utente abilita un'approvazione richiesta che controlla quando una pipeline viene eseguita e quando deve essere interrotta per l'approvazione.	Vedere Come eseguire una pipeline e visualizzare i risultati . e Come gestire accesso utente e approvazioni in Code Stream .
Condizione	Aggiunge un punto di decisione, che determina se la pipeline continua a essere eseguita o si interrompe, in base alle espressioni di condizione. Quando la condizione è true, la pipeline esegue le attività successive. Quando è false, la pipeline si interrompe.	Vedere Come utilizzare le associazioni di variabili in un'attività di condizione per eseguire o arrestare una pipeline in Code Stream .

Tabella 3-3. Automatizzare l'integrazione e la distribuzione continue

Tipo di attività	Funzioni	Esempi e dettagli
Modello cloud	Distribuisce un modello di cloud di automazione da GitHub, esegue il provisioning di un'applicazione e automatizza l'integrazione e la consegna continue (CICD) di tale modello di cloud per la distribuzione.	<p>Vedere Come automatizzare il rilascio di un'applicazione da distribuire da un modello cloud YAML in Code Stream.</p> <p>I parametri del modello cloud vengono visualizzati dopo aver prima selezionato Crea o Aggiorna e quindi aver selezionato Modello cloud e Versione. È possibile aggiungere questi elementi, che ospitano associazioni di variabili, alle aree di testo di input nell'attività del modello di cloud:</p> <ul style="list-style-type: none"> ■ Intero ■ Stringa di enumerazione ■ Booleano ■ Variabile array <p>Quando si utilizza l'associazione di variabili nell'input, tenere conto di tali eccezioni. Per le enumerazioni, è necessario selezionare un valore di enumerazione da un set fisso. Per i valori booleani, è necessario immettere il valore nell'area di testo di input.</p> <p>Il parametro del modello di cloud viene visualizzato nell'attività del modello di cloud quando un modello di cloud in Cloud Assembly include variabili di input. Ad esempio, se un modello di cloud dispone di un tipo di input di <code>Integer</code>, è possibile immettere il numero intero direttamente o come variabile utilizzando l'associazione di variabili.</p>
CI	<p>L'attività CI consente l'integrazione continua del codice nella pipeline estraendo un'immagine della build di Docker da un endpoint del registro e distribuendo il codice in un cluster Kubernetes.</p> <p>L'attività CI mostra 100 righe del registro come output e 500 righe quando si scaricano i registri.</p> <p>Le attività CI richiedono porte non attive da 32768 a 61000.</p>	Vedere Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart .
Personalizzato	L'attività personalizzata integra Code Stream con i propri strumenti di creazione, test e distribuzione.	Vedere Integrazione degli strumenti di creazione, test e distribuzione con Code Stream .

Tabella 3-3. Automatizzare l'integrazione e la distribuzione continue (continua)

Tipo di attività	Funzioni	Esempi e dettagli
Kubernetes	Automatizza la distribuzione delle applicazioni software in cluster Kubernetes in AWS.	Vedere Come automatizzare il rilascio di un'applicazione in Code Stream in un cluster Kubernetes .
Pipeline	<p>Nidifica una pipeline in una pipeline primaria. Quando una pipeline è nidificata, si comporta come un'attività nella pipeline primaria.</p> <p>Nella scheda Attività della pipeline primaria, è possibile passare facilmente alla pipeline nidificata facendo clic sul relativo collegamento. La pipeline nidificata viene aperta in una nuova scheda del browser.</p>	Per trovare le pipeline nidificate in Esecuzioni , immettere nested nell'area di ricerca.

Tabella 3-4. Integrare applicazioni di sviluppo, test e distribuzione

Tipo di attività...	Che cosa fa...	Esempi e dettagli...
Bamboo	Interagisce con un server di integrazione continua (CI) di Bamboo che crea, testa e integra in modo continuativo il software come preparazione alla distribuzione e attiva le build di codice quando gli sviluppatori confermano le modifiche. Espone le posizioni degli artefatti che la creazione di Bamboo produce in modo che l'attività possa generare parametri utilizzabili da altre attività per la creazione e la distribuzione.	Connettersi a un endpoint del server Bamboo e avviare un piano di creazione Bamboo dalla pipeline.
Jenkins	Attiva i processi Jenkins, che generano e testano il codice sorgente, esegue test case e può utilizzare script personalizzati.	Vedere Come integrare Code Stream con Jenkins .
TFS	Consente di connettere la pipeline a Team Foundation Server per gestire e richiamare i progetti di creazione, inclusi i processi configurati che creano e testano il codice.	Per le versioni di Team Foundation Server supportate da Code Stream, vedere Che cosa sono gli endpoint in Code Stream .
vRO	<p>Estende la funzionalità di Code Stream mediante l'esecuzione di workflow predefiniti o personalizzati in vRealize Orchestrator.</p> <p>Code Stream supporta l'autenticazione di base e l'autenticazione basata su token per vRealize Orchestrator. Code Stream supporta il token dell'API per autenticare e convalidare il cluster vRealize Orchestrator. Con l'autenticazione basata su token, Code Stream supporta endpoint vRealize Orchestrator che usano un Proxy di estensibilità cloud. Di conseguenza, in Code Stream è possibile attivare workflow con un endpoint vRealize Orchestrator che usa il proxy di estensibilità cloud.</p>	Vedere Come integrare Code Stream con vRealize Orchestrator .

Tabella 3-5. Integrare altre applicazioni tramite un'API

Tipo di attività...	Che cosa fa...	Esempi e dettagli...
REST	Integra Code Stream con altre applicazioni che utilizzano una REST API per consentire lo sviluppo e la consegna continui di applicazioni software che interagiscono tra loro.	Vedere Come utilizzare una REST API per integrare Code Stream con altre applicazioni .
Polling	<p>Richiama una REST API e ne esegue il polling finché l'attività della pipeline non soddisfa i criteri di uscita e viene completata.</p> <p>Un amministratore di Code Stream può impostare il conteggio di polling fino a un massimo di 10000. L'intervallo di polling deve essere maggiore o uguale a 60 secondi.</p> <p>Quando si contrassegna la casella di controllo Continua in caso di errore, se il conteggio o l'intervallo superano questi valori, l'esecuzione dell'attività di polling continua.</p> <p>POLL Iteration Count: viene visualizzato nell'esecuzione della pipeline e viene mostrato il numero di volte in cui l'attività POLL ha richiesto una risposta dall'URL. Ad esempio, se l'input POLL è 65 e il numero effettivo di esecuzioni della richiesta POLL è 4, il conteggio delle iterazioni nell'output di esecuzione della pipeline mostra 4 (su 65).</p>	Vedere Come utilizzare una REST API per integrare Code Stream con altre applicazioni .

Tabella 3-6. Eseguire script remoti e definiti dall'utente

Tipo di attività	Funzioni	Esempi e dettagli
PowerShell	<p>Con l'attività PowerShell, Code Stream può eseguire comandi di script su un host remoto. Ad esempio, uno script può automatizzare le attività di test ed eseguire i tipi di comandi amministrativi.</p> <p>Lo script può essere remoto o definito dall'utente. Può connettersi tramite HTTP o HTTPS e può utilizzare TLS.</p> <p>Nell'host Windows deve essere configurato il servizio winrm, e in winrm devono essere configurati MaxShellsPerUser e MaxMemoryPerShellMB.</p> <p>Per eseguire un'attività di PowerShell, è necessario disporre di una sessione attiva nell'host Windows remoto.</p> <p>Lunghezza della riga di comando di PowerShell</p> <p>Se si immette un comando PowerShell base64, tenere presente che è necessario calcolare la lunghezza complessiva del comando.</p> <p>La pipeline di Code Stream codifica ed esegue il wrapping di un comando PowerShell base64 in un altro comando, incrementando la lunghezza complessiva del comando.</p> <p>La lunghezza massima consentita per un comando di winrm PowerShell è 8192 byte. Il limite di lunghezza del comando è inferiore per l'attività PowerShell quando viene codificata e ne viene eseguito il wrapping. Di conseguenza, è necessario calcolare la lunghezza del comando prima di immettere il comando PowerShell.</p> <p>Il limite di lunghezza del comando per l'attività PowerShell di Code Stream dipende dalla lunghezza codificata in base64 del comando originale. La lunghezza del comando viene calcolata come segue.</p> <pre>3 * (length of original command / 4) - (numberOfPaddingCharacters) + 77 (Length of Write-output command)</pre> <p>La lunghezza del comando per Code Stream deve essere inferiore al limite massimo di 8192.</p>	<p>Quando si configurano MaxShellsPerUser e MaxMemoryPerShellMB:</p> <ul style="list-style-type: none"> ■ Il valore accettabile per MaxShellsPerUser è 500 per 50 pipeline simultanee, con 5 attività PowerShell per ogni pipeline. Per impostare il valore, eseguire: winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}' ■ Il valore della memoria accettabile per MaxMemoryPerShellMB è 2048. Per impostare il valore, eseguire: winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}' <p>Lo script scrive l'output in un file di risposta, che può essere utilizzato da un'altra pipeline.</p>
SSH	<p>L'attività SSH consente all'attività di script della shell Bash di eseguire comandi di script su un host remoto. Ad esempio, uno script può automatizzare le attività di test ed eseguire i tipi di comandi amministrativi.</p> <p>Lo script può essere remoto o definito dall'utente. Può connettersi tramite HTTP o HTTPS e richiede una chiave privata o una password.</p>	<p>Lo script può essere remoto o definito dall'utente. Ad esempio, uno script può essere simile al seguente:</p> <pre>message="Hello World" echo \$message</pre> <p>Lo script scrive l'output in un file di risposta, che può essere utilizzato da un'altra pipeline.</p>

Tabella 3-6. Eseguire script remoti e definiti dall'utente (continua)

Tipo di attività	Funzioni	Esempi e dettagli
	<p>Il servizio SSH deve essere configurato sull'host Linux e la configurazione SSHD di <code>MaxSessions</code> deve essere impostata su 50.</p> <p>Se si eseguono molte attività SSH contemporaneamente, aumentare <code>MaxSessions</code> e <code>MaxOpenSessions</code> nell'host SSH. Non utilizzare l'istanza di vRealize Automation come host SSH se è necessario modificare le impostazioni di configurazione di <code>MaxSessions</code> e <code>MaxOpenSessions</code>.</p>	

Come utilizzare i binding delle variabili nelle pipeline di Code Stream

Eseguire il binding di un'attività della pipeline significa creare una dipendenza per l'attività quando la pipeline è in esecuzione. È possibile creare un binding per un'attività della pipeline in diversi modi. È possibile associare un'attività a un'altra attività, associarla a una variabile e a un'espressione oppure associarla a una condizione.

Come applicare binding dollaro alle variabili del modello cloud in un'attività del modello cloud

È possibile applicare binding dollaro alle variabili del modello cloud in un'attività del modello cloud della pipeline di Code Stream. Le modalità di modifica delle variabili in Code Stream dipendono dalla codifica delle proprietà delle variabili nel modello cloud.

Se è necessario utilizzare binding dollaro in un'attività del modello cloud, ma la versione corrente del modello cloud che si sta utilizzando nell'attività del modello cloud non lo consente, modificare il modello cloud in Cloud Assembly e distribuire una nuova versione. Utilizzare quindi la nuova versione del modello cloud nell'attività del modello cloud e aggiungere i binding dollaro dove necessario.

Per applicare binding dollaro ai tipi di proprietà forniti dal modello cloud di Cloud Assembly, è necessario disporre delle autorizzazioni corrette.

- È necessario avere lo stesso ruolo della persona che ha creato la distribuzione del modello cloud in Cloud Assembly.
- La persona che modella la pipeline e la persona che esegue la pipeline possono essere due utenti diversi e possono avere ruoli diversi.
- Se uno sviluppatore ha il ruolo di esecutore di Code Stream e modella la pipeline, deve anche avere lo stesso ruolo di Cloud Assembly della persona che ha distribuito il modello cloud. Ad esempio, è possibile che il ruolo richiesto sia amministratore di Cloud Assembly.

- Solo la persona che modella la pipeline può creare la pipeline e la distribuzione perché dispone delle autorizzazioni necessarie.

Per utilizzare un token API nell'attività del modello cloud:

- La persona che modella la pipeline può fornire un token API a un altro utente che ha il ruolo di esecutore di Code Stream. Quando l'esecutore esegue la pipeline, utilizza il token API e le credenziali create da tale token.
- Quando un utente immette il token API nell'attività del modello cloud, il token crea le credenziali richieste dalla pipeline.
- Per crittografare il valore del token API, fare clic su **Crea variabile**.
- Se non si crea una variabile per il token API e lo si utilizza nell'attività del modello cloud, il valore del token API viene visualizzato come testo normale.

Per applicare binding dollaro alle variabili del modello cloud in un'attività del modello cloud, eseguire i passaggi seguenti.

Iniziare con un modello cloud in cui siano definite le proprietà delle variabili di input, ad esempio `integerVar`, `stringVar`, `flavorVar`, `BooleanVar`, `objectVar` e `arrayVar`. Le proprietà dell'immagine definite sono disponibili nella sezione `resources`. Le proprietà nel codice del modello cloud potrebbero avere il seguente aspetto:

```
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}'
```

È possibile utilizzare le variabili con simbolo del dollaro (\$) per `image` e `flavor`. Ad esempio:

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

Per utilizzare un modello cloud in una pipeline di Code Stream e aggiungervi binding dollaro, eseguire questi passaggi.

- 1 In Code Stream, fare clic su **Pipeline > Tela vuota**.
- 2 Aggiungere un'attività **Modello cloud** alla pipeline.
- 3 Nell'attività Modello cloud, in **Origine modello cloud** selezionare **Modello cloud di Cloud Assembly**, immettere il nome del modello cloud e selezionare la versione del modello cloud.
- 4 Si tenga presente che è possibile immettere un token API, che fornisce le credenziali per la pipeline. Per creare una variabile che crittografa il token API nell'attività del modello cloud, fare clic su **Crea variabile**.
- 5 Nella tabella **Parametro e Valore** che viene visualizzata, si notino i valori dei parametri. Il valore predefinito per `flavor` è `small` e il valore predefinito per `image` è `ubuntu`.
- 6 Si supponga di dover modificare il modello cloud in Cloud Assembly. Ad esempio, è possibile:
 - a Impostare il parametro `flavor` affinché utilizzi una proprietà di tipo `array`. Cloud Assembly accetta valori separati da virgole per `flavor` quando il tipo è **array**.
 - b Fare clic su **Distribuisci**.
 - c Nella pagina Tipo di distribuzione, immettere il nome di una distribuzione e selezionare la versione del modello cloud.
 - d Nella pagina Input di distribuzione, è possibile definire uno o più valori per `flavor`.
 - e Si tenga presente che gli input di distribuzione includono tutte le variabili definite nel codice del modello cloud e vengono visualizzati in base alla definizione del codice del modello cloud. Ad esempio: `Integer Var`, `String Var`, `Flavor Var`, `Boolean Var`, `Object Var` e `Array Var`. `String Var` e `Flavor Var` sono valori stringa, mentre `Boolean Var` è una casella di controllo.
 - f Fare clic su **Distribuisci**.

- 7 In Code Stream, selezionare la nuova versione del modello cloud e immettere i valori nella tabella **Parametro e Valore**. I modelli cloud supportano i seguenti tipi di parametri, che consentono i binding di Code Stream utilizzando variabili del simbolo del dollaro. Esistono lievi differenze tra l'interfaccia utente dell'attività del modello cloud di Code Stream e l'interfaccia utente del modello cloud di Cloud Assembly. In base alla codifica di un modello cloud in Cloud Assembly, l'immissione dei valori nell'attività del modello cloud in Code Stream potrebbe non essere consentita.
- In **flavorVar**, se il modello cloud definisce il tipo come stringa o array, immettere una stringa o un array di valori separati da virgole. Un array di esempio ha il seguente aspetto: **test, test**.
 - In **BooleanVar**, nel menu a discesa selezionare **true** o **false**. In alternativa, per utilizzare il binding della variabile, immettere **\$** e selezionare un binding di variabile

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

Output Parameter

name
description
Stage0

status deploymentcreation deployment deploymentcreation

nell'elenco.

- In **objectVar**, immettere il valore con parentesi graffe e virgolette in questo formato: **{"bkix": "bkix":}**.
 - Il valore **objectVar** verrà passato al modello cloud e potrà essere utilizzato in vari modi in base al modello cloud. Consente un formato stringa per un oggetto JSON ed è possibile aggiungere coppie chiave-valore come valori separati da virgole nella tabella chiave-valore. È possibile immettere testo normale per un oggetto JSON o una coppia chiave-valore come formato di stringa normale per JSON.
 - In **arrayVar**, immettere il valore di input separato da virgole come array in questo formato: **["1", "2"]**.
- 8 Nella pipeline, è possibile associare un parametro di input a un array.
- Fare clic sulla scheda **Input**.
 - Immettere un nome per l'input. Ad esempio, **arrayInput**.
 - Nella tabella **Parametro e Valore**, fare clic su **arrayVar** e immettere **\$ {input.arrayInput}**.

- d Dopo aver salvato e abilitato la pipeline, quando questa viene eseguita, è necessario specificare un valore di input array. Ad esempio, immettere `["1", "2"]` e fare clic su **Esegui**.

Ora si sa come utilizzare i binding di variabili con il simbolo del dollaro (\$) in un modello cloud in un'attività Modello cloud della pipeline di Code Stream.

Come passare un parametro a una pipeline quando è in esecuzione

È possibile aggiungere parametri di input alla pipeline affinché Code Stream li passi alla pipeline. Quando la pipeline è in esecuzione, un utente deve quindi immettere il valore per il parametro di input. Quando si aggiungono parametri di output alla pipeline, le attività della pipeline possono utilizzare il valore di output di un'attività. Code Stream supporta l'utilizzo di parametri in molti modi che supportano le proprie esigenze di pipeline.

Ad esempio, per richiedere all'utente l'URL del server Git quando viene eseguita una pipeline con un'attività REST, è possibile associare l'attività REST a un URL di server Git.

Per creare il binding della variabile, aggiungere una variabile di binding URL all'attività REST. Quando la pipeline è in esecuzione e raggiunge l'attività REST, l'utente deve immettere l'URL del server Git. Per creare il binding, procedere come segue:

- 1 Nella pipeline, fare clic sulla scheda **Input**.
- 2 Per impostare il parametro, per **Invia automaticamente parametri** fare clic su **Git**.

Viene visualizzato l'elenco dei parametri Git, che include **GIT_SERVER_URL**. Se è necessario utilizzare un valore predefinito per l'URL del server Git, modificare questo parametro.

- 3 Fare clic su **Modello**, quindi fare clic sull'attività REST.
- 4 Nella scheda **Attività** nell'area **URL**, immettere \$, quindi selezionare **input** 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} (with a dropdown menu showing various Git variables like GIT_BRANCH_NAME, GIT_CHANGE_SUBJECT, etc.)
 - Agent endpoint:**
 - Headers:**
- Output Parameters:** status, responseHeaders, responseBody, responseJson, responseCode

La voce ha il seguente aspetto: `${input.GIT_SERVER_URL}`

- Per verificare l'integrità dell'associazione delle variabili per l'attività, fare clic su **Convalida attività**.

Code Stream indica che l'attività è stata convalidata correttamente.

- Quando la pipeline esegue l'attività REST, l'utente deve immettere l'URL del server Git. In caso contrario, l'esecuzione dell'attività non viene completata.

Come associare due attività della pipeline mediante la creazione di parametri di input e output

Quando si associano due attività, aggiungere una variabile di binding alla configurazione dell'input dell'attività ricevente. Quindi, quando la pipeline è in esecuzione, un utente sostituisce la variabile di binding con l'input richiesto.

Per associare le attività della pipeline, utilizzare la variabile con il simbolo del dollaro (\$) nei parametri di input e nei parametri di output. Questo esempio illustra come procedere.

Si supponga di voler fare in modo che la pipeline richiami un URL in un'attività REST e che emetta una risposta. Per richiamare l'URL e restituire la risposta, includere i parametri di input e i parametri di output nell'attività REST. È inoltre necessario un utente che possa approvare l'attività e includere un'attività Operazioni utente in modo che un altro utente la possa approvare quando la pipeline è in esecuzione. In questo esempio viene illustrato come utilizzare le espressioni nei parametri di input e output e fare in modo che la pipeline attenda l'approvazione dell'attività.

- 1 Nella pipeline, fare clic sulla scheda **Input**.

The screenshot shows the 'rest-ix-1' pipeline configuration in the 'Input' tab. The 'Auto inject parameters' section has four radio buttons: 'Gerrit', 'Git', 'Docker', and 'None' (which is selected). Below this is an 'ADD' button and a text box 'ADD/REMOVE INJECTED PARAMETERS'. A table below lists the injected parameters:

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

- 2 Lasciare l'opzione **Invia automaticamente parametri** impostata su **Nessuno**.
- 3 Fare clic su **Aggiungi** e immettere il nome, il valore e la descrizione del parametro, quindi fare clic su **OK**. Ad esempio:
 - a Immettere un nome per l'URL.
 - b Immettere il valore: {Stage0.Task3.input.http://www.docs.vmware.com}
 - c Immettere una descrizione.
- 4 Fare clic sulla scheda **Output**, fare clic su **Aggiungi** e immettere il nome e la mappatura del parametro di output.

Add Pipeline Output Parameter

Name *

Reference \$ *

- Immettere un nome univoco per il parametro di output.
- Fare clic nell'area **Riferimento** e immettere \$.
- Immettere la mappatura dell'output dell'attività selezionando le opzioni quando vengono visualizzate. Selezionare **Stage0**, selezionare **Task3**, selezionare **output**, quindi selezionare **responseCode**. Fare clic su **OK**.

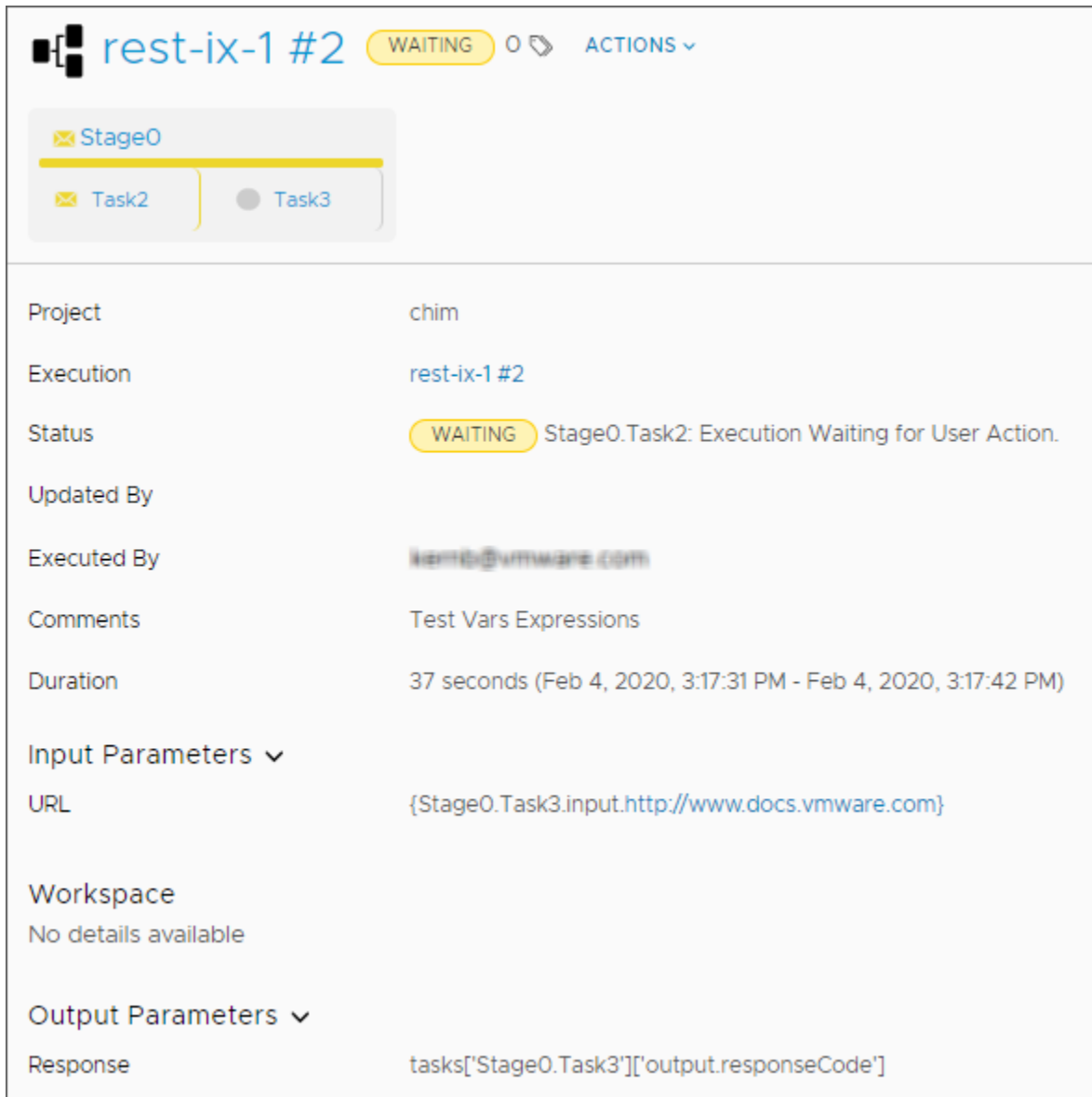
rest-ix-1 Enabled ACTIONS ▾

Workspace Input Model **Output**

Output Parameters ⓘ

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

- Salvare la pipeline.
- Dal menu **Azioni**, fare clic su **Esegui**.
- Fare clic su **Azioni > Visualizza esecuzioni**.
- Fare clic sull'esecuzione della pipeline ed esaminare i parametri di input e i parametri di output definiti in precedenza.



rest-ix-1 #2 WAITING 0 **ACTIONS** ▾

✉ **Stage0**

✉ **Task2** Task3

Project	chim
Execution	rest-ix-1 #2
Status	WAITING Stage0.Task2: Execution Waiting for User Action.
Updated By	
Executed By	kent@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 Per approvare la pipeline, fare clic su **Operazioni utente**. Nella scheda **Elementi attivi** è disponibile l'elenco delle approvazioni. In alternativa, rimanere in Esecuzioni, fare clic sull'attività e quindi fare clic su **Approva**.
- 10 Per abilitare i pulsanti **Approva** e **Rifiuta**, fare clic sulla casella di controllo accanto all'esecuzione.
- 11 Per visualizzare i dettagli, espandere il menu a discesa.
- 12 Per approvare l'attività, fare clic su **APPROVA**, immettere un motivo e fare clic su **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, f0f2@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 Fare clic su **Esecuzioni**. L'esecuzione della pipeline continua.

Executions 3,347 items GUIDED SETUP

+ NEW EXECUTION

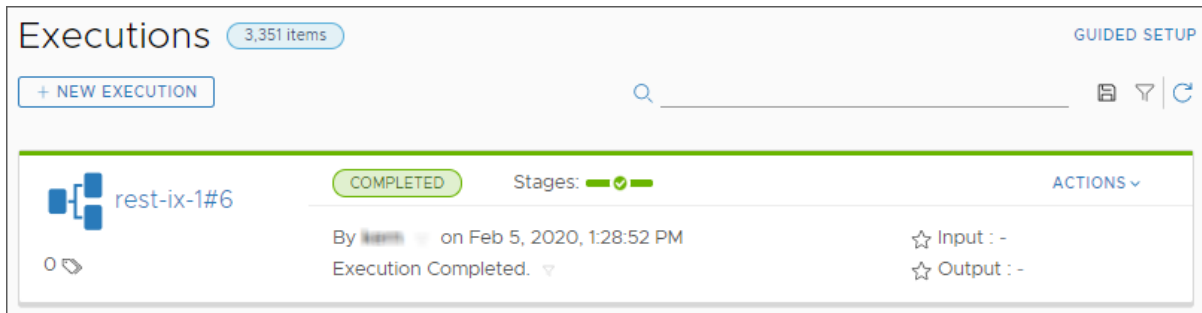
rest-i... #3 RUNNING Stages: 0 ACTIONS

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

RUNNING ☆ Output : -

Comments: Testing

- 14 Se la pipeline non riesce, correggere gli eventuali errori, quindi salvare la pipeline ed eseguirla di nuovo.



Ulteriori informazioni su variabili ed espressioni

Per informazioni dettagliate sull'utilizzo di variabili ed espressioni quando si associano le attività della pipeline, vedere [Variabili ed espressioni che è possibile utilizzare quando si associano le attività della pipeline in Code Stream](#).

Per informazioni su come utilizzare l'output delle attività della pipeline con un binding di variabili di condizioni, vedere [Come utilizzare le associazioni di variabili in un'attività di condizione per eseguire o arrestare una pipeline in Code Stream](#).

Come utilizzare le associazioni di variabili in un'attività di condizione per eseguire o arrestare una pipeline in Code Stream

È possibile fare in modo che l'output di un'attività nella pipeline determini se la pipeline deve continuare a essere eseguita oppure deve essere interrotta in base a una condizione specificata.. Per fare in modo che la pipeline riesca o meno in base all'output dell'attività, è necessario utilizzare l'attività Condizione.

È possibile utilizzare l'attività **Condizione** come punto di decisione nella pipeline. Utilizzando l'attività Condizione con un'espressione della condizione specificata, è possibile valutare tutte le proprietà nella pipeline, nelle fasi e nelle attività.

Il risultato dell'attività Condizione determina se l'attività successiva nella pipeline viene eseguita.

- Una condizione true consente alla pipeline di continuare a essere eseguita.
- Una condizione false interrompe la pipeline.

Per esempi di come utilizzare il valore di output di un'attività come input per l'attività successiva associando le attività con un'attività Condizione, vedere [Come utilizzare i binding delle variabili nelle pipeline di Code Stream](#).

Tabella 3-7. Correlazione tra la pipeline, l'attività Condizione e l'espressione della condizione

Attività Condizione	Su cosa influisce	Funzioni
Attività Condizione	Pipeline	L'attività Condizione determina se la pipeline viene eseguita o interrotta in quel punto, a seconda che l'output dell'attività sia true o false.
Espressione della condizione	Output dell'attività Condizione	<p>Quando viene eseguita la pipeline, l'espressione della condizione che viene inclusa nell'attività Condizione produce uno stato di output true o false. Ad esempio, un'espressione della condizione può richiedere che lo stato dell'output dell'attività Condizione sia <code>Completato</code> o che venga utilizzato il numero di build 74.</p> <p>L'espressione della condizione viene visualizzata nella scheda Attività dell'attività Condizione.</p>

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><</code> , <code><=</code> , <code>></code> , <code>>=</code> , <code>==</code> , <code>!=</code>
Arithmetic operators	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>
Boolean	<code>&&</code> (logical and), <code> </code> (logical or)

L'attività **Condizione** differisce nella funzione e nel comportamento dall'impostazione **Su condizione** in altri tipi di attività.

In altri tipi di attività, **Su condizione** determina se viene eseguita l'attività corrente, anziché le attività successive, in base alla valutazione della relativa espressione di preconditione di true o false. L'espressione della condizione per l'impostazione **Su condizione** produce uno stato di output true o false per l'attività corrente quando la pipeline viene eseguita. L'impostazione **Su condizione** viene visualizzata nella scheda Attività con la propria espressione della condizione. In questo esempio viene utilizzata l'attività Condizione.

Prerequisiti

- Verificare che esista una pipeline e che includa fasi e attività.

Procedura

- 1 Nella pipeline determinare il punto di decisione in cui deve essere visualizzata l'attività Condizione.
- 2 Aggiungere l'attività Condizione prima dell'attività che dipende dal suo esito riuscito o meno.
- 3 Aggiungere un'espressione della condizione all'attività Condizione.

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

- 4 Convalidare l'attività.
- 5 Salvare la pipeline, quindi abilitarla ed eseguirla.

Risultati

Osservare le esecuzioni della pipeline e verificare se la pipeline continua a essere eseguita oppure si interrompe in corrispondenza dell'attività Condizione.

Operazioni successive

Se si esegue il rollback di una distribuzione di pipeline, è anche possibile utilizzare l'attività Condizione. Ad esempio, in una pipeline di rollback l'attività Condizione consente a Code Stream di contrassegnare un errore della pipeline in base all'espressione della condizione e può attivare un singolo flusso di rollback per diversi tipi di errore.

Per eseguire il rollback di una distribuzione, vedere [Come eseguire il rollback di una distribuzione in Code Stream](#).

Variabili ed espressioni che è possibile utilizzare quando si associano le attività della pipeline in Code Stream

Con le variabili e le espressioni, è possibile utilizzare i parametri di input e i parametri di output con le attività della pipeline. I parametri immessi associano l'attività della pipeline a una o più variabili, espressioni o condizioni e determinano il comportamento della pipeline durante l'esecuzione.

Le pipeline possono eseguire soluzioni di distribuzione del software semplici o complesse

Quando si associano le attività della pipeline tra loro, è possibile includere espressioni predefinite e complesse. Di conseguenza, la pipeline può eseguire soluzioni di distribuzione del software semplici o complesse.

Per creare i parametri nella pipeline, fare clic sulla scheda **Input** o **Output** e aggiungere una variabile immettendo il simbolo di dollaro \$ e un'espressione. Ad esempio, questo parametro viene utilizzato come input di attività che richiama un URL: `${Stage0.Task3.input.URL}`.

Il formato delle associazioni di variabili utilizza componenti della sintassi denominati ambiti (scope) e chiavi (key). `SCOPE` definisce il contesto come input o output e `KEY` definisce i dettagli. Nell'esempio di parametro `${Stage0.Task3.input.URL}`, `input` è il componente `SCOPE` e l'URL è il componente `KEY`.

Le proprietà di output di qualsiasi attività possono essere risolte in un numero qualsiasi di livelli nidificati di binding di variabili.

Per ulteriori informazioni sull'utilizzo delle associazioni delle variabili nelle pipeline, vedere [Come utilizzare i binding delle variabili nelle pipeline di Code Stream](#).

Utilizzo delle espressioni di dollaro con gli ambiti e le chiavi per associare le attività della pipeline

È possibile associare le attività della pipeline utilizzando le espressioni nelle variabili del simbolo di dollaro. Le espressioni vengono immesse come `${SCOPE.KEY.<PATH>}`.

Per determinare il comportamento di un'attività di pipeline, in ogni espressione, **SCOPE** è il contesto utilizzato da Code Stream. L'ambito cerca un valore **KEY**, che definisce il dettaglio dell'azione eseguita dall'attività. Quando il valore di **KEY** è un oggetto nidificato, è possibile fornire un valore **PATH** facoltativo.

Questi esempi descrivono **SCOPE** e **KEY**, e illustrano come usarli nella pipeline.

Tabella 3-8. Utilizzo di SCOPE e KEY

SCOPE	Scopo dell'espressione ed esempio	KEY	Come utilizzare SCOPE e KEY nella pipeline
input	Proprietà di input di una pipeline: <code>\${input.input1}</code>	Nome della proprietà di input	Per fare riferimento alla proprietà di input di una pipeline in un'attività, utilizzare il seguente formato: <pre>tasks: mytask: type: REST input: url: \$ {input.url} action: get</pre> <pre>input: url: https:// www.vmware.com</pre>
output	Proprietà di output di una pipeline: <code>\${output.output1}</code>	Nome della proprietà di output	Per fare riferimento a una proprietà di output per l'invio di una notifica, utilizzare il seguente formato: <pre>notifications: email: - endpoint: MyEmailEndpoint subject: "Deployment Successful" event: COMPLETED to: - user@example.org body: Pipeline deployed the service successfully. Refer \$ {output.serviceURL}</pre>

Tabella 3-8. Utilizzo di SCOPE e KEY (continua)

SCOPE	Scopo dell'espressione ed esempio	KEY	Come utilizzare SCOPE e KEY nella pipeline
task input	Input in un'attività: \$ {MY_STAGE.MY_TASK.input. SOMETHING}	Indica l'input di un'attività in una notifica	Quando viene avviato un processo Jenkins, può fare riferimento al nome del processo attivato dall'input dell'attività. In questo caso, inviare una notifica utilizzando il seguente 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>
task output	Output di un'attività: \$ {MY_STAGE.MY_TASK.output .SOMETHING}	Indica l'output di un'attività in un'attività successiva	Per fare riferimento all'output dell'attività della pipeline 1 nell'attività 2, utilizzare il seguente 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>

Tabella 3-8. Utilizzo di SCOPE e KEY (continua)

SCOPE	Scopo dell'espressione ed esempio	KEY	Come utilizzare SCOPE e KEY nella pipeline
var	Variabile: <code>\${var.myVariable}</code>	Fa riferimento alla variabile in un endpoint	Per fare riferimento a una variabile segreta in un endpoint per una password, utilizzare il seguente formato: <pre>--- project: MyProject kind: ENDPOINT name: MyJenkinsServer type: jenkins properties: url: https:// jenkins.example.com username: jenkinsUser password: \$ {var.jenkinsPassword}</pre>
var	Variabile: <code>\${var.myVariable}</code>	Fa riferimento alla variabile in una pipeline	Per fare riferimento alla variabile in un URL della pipeline, utilizzare il seguente formato: <pre>tasks: task1: type: REST input: action: get url: \$ {var.MY_SERVER_URL}</pre>
task status	Stato di un'attività: <pre>\$ {MY_STAGE.MY_TASK.status} \$ {MY_STAGE.MY_TASK.status Message}</pre>		
stage status	Stato di una fase: <pre>\${MY_STAGE.status} \$ {MY_STAGE.statusMessage}</pre>		

Espressioni predefinite

È possibile utilizzare variabili con espressioni nella pipeline. In questo riepilogo sono incluse le espressioni predefinite che è possibile utilizzare.

Espressione	Descrizione
<code>\${comments}</code>	Commenti forniti quando al momento della richiesta di esecuzione della pipeline.
<code>\${duration}</code>	Durata dell'esecuzione della pipeline.
<code>\${endTime}</code>	Ora di fine dell'esecuzione della pipeline in UTC, se conclusa.
<code>\${executedOn}</code>	Come l'ora iniziale, l'ora di inizio dell'esecuzione della pipeline in UTC.
<code>\${executionId}</code>	ID dell'esecuzione della pipeline.
<code>\${executionUrl}</code>	URL che passa all'esecuzione della pipeline nell'interfaccia utente.
<code>\${name}</code>	Nome della pipeline.
<code>\${requestBy}</code>	Nome dell'utente che ha richiesto l'esecuzione.
<code>\${stageName}</code>	Nome della fase corrente, quando utilizzato nell'ambito di una fase.
<code>\${startTime}</code>	Ora di inizio dell'esecuzione della pipeline in UTC.
<code>\${status}</code>	Stato dell'esecuzione.
<code>\${statusMessage}</code>	Messaggio di stato dell'esecuzione della pipeline.
<code>\${taskName}</code>	Nome dell'attività corrente, quando utilizzata in un input o una notifica di attività.

Utilizzo di SCOPE e KEY nelle attività della pipeline

È possibile utilizzare espressioni con una qualsiasi delle attività della pipeline supportate. In questi esempi viene illustrato come definire `SCOPE` e `KEY`, quindi confermare la sintassi. Gli esempi di codice utilizzano `MY_STAGE` e `MY_TASK` come nomi della fase e dell'attività della pipeline.

Per ulteriori informazioni sulle attività disponibili, vedere [Tipi di attività disponibili in Code Stream](#).

Tabella 3-9. Attività di gating

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
Operazione utente			
	Input	<p>summary: riepilogo della richiesta per Operazione utente</p> <p>description: descrizione della richiesta per Operazione utente</p> <p>approvers: elenco degli indirizzi email degli approvatori, in cui ciascuna voce può essere una variabile con una virgola, oppure utilizzare un punto e virgola per gli indirizzi e-mail separati</p> <p>approverGroups: elenco degli indirizzi del gruppo di approvatori per la piattaforma e l'identità</p> <p>sendemail: facoltativamente, invia una notifica tramite e-mail su richiesta o una risposta quando è impostata su true</p> <p>expirationInDays: numero di giorni che rappresenta l'ora di scadenza della richiesta</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>index: stringa esadecimale di sei cifre che rappresenta la richiesta</p> <p>respondedBy: nome dell'account della persona che ha approvato/rifiutato l'operazione utente</p> <p>respondedByEmail: indirizzo e-mail della persona che ha risposto</p> <p>comments: commenti forniti durante la risposta</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>
Condizione			
	Input	<p>condition: condizione da valutare. Quando la condizione viene valutata su true, contrassegna l'attività come completata, mentre le altre risposte non riescono a eseguire l'attività</p>	<pre> \${MY_STAGE.MY_TASK.input.condition} </pre>
	Output	<p>result: risultato dopo la valutazione</p>	<pre> \${MY_STAGE.MY_TASK.output.response} </pre>

Tabella 3-10. Attività della pipeline

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
Pipeline			
	Input	name: nome della pipeline da eseguire inputProperties: proprietà di input da trasmettere all'esecuzione della pipeline nidificata	<pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.inputProperties} # Fa riferimento a tutte le proprietà \$ {MY_STAGE.MY_TASK.input.inputProperties.inpu t1} # Fa riferimento al valore di input1 </pre>
	Output	executionStatus: stato dell'esecuzione della pipeline executionIndex: indice dell'esecuzione della pipeline outputProperties: proprietà di output dell'esecuzione di una pipeline	<pre> \${MY_STAGE.MY_TASK.output.executionStatus} \${MY_STAGE.MY_TASK.output.executionIndex} \${MY_STAGE.MY_TASK.output.outputProperties} # Fa riferimento a tutte le proprietà \$ {MY_STAGE.MY_TASK.output.outputProperties.ou tput1} # Fa riferimento al valore di output1 </pre>

Tabella 3-11. Automazione delle attività di integrazione continua

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
CI			
	Input	steps: un set di stringhe, che rappresentano i comandi da eseguire export: le variabili di ambiente da conservare dopo l'esecuzione delle fasi artifacts: percorsi degli artefatti da conservare in un percorso condiviso process: set di elementi di configurazione per l'elaborazione 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 } # Fa riferimento al percorso della prima configurazione </pre>
	Output	exports: coppia chiave-valore, che rappresenta le variabili di ambiente esportate dal valore export di input artifacts: percorso degli artefatti conservati correttamente processResponse: set di risultati elaborati per l'input process	<pre> \${MY_STAGE.MY_TASK.output.exports} # Fa riferimento a tutte le esportazioni \$ {MY_STAGE.MY_TASK.output.exports.myvar} # Fa riferimento al valore di myvar \${MY_STAGE.MY_TASK.output.artifacts} \$ {MY_STAGE.MY_TASK.output.processRespons e} \$ {MY_STAGE.MY_TASK.output.processRespons e[0].result} # Risultato della prima configurazione del processo </pre>

Tabella 3-11. Automazione delle attività di integrazione continua (continua)

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
Personalizzato			
	Input	name: nome dell'integrazione personalizzata version: una versione dell'integrazione personalizzata, rilasciata oppure obsoleta properties: proprietà da inviare all'integrazione personalizzata	<pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.version} \${MY_STAGE.MY_TASK.input.properties} # Fa riferimento a tutte le proprietà \$ {MY_STAGE.MY_TASK.input.properties.property1} # Fa riferimento al valore di property1 </pre>
	Output	properties: proprietà di output dalla risposta di integrazione personalizzata	<pre> \${MY_STAGE.MY_TASK.output.properties} # Fa riferimento a tutte le proprietà \$ {MY_STAGE.MY_TASK.output.properties.property1} # Fa riferimento al valore di property1 </pre>

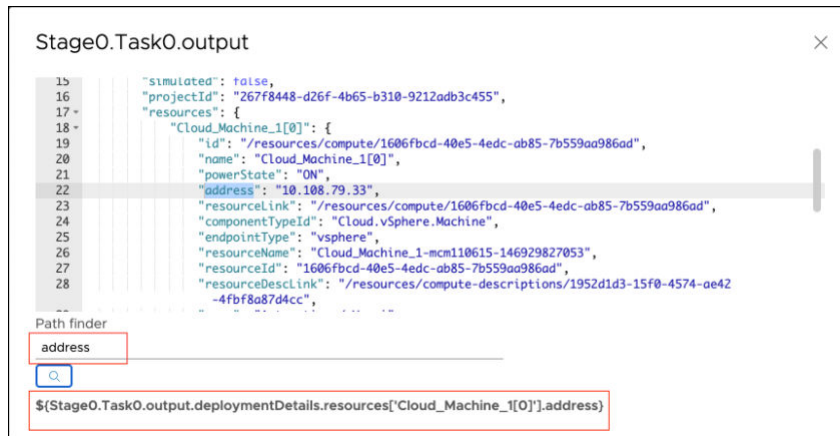
Tabella 3-12. Automazione delle attività di distribuzione continua: modello cloud

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
Modello cloud			
	Input	<p><code>action</code>: uno di createDeployment, updateDeployment, deleteDeployment o rollbackDeployment</p> <p><code>blueprintInputParams</code>: utilizzato per le azioni crea distribuzione e aggiorna distribuzione</p> <p><code>allowDestroy</code>: le macchine possono essere distrutte nel processo di distribuzione dell'aggiornamento.</p> <p>CREATE_DEPLOYMENT</p> <ul style="list-style-type: none"> ■ <code>blueprintName</code>: nome del modello di cloud ■ <code>blueprintVersion</code>: versione del modello cloud <p>OPPURE</p> <ul style="list-style-type: none"> ■ <code>fileUrl</code>: URL del codice YAML del modello cloud remoto, dopo aver selezionato un server GIT. <p>UPDATE_DEPLOYMENT</p> <p>Una di queste combinazioni:</p> <ul style="list-style-type: none"> ■ <code>blueprintName</code>: nome del modello di cloud ■ <code>blueprintVersion</code>: versione del modello cloud <p>OPPURE</p> <ul style="list-style-type: none"> ■ <code>fileUrl</code>: URL del codice YAML del modello cloud remoto, dopo aver selezionato un server GIT. <p>-----</p> <ul style="list-style-type: none"> ■ <code>deploymentId</code>: ID della distribuzione <p>OPPURE</p> <ul style="list-style-type: none"> ■ <code>deploymentName</code>: nome della distribuzione <p>-----</p> <p>DELETE_DEPLOYMENT</p> <ul style="list-style-type: none"> ■ <code>deploymentId</code>: ID della distribuzione 	

Tabella 3-12. Automazione delle attività di distribuzione continua: modello cloud (continua)

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
		<p>OPPURE</p> <ul style="list-style-type: none"> ■ deploymentName: nome della distribuzione <p>ROLLBACK_DEPLOYMENT</p> <p>Una di queste combinazioni:</p> <ul style="list-style-type: none"> ■ deploymentId: ID della distribuzione <p>OPPURE</p> <ul style="list-style-type: none"> ■ deploymentName: nome della distribuzione <p>-----</p> <ul style="list-style-type: none"> ■ blueprintName: nome del modello di cloud ■ rollbackVersion: versione in cui eseguire il rollback 	
	Output		<p>Parametri che possono essere associati ad altre attività o all'output di una pipeline:</p> <ul style="list-style-type: none"> ■ È possibile accedere al nome della distribuzione come \$ <code>{Stage0.Task0.output.deploymentName}</code> ■ È possibile accedere all'ID della distribuzione come \$ <code>{Stage0.Task0.output.deploymentId}</code> ■ Dettagli distribuzione è un oggetto complesso ed è possibile accedere ai dettagli interni utilizzando i risultati JSON. <p>Per accedere a qualsiasi proprietà, utilizzare l'operatore punto per seguire la gerarchia JSON. Ad esempio, per accedere all'indirizzo della risorsa Cloud_Machine_1[0], il binding \$ è:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}</pre> <p>Analogamente, per la caratteristica, il binding \$ è:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}</pre> <p>Nell'interfaccia utente di Code Stream, è possibile ottenere i binding \$ per qualsiasi proprietà.</p> <ol style="list-style-type: none"> 1 Nell'area della proprietà di output dell'attività, fare clic su VISUALIZZA JSON DI OUTPUT. 2 Per trovare il binding \$, immettere una proprietà qualsiasi. 3 Fare clic sull'icona di ricerca, che mostra il binding \$ corrispondente.

Output JSON di esempio:



Oggetto dettagli della distribuzione di esempio:

```

{
  "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"  
}
```


Tabella 3-13. Automazione delle attività di distribuzione continua: Kubernetes

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
Kubernetes			
	Input	<p>action: uno di GET, CREATE, APPLY, DELETE, ROLLBACK</p> <ul style="list-style-type: none"> ■ timeout: timeout generale per qualsiasi azione ■ filterByLabel: etichetta aggiuntiva in base a cui applicare il filtro per l'azione GET utilizzando labelSelector di K8S <p>GET, CREATE, DELETE, APPLY</p> <ul style="list-style-type: none"> ■ yaml: YAML inline da elaborare e inviare a Kubernetes ■ parameters: coppia KEY, VALUE. Sostituire \$\$KEY con VALUE nell'area di input di YAML inline ■ filePath: percorso relativo dall'endpoint Git di SCM, se specificato, da cui recuperare YAML ■ scmConstants: coppia KEY, VALUE - Sostituire \$\$KEY con VALUE in YAML recuperato da SCM. ■ continueOnConflict: quando è impostata su true, se è già presente una risorsa, l'attività continua. <p>ROLLBACK</p> <ul style="list-style-type: none"> ■ resourceType: tipo di risorsa da sottoporre a rollback ■ resourceName: nome della risorsa da sottoporre a rollback ■ namespace: spazio dei nomi in cui deve essere eseguito il rollback ■ revision: revisione in cui eseguire il rollback 	<p><code>\${MY_STAGE.MY_TASK.input.action} #</code> Determina l'azione da eseguire.</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>response: acquisisce l'intera risposta</p> <p>response.<RESOURCE>: la risorsa corrisponde a configMaps, deployments, endpoints, ingresses, jobs, namespaces, pods, replicaSets, replicationControllers, secrets, services, statefulSets, nodes, loadBalancers.</p> <p>response.<RESOURCE>.<KEY>: la chiave corrisponde a uno dei valori apiVersion, kind, metadata e spec</p>	<p><code>\${MY_STAGE.MY_TASK.output.response}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.response.}</code></p>

Tabella 3-14. Integrare applicazioni di sviluppo, test e distribuzione

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
Bamboo			
	Input	plan: nome del piano planKey: chiave del piano variables: variabili da trasmettere al piano parameters: parametri da trasmettere al piano	<pre> \${MY_STAGE.MY_TASK.input.plan} \${MY_STAGE.MY_TASK.input.planKey} \${MY_STAGE.MY_TASK.input.variables} \${MY_STAGE.MY_TASK.input.parameters} # Fa riferimento a tutti i parametri \${MY_STAGE.MY_TASK.input.parameters.param1} # Fa riferimento al valore di param1 </pre>
	Output	resultUrl: URL della creazione risultante buildResultKey: chiave della creazione risultante buildNumber: numero della creazione buildTestSummary: riepilogo dei test eseguiti successfulTestCount: risultato del test trasmesso failedTestCount: risultato del test non riuscito skippedTestCount: risultato del test ignorato artifacts: artefatti dalla creazione	<pre> \${MY_STAGE.MY_TASK.output.resultUrl} \${MY_STAGE.MY_TASK.output.buildResultKey} \${MY_STAGE.MY_TASK.output.buildNumber} \${MY_STAGE.MY_TASK.output.buildTestSummary} # Fa riferimento a tutti i risultati \${MY_STAGE.MY_TASK.output.successfulTestCount} # Fa riferimento al numero di test specifico \${MY_STAGE.MY_TASK.output.buildNumber} </pre>
Jenkins			
	Input	job: nome del processo Jenkins parameters: parametri da trasmettere al processo	<pre> \${MY_STAGE.MY_TASK.input.job} \${MY_STAGE.MY_TASK.input.parameters} # Fa riferimento a tutti i parametri \${MY_STAGE.MY_TASK.input.parameters.param1} # Fa riferimento al valore di un parametro </pre>
	Output	job: nome del processo Jenkins jobId: ID del processo risultante, ad esempio 1234 jobStatus: stato in Jenkins jobResults: raccolta dei risultati di test/code coverage jobUrl: URL del processo eseguito risultante	<pre> \${MY_STAGE.MY_TASK.output.job} \${MY_STAGE.MY_TASK.output.jobId} \${MY_STAGE.MY_TASK.output.jobStatus} \${MY_STAGE.MY_TASK.output.jobResults} # Fa riferimento a tutti i risultati \${MY_STAGE.MY_TASK.output.jobResults.junitResponse} # Fa riferimento ai risultati di JUnit \${MY_STAGE.MY_TASK.output.jobResults.jacocoResponse} # Fa riferimento ai risultati di JaCoCo \${MY_STAGE.MY_TASK.output.jobUrl} </pre>
TFS			

Tabella 3-14. Integrare applicazioni di sviluppo, test e distribuzione (continua)

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
	Input	projectCollection: raccolta di progetti da TFS teamProject: progetto selezionato dalla raccolta disponibile buildDefinitionId: ID di definizione della creazione da eseguire	<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 della creazione risultante buildUrl: URL per visitare il riepilogo della creazione logUrl: URL per visitare i registri dropLocation: posizione di rilascio degli artefatti, se presenti	<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>
vRO			
	Input	workflowId: ID del workflow da eseguire parameters: parametri da trasmettere al workflow	<code>\${MY_STAGE.MY_TASK.input.workflowId}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code>
	Output	workflowExecutionId: ID dell'esecuzione del workflow properties: proprietà di output dall'esecuzione del workflow	<code>\${MY_STAGE.MY_TASK.output.workflowExecutionId}</code> <code>\${MY_STAGE.MY_TASK.output.properties}</code>

Tabella 3-15. Integrare altre applicazioni tramite un'API

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
REST			
	Input	url: URL da chiamare action: metodo HTTP da utilizzare headers: intestazioni HTTP da trasmettere payload: payload della richiesta fingerprint: l'impronta digitale da abbinare per un URL HTTPS allowAllCerts: se impostata su true, può essere qualsiasi certificato con un URL di 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>

Tabella 3-15. Integrare altre applicazioni tramite un'API (continua)

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
	Output	<p>responseCode: codice di risposta HTTP</p> <p>responseHeaders: intestazioni di risposta HTTP</p> <p>responseBody: formato stringa della risposta ricevuta</p> <p>responseJson: risposta attraversabile se content-type è application/json</p>	<pre> \${MY_STAGE.MY_TASK.output.responseCode} \${MY_STAGE.MY_TASK.output.responseHeaders} \$ {MY_STAGE.MY_TASK.output.responseHeaders.header1} # Fa riferimento all'intestazione della risposta 'header1' \${MY_STAGE.MY_TASK.output.responseBody} \${MY_STAGE.MY_TASK.output.responseJson} # Fa riferimento alla risposta come JSON \${MY_STAGE.MY_TASK.output.responseJson.a.b.c} # Fa riferimento all'oggetto nidificato che segue il percorso a. b. c JSON in risposta </pre>
Polling			
	Input	<p>url: URL da chiamare</p> <p>headers: intestazioni HTTP da trasmettere</p> <p>exitCriteria: criteri da soddisfare affinché l'attività abbia esito positivo o negativo. Una coppia chiave-valore di 'success' → Expression, 'failure' → Expression</p> <p>pollCount: numero di iterazioni da eseguire. Un amministratore di Code Stream può impostare il conteggio di polling fino a un massimo di 10000.</p> <p>pollIntervalSeconds: numero di secondi da attendere tra ciascuna iterazione. L'intervallo di polling deve essere maggiore o uguale a 60 secondi.</p> <p>ignoreFailure: quando impostata su true, ignora gli errori di risposta intermedi</p> <p>fingerprint: l'impronta digitale da abbinare per un URL HTTPS</p> <p>allowAllCerts: se impostata su true, può essere qualsiasi certificato con un URL di HTTPS</p>	<pre> \${MY_STAGE.MY_TASK.input.url} \${MY_STAGE.MY_TASK.input.headers} \${MY_STAGE.MY_TASK.input.exitCriteria} \${MY_STAGE.MY_TASK.input.pollCount} \${MY_STAGE.MY_TASK.input.pollIntervalSeconds} \${MY_STAGE.MY_TASK.input.ignoreFailure} \${MY_STAGE.MY_TASK.input.fingerprint} \${MY_STAGE.MY_TASK.input.allowAllCerts} </pre>
	Output	<p>responseCode: codice di risposta HTTP</p> <p>responseBody: formato stringa della risposta ricevuta</p> <p>responseJson: risposta attraversabile se content-type è application/json</p>	<pre> \${MY_STAGE.MY_TASK.output.responseCode} \${MY_STAGE.MY_TASK.output.responseBody} \${MY_STAGE.MY_TASK.output.responseJson} # Refer to response as JSON </pre>

Tabella 3-16. Eseguire script remoti e definiti dall'utente

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
PowerShell Per eseguire un'attività di PowerShell, è necessario: <ul style="list-style-type: none"> ■ Avere una sessione attiva in un host Windows remoto. ■ Se si desidera immettere un comando PowerShell base64, calcolare prima la lunghezza complessiva del comando. Per i dettagli, vedere Tipi di attività disponibili in Code Stream. 			
	Input	host: indirizzo IP o nome host della macchina username: nome utente da utilizzare da utilizzare per la connessione password: password da utilizzare per la connessione useTLS: tentativo di connessione https trustCert: quando è impostata su true, considera attendibili i certificati autofirmati script: script da eseguire workingDirectory: percorso della directory a cui passare prima di eseguire lo script environmentVariables: una coppia chiave-valore della variabile di ambiente da impostare arguments: argomenti di trasmettere allo 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>
	Output	response: contenuto del file \$SCRIPT_RESPONSE_FILE responseFilePath: valore di \$SCRIPT_RESPONSE_FILE exitCode: codice di uscita del processo logFilePath: percorso del file che contiene stdout errorFilePath: percorso del file che contiene 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>
SSH			

Tabella 3-16. Eseguire script remoti e definiti dall'utente (continua)

Attività	Scope	Key	Come utilizzare SCOPE e KEY nell'attività
	Input	host: indirizzo IP o nome host della macchina username: nome utente da utilizzare da utilizzare per la connessione password: password da utilizzare per la connessione (facoltativamente può utilizzare privateKey) privateKey: PrivateKey da utilizzare per la connessione passphrase: passphrase facoltativa per sbloccare privateKey script: script da eseguire workingDirectory: percorso della directory a cui passare prima di eseguire lo script environmentVariables: coppia chiave-valore della variabile di ambiente da impostare	<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: contenuto del file \$SCRIPT_RESPONSE_FILE responseFilePath: valore di \$SCRIPT_RESPONSE_FILE exitCode: codice di uscita del processo logFilePath: percorso del file che contiene stdout errorFilePath: percorso del file che contiene 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>

Come utilizzare un binding di variabili tra le attività

In questo esempio viene illustrato come utilizzare le associazioni di variabili nelle attività della pipeline.

Tabella 3-17. Formati di sintassi di esempio

Esempio	Sintassi
Per utilizzare un valore di output dell'attività per le notifiche della pipeline e le proprietà di output della pipeline	<code>\${<Stage Key>.<Task Key>.output.<Task output key>}</code>
Per fare riferimento al valore di output dell'attività precedente come input per l'attività corrente	<code>\${<Previous/Current Stage key>.<Previous task key not in current Task group>.output.<task output key>}</code>

Ulteriori informazioni

Per ulteriori informazioni sulle variabili di binding nelle attività, vedere:

- [Come utilizzare i binding delle variabili nelle pipeline di Code Stream](#)
- [Come utilizzare le associazioni di variabili in un'attività di condizione per eseguire o arrestare una pipeline in Code Stream](#)
- [Tipi di attività disponibili in Code Stream](#)

Come inviare notifiche sulla pipeline in Code Stream

Le notifiche sono modi per comunicare ai team lo stato delle pipeline in Code Stream.

Per inviare notifiche quando viene eseguita una pipeline, è possibile configurare le notifiche di Code Stream in base allo stato dell'intera pipeline, fase o attività.

- Una notifica tramite e-mail invia un'email su:
 - Completamento, attesa, errore, annullamento o avvio della pipeline.
 - Completamento, errore o avvio della fase.
 - Completamento, attesa, errore o avvio dell'attività.
- Una notifica del ticket crea un ticket e lo assegna a un membro del team su:
 - Errore o completamento della pipeline.
 - Errore della fase.
 - Errore dell'attività.
- Una notifica webhook invia una richiesta a un'altra applicazione su:
 - Errore, completamento, attesa, annullamento o avvio della pipeline.
 - Errore, completamento o avvio della fase.
 - Errore, completamento, attesa o avvio dell'attività.

Ad esempio, è possibile configurare una notifica tramite e-mail su un'attività Operazione utente per ottenere l'approvazione in un punto specifico della pipeline. Quando la pipeline viene eseguita, questa attività invia un'e-mail alla persona che deve approvare l'attività. Se l'attività Operazione utente ha un timeout di scadenza impostato in giorni, ore o minuti, l'utente richiesto deve approvare la pipeline prima che l'attività scada. In caso contrario, la pipeline non riuscirà come previsto.

Per creare un ticket Jira quando un'attività della pipeline non riesce, è possibile configurare una notifica. In alternativa, per inviare una richiesta a un canale Slack sullo stato di una pipeline in base all'evento della pipeline, è possibile configurare una notifica webhook.

È possibile utilizzare variabili in tutti i tipi di notifiche. Ad esempio, è possibile utilizzare `${var}` nell'URL di una notifica webhook.

Prerequisiti

- Verificare che vengano create una o più pipeline. Vedere i casi d'uso in [Capitolo 5 Tutorial sull'utilizzo di Code Stream](#).
- Per inviare notifiche e-mail, verificare che sia possibile accedere a un server e-mail funzionante. Per ulteriori informazioni, consultare l'amministratore.
- Per creare ticket, ad esempio un ticket JIRA, verificare che l'endpoint esista. Vedere [Che cosa sono gli endpoint in Code Stream](#).
- Per inviare una notifica in base a un'integrazione, è possibile creare una notifica webhook. Quindi, verificare che il webhook sia aggiunto e funzionante. È possibile utilizzare le notifiche con applicazioni quali Slack, GitHub o GitLab.

Procedura

- 1 Aprire una pipeline.
- 2 Per creare una notifica sullo stato generale della pipeline o sullo stato di una fase o di un'attività:

Per creare una notifica per:	Cosa fare:
Stato pipeline	Fare clic su un'area vuota della tela della pipeline.
Stato di una fase	Fare clic su un'area vuota in una fase della pipeline.
Stato di un'attività	Fare clic su un'attività in una fase della pipeline.

- 3 Fare clic sulla scheda **Notifiche**.
- 4 Fare clic su **Aggiungi**, selezionare il tipo di notifica e configurare i dettagli della notifica.
- 5 Per creare una notifica Slack quando una pipeline viene eseguita correttamente, creare una notifica webhook.
 - a Selezionare **Webhook**.
 - b Per configurare la notifica Slack, immettere le informazioni.

- c Fare clic su **Salva**.
- d Quando la pipeline viene eseguita, il canale Slack riceve la notifica dello stato della pipeline. Ad esempio, gli utenti potrebbero vedere quanto segue sul canale Slack:

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

- 6 Per creare un ticket JIRA, configurare le informazioni sul ticket.
 - a Selezionare **Ticket**.
 - b Per configurare la notifica Jira, immettere le informazioni.
 - c Fare clic su **Salva**.

Notification

Send notification type

☐ Email
 ☒ Ticket
 ☐ Webhook

When pipeline *

☒ Fails
 ☐ Completes

Jira endpoint *

Jira-Notification ▼

Create Ticket

Jira project *

YourProject

Issue type *

Bug

Assignee *

username@yourcompany.com

Summary \$ *

Pipeline failed

Description \$

Research and correct

CANCEL SAVE

Risultati

Congratulazioni! È stato dimostrato che è possibile creare vari tipi di notifica in diverse aree della pipeline in Code Stream.

Operazioni successive

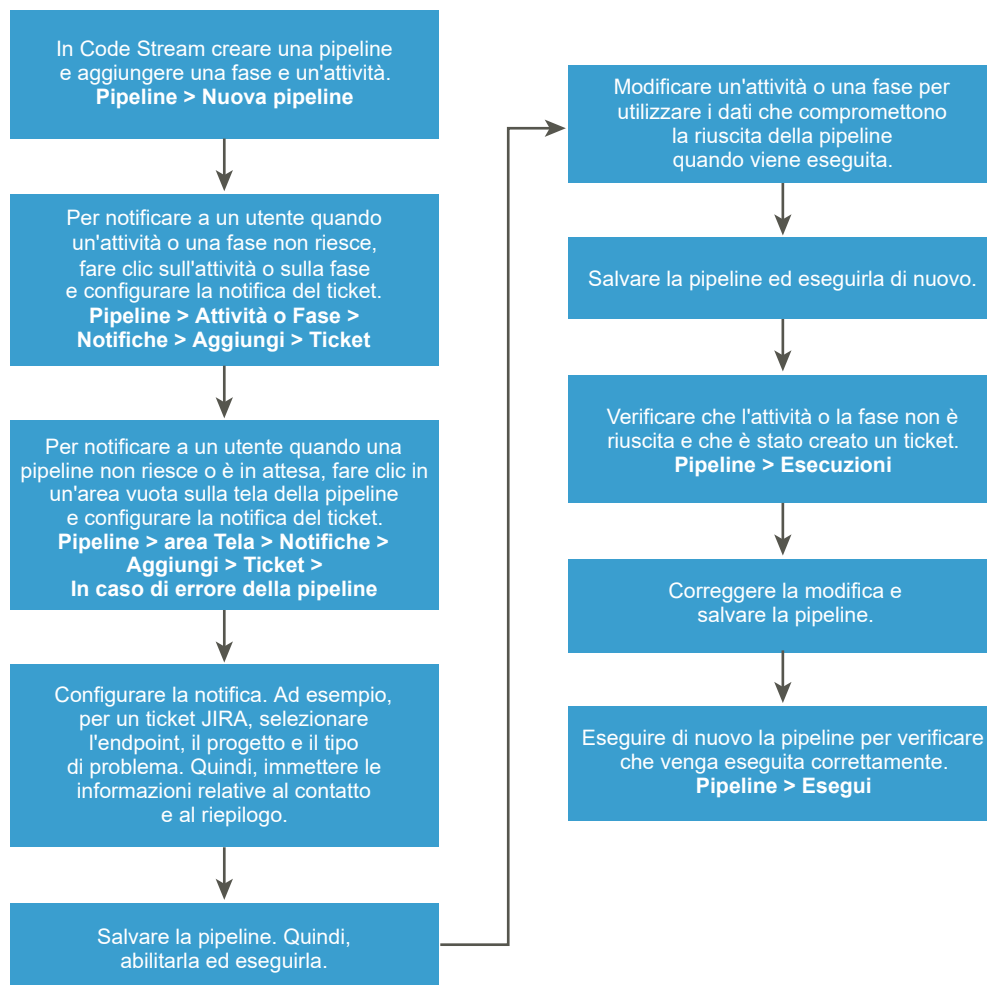
Per un esempio dettagliato di come creare una notifica, vedere [Come creare un ticket Jira in Code Stream quando un'attività della pipeline non riesce](#).

Come creare un ticket Jira in Code Stream quando un'attività della pipeline non riesce

Se una fase o un'attività della pipeline non riesce, è possibile fare in modo che Code Stream crei un ticket Jira. È possibile assegnare il ticket alla persona che deve risolvere il problema. È possibile creare un ticket anche quando la pipeline è in attesa o quando viene eseguita correttamente.

È possibile aggiungere e configurare notifiche relative a un'attività, una fase o una pipeline. Code Stream crea il ticket in base allo stato dell'attività, della fase o della pipeline in cui viene aggiunta la notifica. Ad esempio, se un endpoint non è disponibile, è possibile fare in modo che Code Stream crei un ticket Jira per l'attività che non riesce perché non può connettersi all'endpoint.

È possibile creare notifiche anche per i casi in cui la pipeline viene eseguita correttamente. Ad esempio, è possibile segnalare al team QA la riuscita della pipeline in modo che possa confermare la creazione ed eseguire una pipeline di test diversa. Oppure è possibile informare il team delle prestazioni affinché possa misurare le prestazioni della pipeline e preparare un aggiornamento dello staging o della produzione.



In questo esempio viene creato un ticket Jira quando un'attività della pipeline non riesce.

Prerequisiti

- Verificare di disporre di un account Jira valido e di poter accedere alla propria istanza di Jira.
- Verificare che esista un endpoint Jira e che funzioni.

Procedura

- 1 Nella pipeline fare clic su un'attività.
- 2 Nell'area di configurazione dell'attività, fare clic su **Notifiche**.
- 3 Fare clic su **Aggiungi** e configurare le informazioni sul ticket.
 - a Fare clic su **Ticket**.
 - b Selezionare l'endpoint Jira.
 - c Immettere il progetto Jira e il tipo di problema.
 - d Immettere l'indirizzo e-mail della persona che riceve il ticket.
 - e Immettere un riepilogo e una descrizione del ticket, quindi fare clic su **Salva**.

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 Salvare la pipeline, quindi abilitarla ed eseguirla.

5 Testare il ticket.

- a Modificare le informazioni sull'attività in modo che includano i dati che compromettono la riuscita dell'attività.
- b Salvare la pipeline ed eseguirla di nuovo.
- c Fare clic su **Esecuzioni** e verificare che la pipeline non sia riuscita.
- d Nell'esecuzione, verificare che Code Stream abbia creato il ticket e lo abbia inviato.
- e Modificare le informazioni sull'attività per correggerle, quindi eseguire di nuovo la pipeline e assicurarsi che riesca.

Risultati

Congratulazioni! Code Stream è stato configurato per creare un ticket Jira quando l'attività della pipeline non riesce e per assegnarlo alla persona che deve risolvere il problema.

Operazioni successive

Continuare ad aggiungere notifiche per avvisare il team degli esiti delle pipeline.

Come eseguire il rollback di una distribuzione in Code Stream

È possibile configurare il rollback come pipeline con attività che, in seguito un errore in una pipeline di distribuzione, ripristinano uno stato stabile precedente della distribuzione. Per eseguire il rollback in caso di errore, collegare la pipeline di rollback alle attività o alle fasi.

Le ragioni del rollback possono variare in base al ruolo.

- Per un ingegnere del rilascio potrebbe essere importante che Code Stream verifichi se il rilascio sta avvenendo in modo corretto per sapere se procedere o eseguire il rollback. I possibili errori includono un errore di attività, un rifiuto in UserOps, il superamento della soglia delle metriche.
- Il proprietario dell'ambiente potrebbe voler ridistribuire una versione precedente in modo da ripristinare rapidamente uno stato noto dell'ambiente.
- Potrebbe anche pensare di supportare il rollback di una distribuzione blu-verde in modo da ridurre al minimo i tempi di inattività in caso di errore delle versioni.

Quando si utilizza un modello di pipeline smart per creare una pipeline CD con l'opzione di rollback selezionata, il rollback viene aggiunto automaticamente alle attività nella pipeline. In questo caso d'uso viene utilizzato il modello di pipeline smart per definire il rollback per la distribuzione di un'applicazione in un cluster Kubernetes utilizzando il modello di distribuzione dell'aggiornamento in sequenza. Il modello di pipeline smart crea una pipeline di distribuzione e una o più pipeline di rollback.

- Nella pipeline di distribuzione il rollback è obbligatorio se le attività di aggiornamento o di verifica della distribuzione non riescono.

- Nella pipeline di rollback la distribuzione viene aggiornata con un'immagine precedente.

È inoltre possibile creare manualmente una pipeline di rollback utilizzando un modello vuoto. Prima di creare una pipeline di rollback, è necessario pianificare il flusso di rollback. Per ulteriori informazioni di base sul rollback, vedere [Pianificazione del rollback in Code Stream](#).

Prerequisiti


- Verificare di essere un membro di un progetto in Code Stream. In caso contrario, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).
- Configurare i cluster Kubernetes in cui la pipeline distribuirà l'applicazione. Configurare un cluster di sviluppo e un cluster di produzione.
- Verificare che sia configurato un registro Docker.
- Identificare un progetto che raggruppi tutto il lavoro, inclusi la pipeline, gli endpoint e i dashboard.
- Familiarizzare con il modello smart CD come descritto nella parte CD di [Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart](#), ad esempio:
 - Creare gli endpoint di produzione e di sviluppo Kubernetes che distribuiscono l'immagine dell'applicazione ai cluster Kubernetes.
 - Preparare il file YAML di Kubernetes che crea lo spazio dei nomi, il servizio e la distribuzione. Se è necessario scaricare un'immagine da un repository di proprietà privata, il file YAML deve includere una sezione con il segreto di configurazione di Docker.


Procedura

- 1 Fare clic su **Pipeline > Nuova pipeline > Modello smart > Consegna continua**.
- 2 Immettere le informazioni nel modello di pipeline smart.
 - a Selezionare un progetto.
 - b Immettere il nome di una pipeline ad esempio **RollingUpgrade-Example**.
 - c Selezionare gli ambienti dell'applicazione. Per aggiungere il rollback alla distribuzione, è necessario selezionare **Prod**.
 - d Fare clic su **Seleziona**, scegliere un file YAML Kubernetes e fare clic su **Elabora**.
Il modello di pipeline smart consente di visualizzare gli ambienti di distribuzione e i servizi disponibili.
 - e Selezionare il servizio che la pipeline utilizzerà per la distribuzione.
 - f Selezionare gli endpoint di cluster per l'ambiente di sviluppo e l'ambiente di produzione.
 - g Per l'origine dell'immagine, selezionare **Input runtime pipeline**.
 - h Per il modello di distribuzione, selezionare **Aggiornamento in sequenza**.


- i Fare clic su **Rollback**.
- j Specificare l'**URL controllo dello stato**.

Smart Template: Continuous Delivery

Endpoint prerequisites  Kubernetes Docker Registry


Project * test1 

Pipeline name * RollbackUpgrade-Example

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 * /health-check.json

CREATE CANCEL



- 3 Per creare la pipeline denominata RollbackUpgrade-example, fare clic su **Crea**.

Viene visualizzata la pipeline denominata RollbackUpgrade-Example e l'icona di rollback viene visualizzata nelle attività che possono eseguire il rollback nella fase di sviluppo e nella fase di produzione.

RollbackUpgrade-Example Disabled

Workspace | Input | **Model** | Output

Development

- Create Namespace (Kubernetes)
- Create Secret (Kubernetes)
- Create Service (Kubernetes)

Production

- Update Deployment (Kubernetes)
- Verify Deployment (Kubernetes)

Task: Create Secret

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:

```
1 apiVersion: v1
2 data:
3   .dockercfg: eyJ2ZWZlIiw6G0ue510YV5nby11ZXRM15qZnJvZy5pb15ey31c
4   2VybWZlZS16InhhbmVlLW10dGEy11w1cGFzc3dvcmQ101Jm8tcmV0LW
5   1Uq115ej-c11c11bWp1c15lnhbmVlLW10dGEy11w1cGFzc3dvcmQ101Jm8tcmV0LW
6   1Uq115ej-c11c11bWp1c15lnhbmVlLW10dGEy11w1cGFzc3dvcmQ101Jm8tcmV0LW
7   1Uq115ej-c11c11bWp1c15lnhbmVlLW10dGEy11w1cGFzc3dvcmQ101Jm8tcmV0LW
8   1Uq115ej-c11c11bWp1c15lnhbmVlLW10dGEy11w1cGFzc3dvcmQ101Jm8tcmV0LW
```

Parameters: ADD DELETE

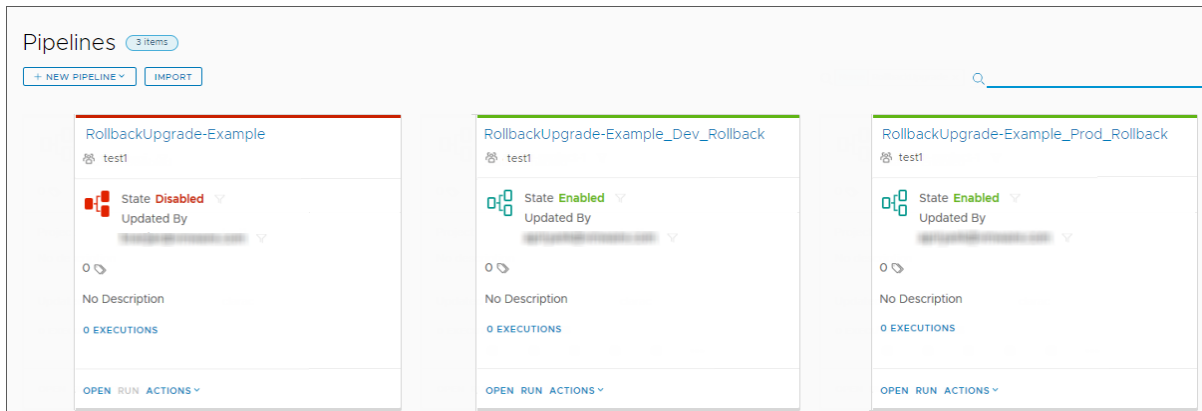
Output Parameters: status response

EDIT RUN CLOSE Last saved 9 minutes ago

4 Chiudere la pipeline

Nella pagina Pipeline viene visualizzata la pipeline creata e una nuova per ogni fase della pipeline.

- RollingUpgrade-Example. Code Stream disattiva la pipeline creata per impostazione predefinita, consentendo di rivederla prima che venga eseguita.
- RollingUpgrade-Example_Dev_Rollback. Errore di attività nella fase di sviluppo, ad esempio **Crea servizio**, **Crea segreto**, **Crea distribuzione** e **Verifica distribuzione** che richiamano questa pipeline di sviluppo di rollback. Per garantire il rollback delle attività di sviluppo, Code Stream abilita la pipeline di sviluppo di rollback per impostazione predefinita.
- RollingUpgrade-Example_Prod_Rollback. Errore delle attività nella fase di produzione, ad esempio **Fase di distribuzione 1**, **Fase di verifica 1**, **Fase di implementazione della distribuzione**, **Fase di implementazione del completamento** e **Fase di implementazione della verifica** che richiamano questa pipeline di produzione di rollback. Per garantire il rollback delle attività di produzione, Code Stream abilita la pipeline di produzione di rollback per impostazione predefinita.



5 Abilitare ed eseguire la pipeline creata.

Quando si avvia l'esecuzione, Code Stream richiede i parametri di input. È possibile fornire l'immagine e il tag per l'endpoint nel repository Docker che si sta utilizzando.

6 Nella pagina Esecuzioni, selezionare **Azioni > Visualizza esecuzione** e osservare l'esecuzione della pipeline.

La pipeline inizia l'**ESECUZIONE** e passa attraverso le attività della fase di sviluppo. Se la pipeline non riesce a eseguire un'attività durante la fase di sviluppo, viene attivata la pipeline denominata RollingUpgrade-Example_Dev_Rollback per eseguire il rollback della distribuzione e lo stato della pipeline cambia in **ROLLING_BACK**.

[< BACK](#)

RollbackUpgrade-Example #1 ROLLING_BACK 0 [ACTIONS](#) [v](#)

[Development](#)

[✓ Create Namespace](#)
[✓ Create Secret](#)
[✓ Create Service](#)
[● Create Deployment](#)
[↺](#)
[● Verify Dep](#)

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 [v](#)
 image demo-image-cs
 tag latest
Workspace
 Details not available
Output Parameters [v](#)
 The Execution did not output any properties

Dopo il rollback, nella pagina Esecuzioni sono elencate due esecuzioni della pipeline RollingUpgrade-Example.

- Viene eseguito il rollback della pipeline creata, dopodiché viene mostrato **ROLLBACK_COMPLETED**.
- La pipeline di sviluppo di rollback che è ha attivata ed eseguito il rollback mostra **COMPLETED**.

Executions 604 items

[+ NEW EXECUTION](#) [Q](#)

RollbackUpgrade-Example_Dev... #1 COMPLETED Stages: ■ ■ ■

1 [Rollback for RollbackUpgrade-Example#1](#)
 By Cloud 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: ■ ■ ■ ■

0 [Rollback for RollbackUpgrade-Example#1](#)
 By Cloud on 01/11/2019 1:24 PM
 Create Deployment ROLLBACK_COMPLETED

Risultati

Congratulazioni! È stata definita una pipeline con rollback ed è stato osservato il rollback della pipeline da parte di Code Stream nel punto di errore.

Pianificazione di creazione, integrazione e consegna native del codice in Code Stream

4

Prima che Code Stream crei, integri e consegna il codice utilizzando la funzionalità nativa che crea una pipeline CI/CD, CI o CD per l'utente, è necessario pianificare la creazione nativa. Quindi, è possibile creare la pipeline utilizzando uno dei modelli di pipeline smart oppure aggiungendo manualmente fasi e attività.

Per pianificare la creazione di integrazione e consegna continue, sono stati inclusi diversi esempi che illustrano come procedere. Questi piani descrivono i prerequisiti e i prerequisiti che possono contribuire a preparare e utilizzare in modo efficace la funzionalità di creazione nativa quando si compilano le pipeline.

Questo capitolo include i seguenti argomenti:

- Configurazione dell'area di lavoro della pipeline
- Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart
- Pianificazione di una creazione nativa di integrazione continua in Code Stream prima di utilizzare il modello di pipeline smart
- Pianificazione di una creazione nativa di consegna continua in Code Stream prima di utilizzare il modello di pipeline smart
- Pianificazione di una creazione nativa di integrazione e consegna continue in Code Stream prima dell'aggiunta manuale di attività
- Pianificazione del rollback in Code Stream

Configurazione dell'area di lavoro della pipeline

Per eseguire le attività di integrazione continua e le attività personalizzate, è necessario configurare un'area di lavoro per la pipeline di Code Stream.

Nell'area di lavoro della pipeline, selezionare **Tipo** come Docker o Kubernetes e specificare il rispettivo endpoint. Le piattaforme Docker e Kubernetes gestiscono l'intero ciclo di vita del contenitore che Code Stream distribuisce per l'esecuzione dell'attività di integrazione continua (CI) o dell'attività personalizzata.

- L'area di lavoro di Docker richiede l'endpoint host Docker, l'URL dell'immagine del generatore, il registro immagini, la directory di lavoro, la cache, le variabili di ambiente, il limite di CPU e il limite di memoria. È inoltre possibile creare un clone del repository Git.
- L'area di lavoro Kubernetes richiede l'endpoint API Kubernetes, l'URL dell'immagine del generatore, il registro immagini, lo spazio dei nomi, la NodePort, la richiesta volume persistente (PVC), la directory di lavoro, le variabili di ambiente, il limite della CPU e il limite di memoria. È inoltre possibile creare un clone del repository Git.

La configurazione dell'area di lavoro della pipeline ha molti parametri comuni e altri parametri specifici per il tipo di area di lavoro, come descritto nella tabella seguente.

Tabella 4-1. Aree di lavoro, dettagli e disponibilità

Selezione	Descrizione	Dettagli e disponibilità
Tipo	Tipo di area di lavoro.	Disponibile con Docker o Kubernetes.
Endpoint host	Endpoint host in cui vengono eseguite le attività di integrazione continua (CI) e personalizzate.	Disponibile nell'area di lavoro di Docker quando si seleziona l'endpoint host di Docker. Disponibile con l'area di lavoro Kubernetes quando si seleziona l'endpoint API Kubernetes.
URL immagine generatore	Nome e posizione dell'immagine del generatore. Viene creato un contenitore utilizzando questa immagine nell'host Docker e nel cluster Kubernetes. Le attività di integrazione continua (CI) e le attività personalizzate vengono eseguite in questo contenitore.	Esempio: fedora:latest L'immagine del generatore deve avere <code>curl</code> o <code>wget</code> .
Registro immagini	Se l'immagine del generatore è disponibile in un registro e il registro richiede le credenziali, è necessario creare un endpoint del Registro immagini, quindi selezionarlo qui in modo che l'immagine possa essere estratta dal registro.	Disponibile con le aree di lavoro di Docker e Kubernetes.
Directory di lavoro	La directory di lavoro è la posizione all'interno del contenitore in cui vengono eseguiti i passaggi dell'attività di integrazione continua (CI), nonché la posizione in cui il codice viene clonato quando un webhook Git attiva l'esecuzione di una pipeline.	Disponibile con Docker o Kubernetes.
Spazio dei nomi	Se non si immette uno spazio dei nomi, Code Stream crea un nome univoco nel cluster Kubernetes specificato.	Specifico dell'area di lavoro Kubernetes.

Tabella 4-1. Aree di lavoro, dettagli e disponibilità (continua)

Selezione	Descrizione	Dettagli e disponibilità
Proxy	<p>Per comunicare con il pod dell'area di lavoro nel cluster Kubernetes, Code Stream distribuisce una singola istanza del proxy nello spazio dei nomi <code>codestream-proxy</code> per ogni cluster Kubernetes. È possibile scegliere il tipo NodePort o LoadBalancer in base alla configurazione del cluster.</p> <p>L'opzione selezionata dipende dalla natura del cluster Kubernetes distribuito.</p> <ul style="list-style-type: none"> ■ In genere, se l'URL del server dell'API Kubernetes specificato nell'endpoint viene esposto tramite uno dei nodi primari, selezionare NodePort. ■ Se l'URL del server dell'API Kubernetes viene esposto da un bilanciamento del carico, come con Amazon EKS (Elastic Kubernetes Service), selezionare LoadBalancer. 	
NodePort	<p>Code Stream utilizza NodePort per comunicare con il contenitore in esecuzione all'interno del cluster Kubernetes.</p> <p>Se non si seleziona una porta, Code Stream utilizza una porta non attiva assegnata da Kubernetes. È necessario assicurarsi che la configurazione delle regole del firewall consenta l'ingresso nell'intervallo di porte temporanee (30000-32767).</p> <p>Se si immette una porta, è necessario assicurarsi che un altro servizio nel cluster non la stia già utilizzando e che la porta sia consentita dalle regole del firewall.</p>	Specifico dell'area di lavoro Kubernetes.
Richiesta volume persistente	<p>Offre un modo per mantenere i file nell'area di lavoro Kubernetes durante le esecuzioni della pipeline. Quando si fornisce un nome di richiesta volume persistente, esso può archiviare i registri, gli artefatti e la cache.</p> <p>Per ulteriori informazioni sulla creazione di una richiesta volume persistente, vedere la documentazione di Kubernetes all'indirizzo https://kubernetes.io/docs/concepts/storage/persistent-volumes/.</p>	Specifico dell'area di lavoro Kubernetes.

Tabella 4-1. Aree di lavoro, dettagli e disponibilità (continua)

Selezione	Descrizione	Dettagli e disponibilità
Variabili di ambiente	Le coppie chiave-valore passate qui saranno disponibili per tutte le attività di integrazione continua (CI) e le attività personalizzate in una pipeline quando viene eseguita.	Disponibile con Docker o Kubernetes. I riferimenti alle variabili possono essere passati qui. Le variabili di ambiente fornite nell'area di lavoro vengono passate a tutte le attività di integrazione continua (CI) e alle attività personalizzate nella pipeline. Se le variabili di ambiente non vengono passate qui, tali variabili devono essere esplicitamente passate a ogni attività di integrazione continua (CI) e attività personalizzata nella pipeline.
Limiti CPU	Limiti per le risorse CPU per il contenitore di integrazione continua (CI) o il contenitore di attività personalizzate.	Il valore predefinito è 1.
Limiti di memoria	Limiti di memoria per il contenitore di integrazione continua (CI) o il contenitore di attività personalizzate.	L'unità è MB.
Clone Git	Quando si seleziona Clone Git e un webhook Git richiama la pipeline, il codice viene clonato nell'area di lavoro (contenitore).	Se non si abilita Clone Git , è necessario configurare un'altra attività di integrazione continua (CI) esplicita nella pipeline per clonare innanzitutto il codice e quindi eseguire altri passaggi come la creazione e il test.
Cache	<p>L'area di lavoro di Code Stream consente di memorizzare nella cache un set di directory o file per velocizzare le esecuzioni della pipeline successive. Esempi di queste directory sono <code>.m2</code> e <code>npm_modules</code>. Se non è necessario memorizzare nella cache i dati tra le esecuzioni della pipeline, non è necessaria una richiesta di volume persistente.</p> <p>Gli artefatti come i file o le directory nel contenitore vengono memorizzati nella cache per poter essere riutilizzati nelle esecuzioni della pipeline. Ad esempio, le cartelle <code>node_modules</code> o <code>.m2</code> possono essere memorizzate nella cache. Cache accetta un elenco di percorsi.</p> <p>Ad esempio:</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>Specifico per il tipo di area di lavoro.</p> <p>Nell'area di lavoro di Docker, la Cache viene ottenuta utilizzando un percorso condiviso nell'host Docker per conservare i dati, gli artefatti e i registri memorizzati nella cache.</p> <p>Nell'area di lavoro di Kubernetes, per consentire l'utilizzo di Cache, è necessario specificare una richiesta di volume persistente. In caso contrario, Cache non è disponibile.</p>

Quando si utilizza un endpoint API Kubernetes nell'area di lavoro della pipeline, Code Stream crea le risorse Kubernetes necessarie, come ConfigMap, Secret e Pod per eseguire l'attività di integrazione continua (CI) o l'attività personalizzata. Code Stream comunica con il contenitore tramite la NodePort.

Per condividere i dati tra le esecuzioni della pipeline, è necessario fornire una richiesta volume persistente e Code Stream monterà la richiesta volume persistente nel contenitore per archiviare i dati e utilizzarla per le esecuzioni della pipeline successive.

Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart

Per creare una pipeline di integrazione e consegna continue (CI/CD) in Code Stream, è possibile utilizzare il modello di pipeline smart CI/CD. Per pianificare la creazione CI/CD nativa, raccogliere le informazioni necessarie per il modello di pipeline smart prima di creare la pipeline in questo piano di esempio.

Per creare una pipeline CI/CD, è necessario pianificare sia le fasi di integrazione continua (CI) che quelle di consegna continua (CD) della pipeline.

Dopo avere immesso le informazioni nel modello di pipeline smart e averlo salvato, il modello crea una pipeline che include fasi e attività. Viene inoltre indicata la destinazione della distribuzione dell'immagine in base ai tipi di ambiente selezionati, ad esempio di sviluppo e di produzione. La pipeline pubblicherà l'immagine del container ed effettuerà le azioni necessarie per eseguirla. Dopo l'esecuzione della pipeline, è possibile monitorare le tendenze nelle esecuzioni.

Quando una pipeline include un'immagine da Docker Hub, è necessario verificare che l'immagine disponga di `cURL` o `wget` incorporato prima di eseguire la pipeline. Quando la pipeline viene eseguita, Code Stream scarica un file binario che utilizza `cURL` o `wget` per eseguire i comandi.

Per informazioni sulla configurazione dell'area di lavoro, vedere [Configurazione dell'area di lavoro della pipeline](#).

Pianificazione della fase di integrazione continua (CI)

Per pianificare la fase CI della pipeline, configurare i requisiti esterni e interni e stabilire le informazioni necessarie per la parte CI del modello di pipeline smart. Di seguito è riportato un riepilogo.

Questo esempio utilizza un'area di lavoro di Docker.

Endpoint e repository necessari:

- Un repository del codice sorgente Git in cui gli sviluppatori archiviano il codice. Code Stream estrae il codice più recente nella pipeline quando gli sviluppatori confermano le modifiche.
- Un endpoint Git per il repository in cui si trova il codice sorgente dello sviluppatore.

- Un endpoint Docker per l'host della build Docker che eseguirà i comandi di creazione all'interno di un contenitore.
- Un endpoint Kubernetes che consente a Code Stream di distribuire l'immagine in un cluster Kubernetes.
- Un'immagine del generatore che crea il contenitore su cui vengono eseguiti i test di integrazione continua.
- Un endpoint del registro immagini dal quale l'host della build Docker può estrarre l'immagine del generatore.

È necessario disporre di accesso a un progetto. Il progetto raggruppa tutto il lavoro, inclusi la pipeline, gli endpoint e i dashboard. Verificare di essere un membro di un progetto in Code Stream. In caso contrario, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).

È necessario un webhook Git che consenta a Code Stream di utilizzare il trigger Git per attivare la pipeline quando gli sviluppatori eseguono il commit delle modifiche al codice. Vedere [Come utilizzare il trigger Git in Code Stream per eseguire una pipeline](#).

I set di strumenti di creazione:

- Il tipo di build, ad esempio Maven.
- Tutti gli strumenti di creazione post-elaborazione utilizzati, tra cui JUnit, JaCoCo, Checkstyle e FindBugs.

Lo strumento di pubblicazione:

- Uno strumento come Docker che distribuirà il contenitore della build.
- Un tag di immagine, ovvero l'ID del commit o il numero di build.

Area di lavoro della build:

- Un host della build Docker, ovvero l'endpoint Docker.
- Un registro immagini. La parte CI della pipeline estrae l'immagine dall'endpoint del registro selezionato. Il contenitore esegue le attività CI e distribuisce l'immagine. Se il registro richiede le credenziali, è necessario creare un endpoint del registro immagini, quindi selezionarlo qui in modo che l'host possa estrarre l'immagine dal registro.
- URL dell'immagine del generatore che crea il contenitore su cui vengono eseguite le attività di integrazione continua.

Pianificazione della fase di consegna continua (CD)

Per pianificare la fase CD della pipeline, configurare i requisiti esterni e interni e stabilire le informazioni da immettere nella parte CD del modello di pipeline smart.

Endpoint necessari:

- Un endpoint Kubernetes che consente a Code Stream di distribuire l'immagine in un cluster Kubernetes.

File e tipi di ambiente:

- Tutti i tipi di ambiente in cui Code Stream distribuirà l'applicazione, ad esempio di sviluppo e di produzione. Il modello di pipeline smart crea le fasi e le attività nella pipeline in base ai tipi di ambiente selezionati.

Tabella 4-2. Fasi della pipeline create dal modello di pipeline smart CICD

Contenuto della pipeline	Funzioni
Fase di creazione-pubblicazione	Crea e testa il codice, crea l'immagine del generatore e pubblica l'immagine nell'host Docker.
Fase di sviluppo	Utilizza un cluster di sviluppo Amazon Web Services (AWS) per creare e distribuire l'immagine. In questa fase è possibile creare uno spazio dei nomi nel cluster e una chiave segreta.
Fase di produzione	Utilizza una versione di produzione di VMware Tanzu Kubernetes Grid Integrated Edition (in precedenza denominato VMware Enterprise PKS) per distribuire l'immagine in un cluster Kubernetes di produzione.

- Un file YAML Kubernetes selezionato nella sezione CD del modello di pipeline smart CICD.

Il file YAML di Kubernetes include tre sezioni obbligatorie per Spazio dei nomi, Servizio e Distribuzione e una sezione facoltativa per Segreto. Se si intende creare una pipeline scaricando un'immagine da un repository di proprietà privata, è necessario includere una sezione con il segreto di configurazione di Docker. Se la pipeline creata utilizza solo immagini pubblicamente disponibili, non è necessario alcun segreto. Il seguente file YAML di esempio include quattro sezioni.

```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream
  namespace: codestream
---
apiVersion: v1
data:
  .dockerconfigjson:
    eyJhdXNldG90I6eyJodHRwczovL2luZ12345678901ci5pby92MS8iOnsidXNldm5hbWUiOiJhdXRvbWF0aW9uYmV0YSIsInBhc3N3b3JkIjo1Vkl3YXJlQDEyMyIsImVtYWlsIjo1YXV0b21hdGlvbmJldGF1c2VyQGdtYWlsLmNvbSIsImF1dGgiOiJZWFFYwYjIxaGRhbmZibUpsZEdFNlZrMTNzWEpsUURFeU13PT0ifX19
  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
```

Nota Il file YAML di Kubernetes viene utilizzato anche nel modello di pipeline smart CD, come nei seguenti esempi di casi d'uso:

- [Come distribuire un'applicazione in Code Stream nella distribuzione blu-verde](#)
- [Come eseguire il rollback di una distribuzione in Code Stream](#)
- [Come utilizzare il trigger Docker in Code Stream per eseguire una pipeline di consegna continua](#)

Per applicare il file nel modello smart, fare clic su **Seleziona** e selezionare il file YAML di Kubernetes. A questo punto, fare clic su **Elabora**. Il modello di pipeline smart consente di visualizzare gli ambienti di distribuzione e i servizi disponibili. Selezionare un servizio, l'endpoint del cluster e la strategia di distribuzione. Ad esempio, per utilizzare il modello di distribuzione Canary, selezionare **Canary** e immettere una percentuale per la fase di distribuzione.

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 ^{*} %

Rollback ☐

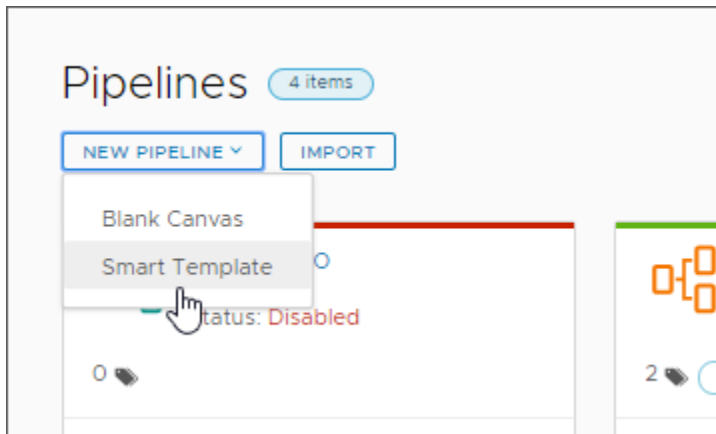
Health check URL ^{*}

Per un esempio di utilizzo del modello di pipeline smart per creare una pipeline per una distribuzione blu-verde, vedere [Come distribuire un'applicazione in Code Stream nella distribuzione blu-verde](#).

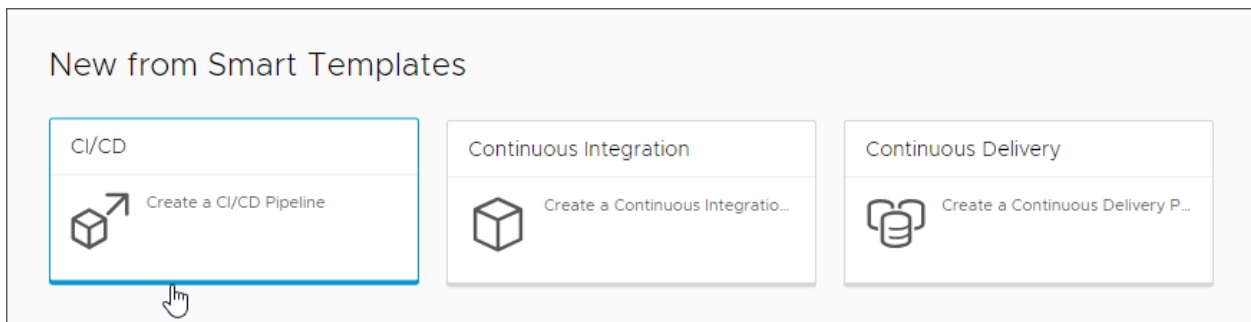
Come creare la pipeline CICD utilizzando il modello di pipeline smart

Dopo aver raccolto tutte le informazioni e aver configurato il necessario, ecco come verrà creata una pipeline dal modello di pipeline smart CICD.

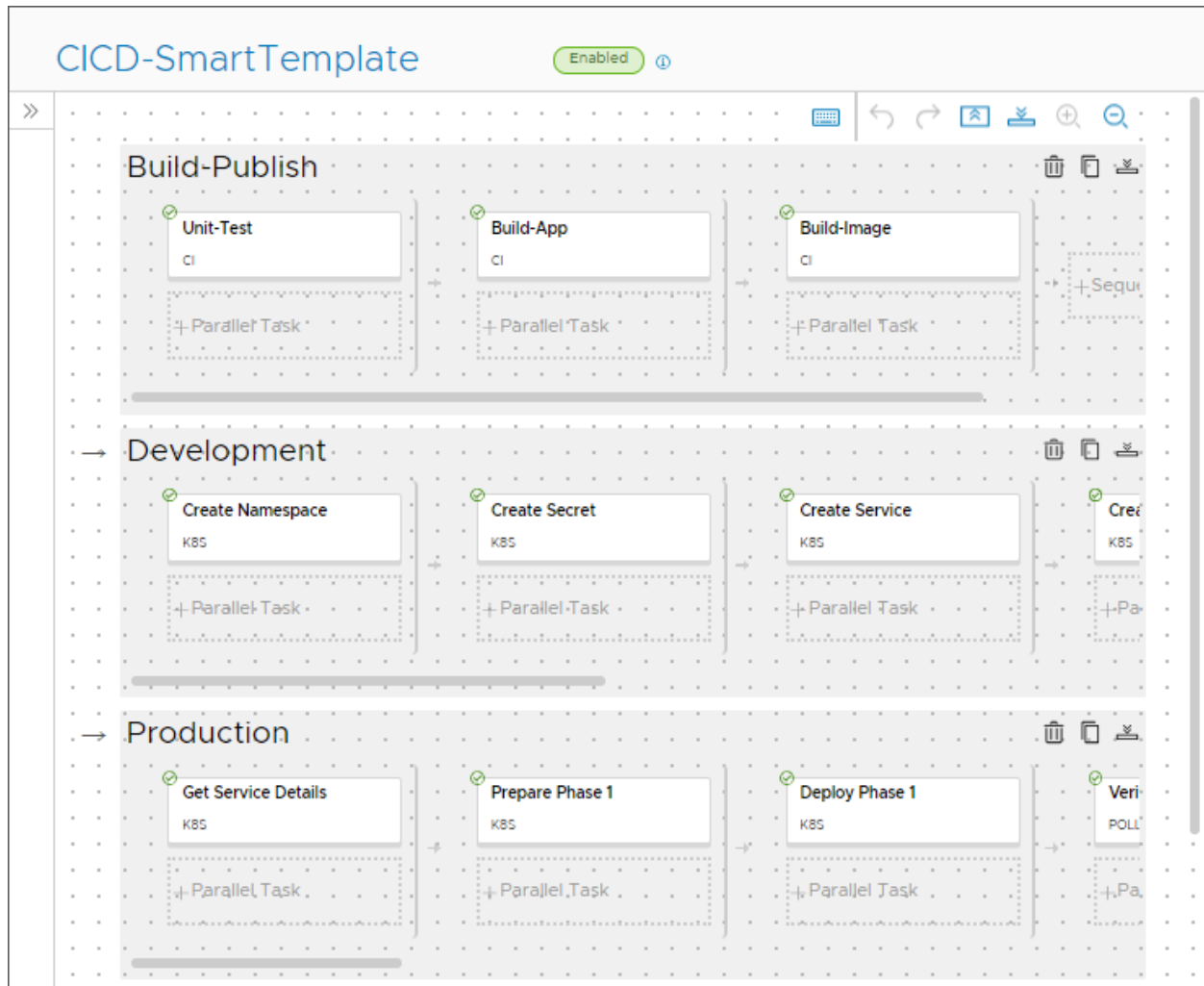
Nelle pipeline selezionare **Nuova pipeline > Modelli smart**.



Selezionare il modello di pipeline smart CICD.



Compilare il modello e salvare la pipeline con le fasi che vengono create. Se necessario, modificare la pipeline prima di salvarla.



Abilitare la pipeline ed eseguirla. Dopo l'esecuzione, eseguire le operazioni seguenti:

- Verificare che la pipeline sia riuscita. Fare clic su **Esecuzioni** e cercare la pipeline. Se non è riuscita, correggere eventuali errori ed eseguirla di nuovo.
- Verificare che il webhook Git funzioni correttamente. Nella scheda **Attività** di Git sono visualizzati gli eventi. Fare clic su **Trigger > Git > Attività**.
- Osservare il dashboard della pipeline ed esaminare le tendenze. Fare clic su **Dashboard** e cercare il dashboard della pipeline. È possibile anche creare un dashboard personalizzato per creare report su indicatori KPI aggiuntivi.

Per un esempio dettagliato, vedere [Come integrare il codice in modo continuativo dal repository GitHub o GitLab nella pipeline in Code Stream](#).

Pianificazione di una creazione nativa di integrazione continua in Code Stream prima di utilizzare il modello di pipeline smart

Per creare una pipeline di integrazione continua (CI) in VMware Code Stream, è possibile utilizzare il modello di pipeline smart di integrazione continua. Per pianificare la creazione nativa di integrazione continua, raccogliere le informazioni necessarie per il modello di pipeline smart prima di creare la pipeline in questo piano di esempio.

Quando si compila il modello di pipeline smart, viene creata una pipeline di integrazione continua nel repository e vengono effettuate le azioni necessarie per eseguirla. Dopo l'esecuzione della pipeline, è possibile monitorare le tendenze nelle esecuzioni.

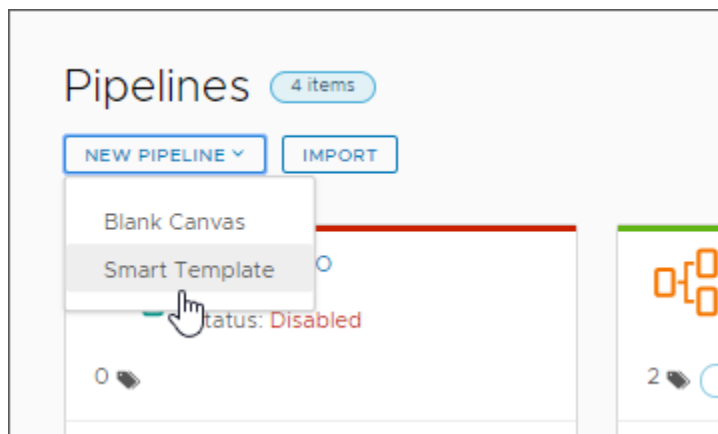
Per pianificare la creazione prima di utilizzare il modello di pipeline smart di integrazione continua:

- Identificare un progetto che raggruppi tutto il lavoro, inclusi la pipeline, gli endpoint e i dashboard.
- Raccogliere le informazioni per la creazione come descritto nella parte consegna continua di [Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart](#).

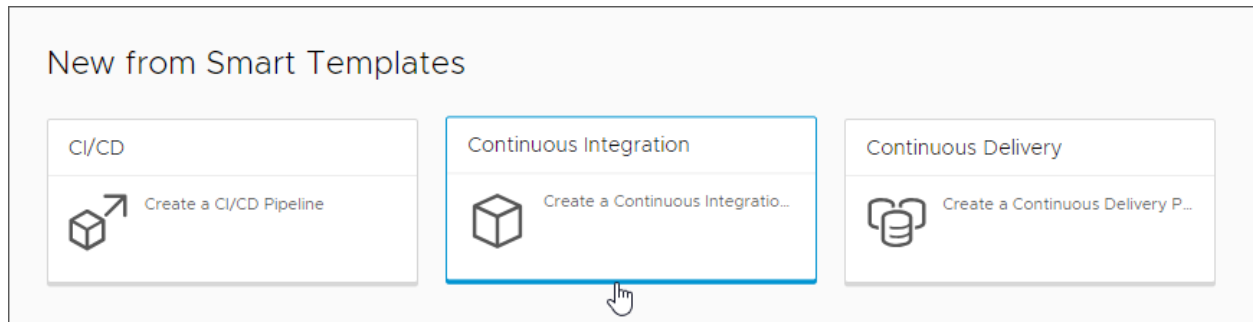
Ad esempio, aggiungere un endpoint Kubernetes in cui Code Stream distribuirà il contenitore.

Quindi, creare una pipeline utilizzando il modello di pipeline smart di integrazione continua.

In Pipeline, selezionare **Modelli smart**.



È possibile selezionare il modello di pipeline smart di integrazione continua.



Per salvare la pipeline con le fasi create, compilare il modello e assegnare un nome alla pipeline. Per salvare la pipeline con le fasi create, fare clic su **Crea**.

L'area di lavoro della pipeline di Code Stream supporta Docker e Kubernetes per le attività di integrazione continua e le attività personalizzate.

Per informazioni sulla configurazione dell'area di lavoro, vedere [Configurazione dell'area di lavoro della pipeline](#).

Per apportare modifiche finali, è possibile modificare la pipeline. Quindi, è possibile abilitare la pipeline ed eseguirla. Dopo l'esecuzione della pipeline:

- Verificare che la pipeline sia riuscita. Fare clic su **Esecuzioni** e cercare la pipeline. Se non è riuscita, correggere eventuali errori ed eseguirla di nuovo.
- Verificare che il webhook Git funzioni correttamente. Nella scheda **Attività** di Git sono visualizzati gli eventi. Fare clic su **Trigger > Git > Attività**.
- Osservare il dashboard della pipeline ed esaminare le tendenze. Fare clic su **Dashboard** e cercare il dashboard della pipeline. Per creare report su più indicatori di prestazioni chiave, è possibile creare un dashboard personalizzato.

Per un esempio dettagliato, vedere [Come integrare il codice in modo continuativo dal repository GitHub o GitLab nella pipeline in Code Stream](#).

Pianificazione di una creazione nativa di consegna continua in Code Stream prima di utilizzare il modello di pipeline smart

Per creare una pipeline di consegna continua (CD) in Code Stream, è possibile utilizzare il modello di pipeline smart di consegna continua. Per pianificare la creazione nativa di consegna continua, raccogliere le informazioni necessarie per il modello di pipeline smart prima di creare la pipeline in questo piano di esempio.

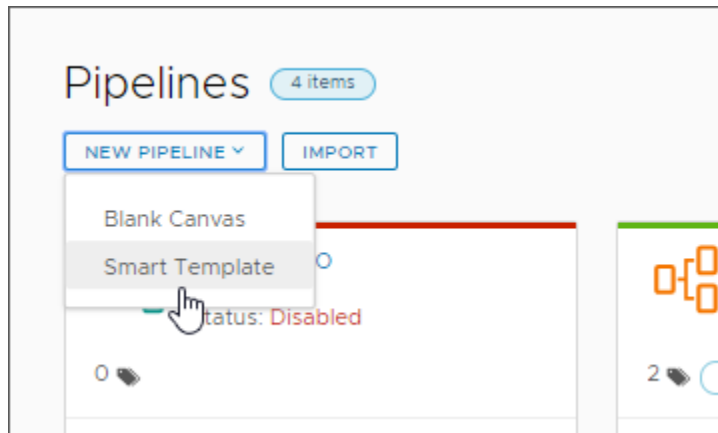
Quando si compila il modello di pipeline smart, viene creata una pipeline di consegna continua nel repository e vengono effettuate le azioni necessarie per eseguirla. Dopo l'esecuzione della pipeline, è possibile monitorare le tendenze nelle esecuzioni.

Per pianificare la creazione prima di utilizzare il modello di pipeline smart di consegna continua:

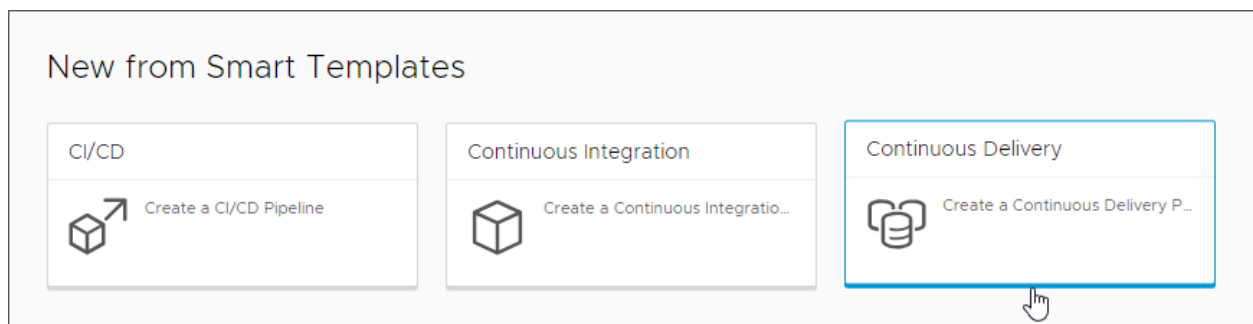
- Identificare un progetto che raggruppi tutto il lavoro, inclusi la pipeline, gli endpoint e i dashboard.
- Raccogliere le informazioni per la creazione come descritto nella parte consegna continua di [Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart](#). Ad esempio:
 - Aggiungere un endpoint Kubernetes in cui Code Stream distribuirà il contenitore.
 - Preparare il file YAML di Kubernetes che crea lo spazio dei nomi, il servizio e la distribuzione. Per scaricare un'immagine da un repository di proprietà privata, il file YAML deve includere una sezione con il segreto di configurazione di Docker.

Quindi, creare una pipeline utilizzando il modello di pipeline smart di consegna continua.

In Pipeline, selezionare **Modelli smart**.



È possibile selezionare il modello di pipeline smart di consegna continua.



Compilare il modello e assegnare un nome alla pipeline. Per salvare la pipeline con le fasi create, fare clic su **Crea**.

L'area di lavoro della pipeline di Code Stream supporta Docker e Kubernetes per le attività di integrazione continua e le attività personalizzate.

Per informazioni sulla configurazione dell'area di lavoro, vedere [Configurazione dell'area di lavoro della pipeline](#).

Per apportare modifiche finali, è possibile modificare la pipeline. Quindi, è possibile abilitare la pipeline ed eseguirla. Dopo l'esecuzione della pipeline:

- Verificare che la pipeline sia riuscita. Fare clic su **Esecuzioni** e cercare la pipeline. Se non è riuscita, correggere eventuali errori ed eseguirla di nuovo.
- Verificare che il webhook Git funzioni correttamente. Nella scheda **Attività** di Git sono visualizzati gli eventi. Fare clic su **Trigger > Git > Attività**.
- Osservare il dashboard della pipeline ed esaminare le tendenze. Fare clic su **Dashboard** e cercare il dashboard della pipeline. Per creare report su più indicatori di prestazioni chiave, è possibile creare un dashboard personalizzato.

Per un esempio dettagliato, vedere [Come integrare il codice in modo continuativo dal repository GitHub o GitLab nella pipeline in Code Stream](#).

Pianificazione di una creazione nativa di integrazione e consegna continue in Code Stream prima dell'aggiunta manuale di attività

Per creare una pipeline di integrazione e consegna continue (CI/CD) in Code Stream, è possibile aggiungere manualmente fasi e attività. Per pianificare la creazione nativa di CI/CD, raccogliere le informazioni necessarie, quindi creare una pipeline e aggiungere manualmente fasi e attività.

È necessario pianificare sia le fasi di integrazione continua (CI) che quelle di consegna continua (CD) della pipeline. Dopo aver creato ed eseguito la pipeline, è possibile monitorare le tendenze nelle relative esecuzioni.

Quando una pipeline include un'immagine da Docker Hub, è necessario verificare che l'immagine disponga di `cURL` o `wget` incorporato prima di eseguire la pipeline. Quando la pipeline viene eseguita, Code Stream scarica un file binario che utilizza `cURL` o `wget` per eseguire i comandi.

L'area di lavoro della pipeline di Code Stream supporta Docker e Kubernetes per le attività di integrazione continua e le attività personalizzate.

Per informazioni sulla configurazione dell'area di lavoro, vedere [Configurazione dell'area di lavoro della pipeline](#).

Pianificazione dei requisiti esterni e interni

Per pianificare le fasi CI e CD della pipeline, è necessario verificare che i seguenti requisiti siano soddisfatti prima di creare la pipeline.

Questo esempio utilizza un'area di lavoro di Docker.

Per creare una pipeline da questo piano di esempio, si utilizza un host Docker, un repository Git, Maven e diversi strumenti di creazione post elaborazione.

Endpoint e repository necessari:

- Un repository del codice sorgente Git in cui gli sviluppatori archiviano il codice. Code Stream estrae il codice più recente nella pipeline quando gli sviluppatori confermano le modifiche.
- Un endpoint Docker per l'host della build Docker che eseguirà i comandi di creazione all'interno di un contenitore.
- Un'immagine del generatore che crea il contenitore su cui vengono eseguiti i test di integrazione continua.
- Un endpoint del registro immagini dal quale l'host della build Docker può estrarre l'immagine del generatore.

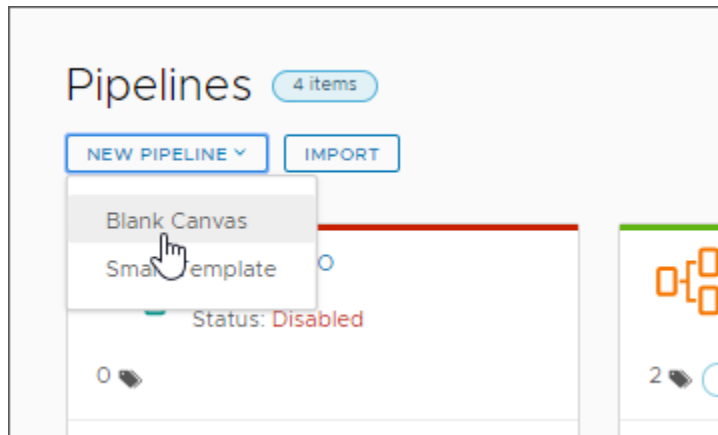
È necessario disporre di accesso a un progetto. Il progetto raggruppa tutto il lavoro, inclusi la pipeline, gli endpoint e i dashboard. Verificare di essere un membro di un progetto in Code Stream. In caso contrario, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).

È necessario un webhook Git che consenta a Code Stream di utilizzare il trigger Git per attivare la pipeline quando gli sviluppatori eseguono il commit delle modifiche al codice. Vedere [Come utilizzare il trigger Git in Code Stream per eseguire una pipeline](#).

Come creare la pipeline CI/CD e configurare l'area di lavoro

È necessario creare la pipeline, quindi configurare le aree di lavoro, i parametri di input e le attività della pipeline.

Per creare la pipeline, fare clic su **Pipeline > Nuova pipeline > Tela vuota**.



Nella scheda Area di lavoro, immettere le informazioni di integrazione continua:

- Includere l'host della build Docker.
- Immettere l'URL dell'immagine del generatore.

- Selezionare l'endpoint del registro immagini da cui la pipeline deve estrarre l'immagine. Il contenitore esegue le attività CI e distribuisce l'immagine. Se il registro richiede le credenziali, è innanzitutto necessario creare un endpoint del registro immagini, quindi selezionarlo qui in modo che l'host possa estrarre l'immagine dal registro.
- Aggiungere gli artefatti che devono essere memorizzati nella cache. Affinché una creazione riesca, gli artefatti come le directory devono essere scaricati come dipendenze. La cache è la posizione in cui si trovano questi artefatti. Ad esempio, gli artefatti dipendenti possono includere la directory `.m2` per Maven e la directory `node_modules` per Node.js. Queste directory vengono memorizzate nella cache nelle esecuzioni della pipeline per risparmiare tempo durante le creazioni.

Nella scheda Input configurare i parametri di input della pipeline.

- Se la pipeline utilizzerà parametri di input da un evento trigger Git, Gerrit o Docker, selezionare il tipo di trigger per i parametri di inserimento automatico. Gli eventi possono includere Modifica oggetto per Gerrit o Git, o il nome del proprietario dell'evento per Docker. Se la pipeline non utilizzerà alcun parametro di input passato dall'evento, lasciare i parametri di invio automatico impostati su **Nessuno**.
- Per applicare un valore e una descrizione a un parametro di input della pipeline, fare clic sui tre punti verticali, quindi su **Modifica**. Il valore immesso viene utilizzato come input per le attività, le fasi o le notifiche.
- Per aggiungere un parametro di input della pipeline, fare clic su **Aggiungi**. Ad esempio, è possibile aggiungere `approvers` per visualizzare un valore predefinito per ogni esecuzione, ma che può essere sovrascritto con un approvatore diverso in fase di runtime.
- Per aggiungere o rimuovere un parametro immesso, fare clic su **Aggiungi/Rimuovi parametro immesso**. Ad esempio, rimuovere un parametro inutilizzato per ridurre l'ingombro nella pagina dei risultati e visualizzare solo i parametri di input utilizzati.

Input Workspace Model Output

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

Configurare la pipeline per testare il codice:

- Aggiungere e configurare un'attività CI.
- Includere i passaggi per eseguire `mvn test` nel codice.
- Per identificare eventuali problemi dopo l'esecuzione dell'attività, eseguire gli strumenti di creazione post-elaborazione, ad esempio JUnit e JaCoCo, FindBugs e 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 to 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

Configurare la pipeline per creare il codice:

- Aggiungere e configurare un'attività CI.

- Includere i passaggi per l'esecuzione di `mvn clean install` nel codice.
- Includere la posizione e il nome del file JAR in modo da preservare l'artefatto.

Task Build-App Notifications Rollback VALIDATE TASK

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

Type *

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
 +

JaCoCo
 +

FindBugs
 +

Checkstyle
 +

Configurare la pipeline per pubblicare l'immagine nell'host Docker:

- Aggiungere e configurare un'attività CI.
- Aggiungere i passaggi per il commit, l'esportazione, la creazione e l'inserimento dell'immagine.
- Aggiungere la chiave di esportazione di `IMAGE` per l'attività successiva da utilizzare.

The screenshot shows the 'Build-Image' task configuration in vRealize Automation. The interface includes tabs for 'Task', 'Notifications', and 'Rollback'. A 'VALIDATE TASK' button is in the top right. The 'Task name' is 'Build-Image' with a note about character restrictions. The 'Type' is 'CI'. There is a 'Precondition' field with a 'SYNTAX GUIDE' button. A 'Continue on failure' checkbox is present. The 'Continuous Integration' section has a note about Docker host setup. The 'Steps' section contains a code block with the following commands:

```
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 steps, there are sections for 'Preserve artifacts', 'Export' (with an 'IMAGE' button), and test configurations for 'JUnit', 'JaCoCo', 'FindBugs', and 'Checkstyle', each with 'Label' and 'Path' fields.

Dopo aver configurato l'area di lavoro, i parametri di input, le attività di test e le attività di creazione, salvare la pipeline.

Come abilitare ed eseguire la pipeline

Dopo aver configurato la pipeline con fasi e attività, è possibile salvarla e abilitarla.

Quindi, attendere che la pipeline venga eseguita e terminata e verificare che sia stata eseguita correttamente. Se non è riuscita, correggere eventuali errori ed eseguirla di nuovo.

Se la pipeline è stata eseguita correttamente, verificare quanto segue:

- Esaminare l'esecuzione della pipeline e visualizzare i risultati dei passaggi delle attività.
- Nell'area di lavoro dell'esecuzione della pipeline, individuare i dettagli relativi al contenitore e al repository Git duplicato.
- Nell'area di lavoro, esaminare i risultati degli strumenti di post elaborazione e verificare la presenza di errori, code coverage, bug e problemi di stile.
- Verificare che l'artefatto sia conservato. Verificare inoltre che l'immagine sia stata esportata con il nome e il valore dell'immagine.
- Passare al repository Docker e verificare che la pipeline abbia pubblicato il contenitore.

Per un esempio dettagliato in cui viene illustrato come Code Stream integra il codice in modo continuativo, vedere [Come integrare il codice in modo continuativo dal repository GitHub o GitLab nella pipeline in Code Stream](#).

Pianificazione del rollback in Code Stream

Se l'esecuzione della pipeline non riesce, è possibile utilizzare il rollback per riportare l'ambiente a uno stato precedentemente stabile. Per utilizzare il rollback, pianificare un flusso di rollback e capire come implementarlo.

Un flusso di rollback prescrive i passaggi necessari per ripristinare il sistema in seguito a un errore nella distribuzione. Il flusso assume il formato di una pipeline di rollback, che include una o più attività sequenziali che variano in base al tipo di distribuzione eseguita e non riuscita. Ad esempio, la distribuzione e il rollback di un'applicazione tradizionale sono diversi dalla distribuzione e dal rollback di un'applicazione contenitore.

Per ripristinare uno stato di distribuzione valido, una pipeline di rollback in genere include attività per:

- Pulire stati o ambienti.
- Eseguire uno script specificato dall'utente per annullare le modifiche.
- Distribuire una revisione precedente di una distribuzione.

Per aggiungere il rollback a una pipeline di distribuzione esistente, collegare la pipeline di rollback alle attività o alle fasi della pipeline di distribuzione di cui si desidera eseguire il rollback prima di eseguire la pipeline di distribuzione.

Come si configura il rollback

Per configurare il rollback nella distribuzione, è necessario:

- Creare una pipeline di distribuzione.
- Identificare i potenziali punti di errore nella pipeline di distribuzione che attiverà il rollback in modo da poter collegare la pipeline di rollback. Ad esempio, è possibile collegare la pipeline di rollback a una condizione o a un tipo di attività di polling nella pipeline di distribuzione, che controlla se un'attività precedente è stata completata correttamente. Per informazioni sulle attività delle condizioni, vedere [Come utilizzare le associazioni di variabili in un'attività di condizione per eseguire o arrestare una pipeline in Code Stream](#).
- Determinare l'ambito di errore che attiverà la pipeline di rollback, ad esempio un errore di attività o di fase. È possibile anche collegare il rollback a una fase.
- Stabilire l'attività o le attività di rollback da eseguire in caso di errore. Verrà creata la pipeline di rollback con tali attività.

La pipeline di rollback può essere creata manualmente oppure da Code Stream.

- Utilizzando una tela vuota, è possibile creare manualmente una pipeline di rollback che segue un flusso in parallelo a una pipeline di distribuzione esistente. Quindi, la pipeline di rollback viene collegata a una o più attività nella pipeline di distribuzione che attiva il rollback in caso di errore.

- Utilizzando un modello di pipeline smart è possibile configurare una pipeline di distribuzione con l'azione di rollback. Quindi, Code Stream crea automaticamente una o più pipeline di rollback predefinite con attività predefinite che eseguono il rollback della distribuzione in caso di errore.

Per un esempio dettagliato su come configurare una pipeline CD con il rollback utilizzando un modello di pipeline smart, vedere [Come eseguire il rollback di una distribuzione in Code Stream](#).

Che cosa accade se la pipeline di distribuzione presenta più attività o fasi con rollback

Se sono presenti più attività o attività e fasi a cui viene aggiunto il rollback, tenere presente che la sequenza di rollback varia.

Tabella 4-3. Determinazione della sequenza di rollback

Se si aggiunge il rollback a...	Quando viene eseguito il rollback...
Attività parallele	Se una delle attività parallele non riesce, il relativo rollback si verifica dopo che tutte le attività parallele sono state completate o non sono riuscite. Il rollback non si verifica immediatamente dopo la mancata riuscita dell'attività.
Sia l'attività all'interno di una fase che la fase	Se un'attività non riesce, viene eseguito il relativo rollback. Se l'attività si trova in un gruppo di attività parallele, il rollback dell'attività viene eseguito dopo che tutte le attività parallele sono state completate o non sono riuscite. Dopo che il rollback dell'attività è stato completato o non è riuscito, viene eseguito il rollback della fase.

Si consideri una pipeline che ha:

- Una fase di produzione con rollback.
- Un gruppo di attività parallele, ciascuna con il proprio rollback.

L'attività denominata **UPD Deploy US** ha la pipeline di rollback **RB_Deploy_US**. Se **UPD Deploy US** non riesce, il rollback segue il flusso definito nella pipeline **RB_Deploy_US**.

RollbackUpgrade-Example Enabled

Workspace Input **Model** Output

Task : *UPD Deploy US* Notifications **Rollback**

Pipeline *RB_Deploy_US*

Production

- UPD Deploy US (Kubernetes)
- UPD Deploy UK (Kubernetes)
- UPD Deploy AU (Kubernetes)
- + Parallel Task

+ Stage

SAVE **RUN** **CLOSE** Last saved 12 days ago

Se **UPD Deploy US** non riesce, la pipeline **RB_Deploy_US** viene eseguita dopo che anche **UPD Deploy UK** e **UPD Deploy AU** sono state completate o non sono riuscite. Il rollback non si verifica immediatamente dopo la mancata riuscita di **UPD Deploy US**. Poiché anche la fase di produzione ha il rollback, dopo l'esecuzione della pipeline **RB_Deploy_US**, viene eseguita la pipeline di rollback della fase.

Tutorial sull'utilizzo di Code Stream

5

Code Stream modella e supporta il ciclo di vita di rilascio di DevOps, testando e rilasciando in modo continuativo le applicazioni in ambienti di sviluppo e di produzione.

È già stato configurato tutto il necessario per poter utilizzare Code Stream. Vedere [Capitolo 2 Impostazione di Code Stream per modellare il processo di rilascio](#).

A questo punto, è possibile creare pipeline che automatizzano la creazione e il test del codice dello sviluppatore prima di rilasciarlo alla produzione. È possibile fare in modo che Code Stream distribuisca applicazioni tradizionali o basate su contenitore.

Tabella 5-1. Utilizzo di Code Stream nel ciclo di vita di DevOps

Funzionalità	Esempi di operazioni che è possibile eseguire
Utilizzare la funzionalità di creazione nativa in Code Stream.	<p>Creare pipeline di integrazione e consegna continue (CI/CD), integrazione continua (CI) e consegna continua (CD) che integrino, containerizzino e consegnino il codice in modo continuativo.</p> <ul style="list-style-type: none">■ Utilizzare un modello di pipeline smart che crei una pipeline per l'utente.■ Aggiungere manualmente fasi e attività a una pipeline.
Rilasciare le applicazioni e automatizzare i rilasci.	<p>Integrare e rilasciare le applicazioni in vari modi.</p> <ul style="list-style-type: none">■ Integrare in modo continuativo il codice da un repository GitHub o GitLab nella pipeline.■ Integrare un host Docker per eseguire le attività Integrazione continua come documentato in questo articolo del blog: Creazione di un host Docker per vRealize Automation Code Stream.■ Automatizzare la distribuzione dell'applicazione utilizzando un modello di cloud YAML.■ Automatizzare la distribuzione dell'applicazione in un cluster Kubernetes.■ Rilasciare l'applicazione in una distribuzione blu-verde.■ Integrare Code Stream con i propri strumenti di creazione, test e distribuzione.■ Utilizzare una REST API che integri Code Stream con altre applicazioni.
Tenere traccia di tendenze, metriche e indicatori di prestazioni chiave (KPI).	<p>Creare dashboard personalizzati e acquisire informazioni sulle prestazioni delle pipeline.</p>
Risolvere i problemi.	<p>Quando l'esecuzione di una pipeline non riesce, fare in modo che Code Stream crei un ticket Jira.</p>

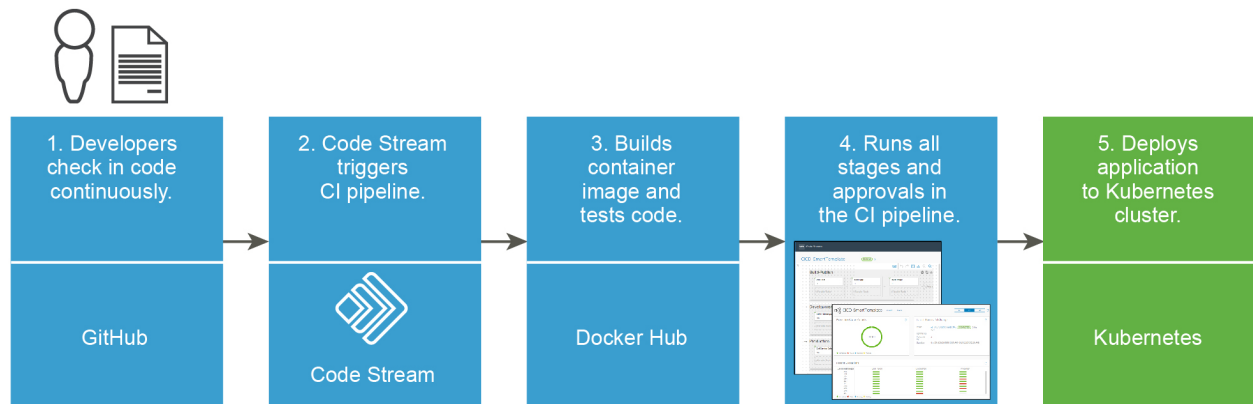
Questo capitolo include i seguenti argomenti:

- [Come integrare il codice in modo continuativo dal repository GitHub o GitLab nella pipeline in Code Stream](#)

- Come automatizzare il rilascio di un'applicazione da distribuire da un modello cloud YAML in Code Stream
- Come automatizzare il rilascio di un'applicazione in Code Stream in un cluster Kubernetes
- Come distribuire un'applicazione in Code Stream nella distribuzione blu-verde
- Integrazione degli strumenti di creazione, test e distribuzione con Code Stream
- Come utilizzare le proprietà delle risorse di un'attività del modello cloud nell'attività successiva
- Come utilizzare una REST API per integrare Code Stream con altre applicazioni
- Come sfruttare la pipeline come codice in Code Stream

Come integrare il codice in modo continuativo dal repository GitHub o GitLab nella pipeline in Code Stream

Uno sviluppatore può integrare in modo continuativo il codice da un repository GitHub o un repository GitLab Enterprise. Ogni volta che gli sviluppatori aggiornano il codice e confermano le modifiche nel repository, Code Stream ascolta le modifiche e attiva la pipeline.



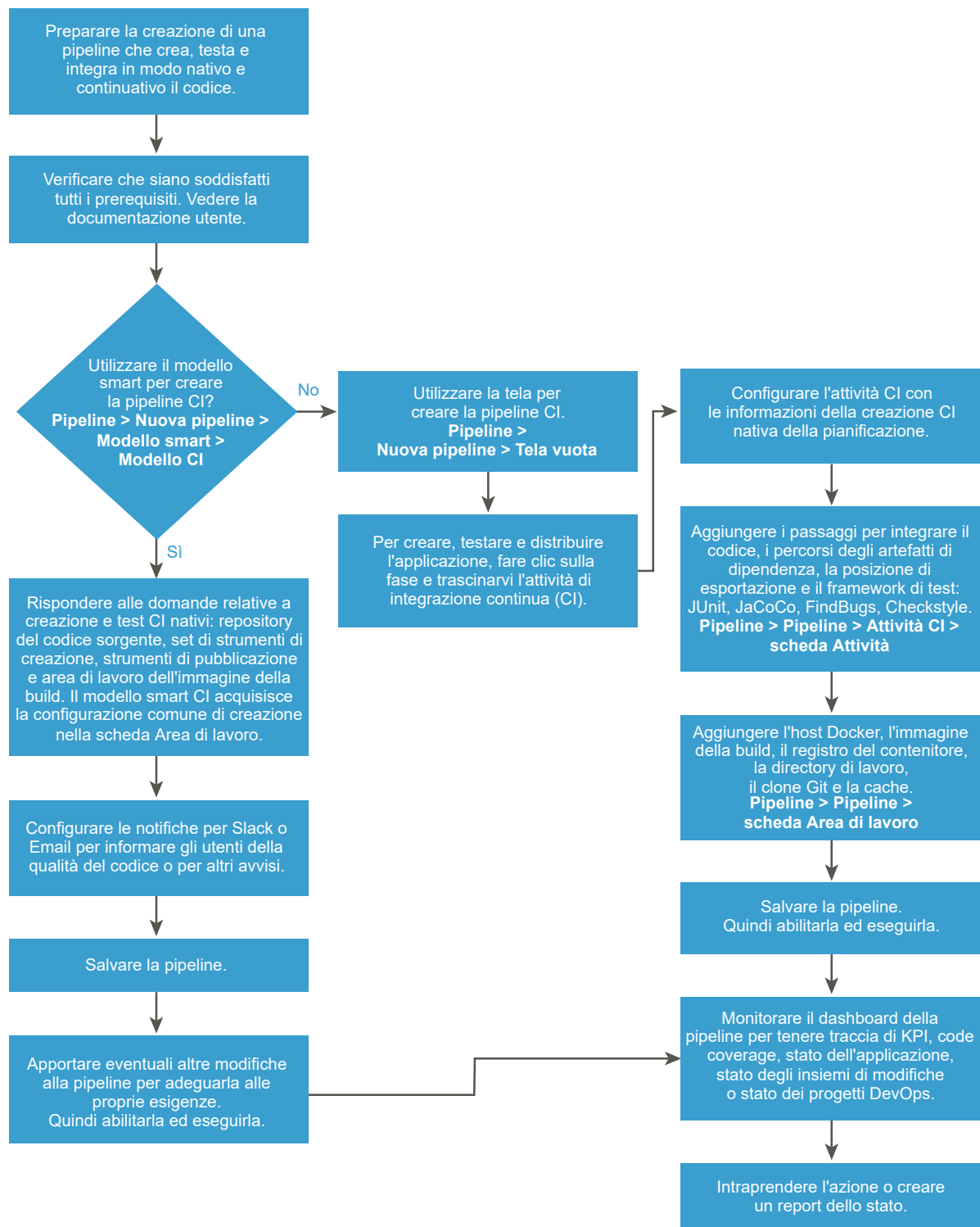
Per consentire a Code Stream di attivare la pipeline quando si verificano modifiche al codice, utilizzare il trigger Git. A questo punto, Code Stream attiva la pipeline ogni volta che verranno confermate le modifiche al codice.

L'area di lavoro della pipeline di Code Stream supporta Docker e Kubernetes per le attività di integrazione continua e le attività personalizzate.

Per ulteriori informazioni sulla configurazione dell'area di lavoro, vedere [Configurazione dell'area di lavoro della pipeline](#).

Il seguente diagramma di flusso mostra il workflow che è possibile eseguire se si utilizza un modello di pipeline smart per creare la pipeline o se la pipeline viene creata manualmente.

Figura 5-1. Workflow che utilizza un modello di pipeline smart o crea una pipeline manualmente



L'esempio seguente utilizza un'area di lavoro di Docker.

Per creare il codice, utilizzare un host Docker. È possibile utilizzare JUnit e JaCoCo come strumenti del framework di test, che eseguono unit test e code coverage, che verranno inclusi nella pipeline.

Quindi, utilizzare il modello di pipeline smart di integrazione continua per creare una pipeline di integrazione continua che crea, verifica e distribuisce il codice al cluster Kubernetes del team di progetto in AWS. Per archiviare gli artefatti di dipendenza del codice per l'attività di integrazione continua, che può consentire di risparmiare tempo nelle creazioni di codice, è possibile utilizzare una cache.

Nell'attività della pipeline che crea e verifica il codice, è possibile includere diversi passaggi di integrazione continua. Tali passaggi possono trovarsi nella stessa directory di lavoro in cui Code Stream clona il codice sorgente quando la pipeline viene attivata.

Per distribuire il codice nel cluster Kubernetes, è possibile utilizzare un'attività Kubernetes nella pipeline. È quindi necessario abilitare ed eseguire la pipeline. Apportare quindi una modifica al codice nel repository e osservare il trigger della pipeline. Per monitorare e creare report sulle tendenze della pipeline dopo l'esecuzione della pipeline, utilizzare i dashboard.

Nell'esempio seguente, per creare una pipeline di integrazione continua che integri in modo continuativo il codice nella pipeline, utilizzare il modello di pipeline smart di integrazione continua. Questo esempio utilizza un'area di lavoro di Docker.

Facoltativamente, è possibile creare manualmente la pipeline e aggiungervi fasi e attività. Per ulteriori informazioni sulla pianificazione di una creazione di integrazione continua e sulla creazione manuale della pipeline, vedere [Pianificazione di una creazione nativa di integrazione e consegna continue in Code Stream prima dell'aggiunta manuale di attività](#).

Prerequisiti

- Pianificare la creazione dell'integrazione continua. Vedere [Pianificazione di una creazione nativa di integrazione continua in Code Stream prima di utilizzare il modello di pipeline smart](#).
- Verificare che sia presente un repository del codice sorgente GitLab. Per ulteriori informazioni, consultare l'amministratore di Code Stream.
- Aggiungere un endpoint Git. Per un esempio, vedere [Come utilizzare il trigger Git in Code Stream per eseguire una pipeline](#).
- Per fare in modo che Code Stream resti in ascolto delle modifiche nel repository GitHub o nel repository GitLab e attivi una pipeline quando si verificano modifiche, aggiungere un webhook. Per un esempio, vedere [Come utilizzare il trigger Git in Code Stream per eseguire una pipeline](#).
- Aggiungere un endpoint host Docker, che crea un contenitore per l'attività di integrazione continua, utilizzabile da più attività di integrazione continua. Per ulteriori informazioni sugli endpoint, vedere [Che cosa sono gli endpoint in Code Stream](#).
- Ottenere l'URL dell'immagine, l'host della creazione e l'URL per l'immagine della creazione. Per ulteriori informazioni, consultare l'amministratore di Code Stream.

- Utilizzare JUnit e JaCoCo per gli strumenti del framework di test.
- Configurare un'istanza esterna per la creazione dell'integrazione continua: Jenkins, TFS o Bamboo. A questo punto, il plug-in Kubernetes distribuisce il codice. Per ulteriori informazioni, consultare l'amministratore di Code Stream.

Procedura

- 1 Seguire i prerequisiti.
- 2 Per creare la pipeline utilizzando il modello di pipeline smart, aprire il modello smart di integrazione continua e compilare il modulo.
 - a Fare clic su **Pipeline > Nuova pipeline > Modello smart > Integrazione continua**.
 - b Rispondere alle domande nel modello relative al repository del codice sorgente, ai set di strumenti di creazione, allo strumento di pubblicazione e all'area di lavoro dell'immagine della build.
 - c Aggiungere le notifiche Slack o le notifiche E-mail per il team.
 - d Per fare in modo che il modello di pipeline smart crei la pipeline, fare clic su **Crea**.
 - e Per apportare ulteriori modifiche alla pipeline, fare clic su **Modifica**, apportare le modifiche e fare clic su **Salva**.
 - f Abilitare la pipeline ed eseguirla.
- 3 Per creare la pipeline manualmente, aggiungere fasi e attività alla tela e includere le informazioni sulla creazione dell'integrazione continua nativa nell'attività di integrazione continua.
 - a Fare clic su **Pipeline > Nuova pipeline > Tela vuota**.
 - b Fare clic sulla fase, quindi trascinare le varie attività di integrazione continua dal riquadro di navigazione alla fase.
 - c Per configurare l'attività di integrazione continua, fare clic su di essa, quindi sulla scheda **Attività**.
 - d Aggiungere i passaggi che integrano il codice in modo continuativo.
 - e Includere i percorsi degli artefatti di dipendenza.
 - f Aggiungere la posizione di esportazione.
 - g Aggiungere gli strumenti del framework di test che verranno utilizzati.
 - h Aggiungere l'host Docker e l'immagine della build.
 - i Aggiungere il registro del contenitore, la directory di lavoro e la cache.
 - j Salvare la pipeline, quindi abilitarla.
- 4 Apportare una modifica al codice nel repository GitHub o nel repository GitLab.
Il trigger Git attiva la pipeline, che inizia a essere eseguita.

- 5 Per verificare che la modifica del codice abbia attivato la pipeline, fare clic su **Trigger** > **Git** > **Attività**.
- 6 Per visualizzare l'esecuzione della pipeline, fare clic su **Esecuzioni** e verificare che i passaggi abbiano creato ed esportato l'immagine della build.

The screenshot displays the vRealize Automation Code Stream interface. On the left, a sidebar contains navigation links: Dashboards, Executions, User Operations, Pipelines, Manage (with sub-links for Endpoints, Variables, and Triggers), and Triggers (with sub-links for Gerrit and Git). The main panel shows the execution details for 'CICD-SmartTemplate #51', which is marked as 'COMPLETED'. The pipeline consists of two stages: 'Build-Publish' and 'Development'. The 'Build-Publish' stage includes tasks 'Unit-Test', 'Build-App', and 'Build-Image', all of which are completed. The 'Development' stage includes tasks 'Create Namespace', 'Create Secret', 'Create Service', and 'Create Deployment', which are not yet started. The 'Build-Image' task is selected, showing its details: Task name 'Build-Image', Type 'CI', Status 'COMPLETED', Duration '5s', and a message 'Execution Completed.'. Below this, the 'Steps' section shows a list of commands executed successfully, including setting environment variables, logging into Docker, and building the image. The 'Preserved Artifacts' section shows the path to the build image artifacts. The 'Exports' section shows a table with one entry: 'IMAGE' with the value 'automation/cicd-smart-template:51'. The 'Process' section indicates 'No process results available.'.

- 7 Per monitorare il dashboard della pipeline per tenere traccia degli indicatori KPI e delle tendenze, fare clic su **Dashboard** > **Dashboard della pipeline**.

Risultati

Congratulazioni! È stata creata una pipeline che integra in modo continuativo il codice da un repository GitHub o un repository GitLab nella propria pipeline e distribuisce l'immagine della build.

Operazioni successive

Per ulteriori informazioni, vedere [Altre risorse per amministratori e sviluppatori di Code Stream](#).

Come automatizzare il rilascio di un'applicazione da distribuire da un modello cloud YAML in Code Stream

In qualità di sviluppatore, si necessita di una pipeline che recuperi un modello cloud di automazione da un'istanza di GitHub locale ogni volta che si conferma una modifica. È necessario che la pipeline distribuisca un'applicazione WordPress in Amazon Web Services (AWS) EC2 o in un data center. Code Stream richiama il modello cloud dalla pipeline e ne automatizza l'integrazione e la consegna continue (CI/CD) per distribuire l'applicazione.

Per creare e attivare la pipeline, è necessario un modello cloud di VMware.

In **Origine modello cloud** nell'attività del modello cloud di Code Stream, è possibile selezionare:

- **Modello Cloud Assembly** come controllo dell'origine. In questo caso, non è necessario un repository GitLab o GitHub.
- **Controllo origine** se si utilizza GitLab o GitHub per il controllo dell'origine. In questo caso, è necessario disporre di un webhook Git e attivare la pipeline tramite il webhook.

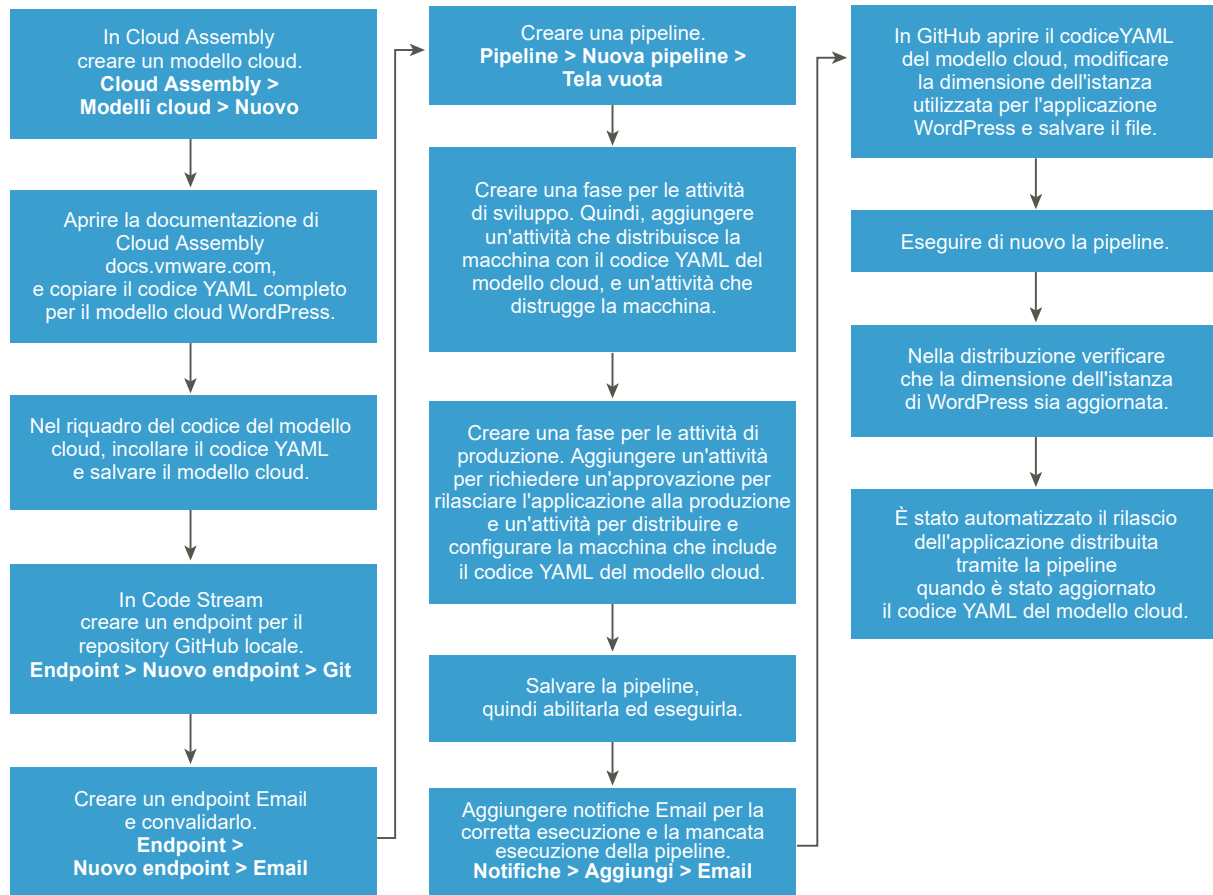
Se si dispone di un modello cloud YAML nel repository GitHub e si desidera utilizzare tale modello cloud nella pipeline, procedere nel modo seguente.

- 1 In Cloud Assembly, inserire il modello cloud nel repository GitHub.
- 2 In Code Stream, creare un endpoint Git. Creare quindi un webhook Git che utilizzi l'endpoint Git e la pipeline.
- 3 Per attivare la pipeline, aggiornare un file qualsiasi nel repository GitHub e confermare la modifica.

Se non si dispone di un modello cloud YAML nel repository GitHub e si desidera utilizzare un modello cloud dal controllo dell'origine, utilizzare questa procedura. Mostra come creare un modello cloud per un'applicazione WordPress e attivarlo da un repository GitHub locale. Ogni volta che si apporta una modifica al modello cloud YAML, la pipeline attiva e automatizza il rilascio dell'applicazione.

- In Cloud Assembly, aggiungere un account cloud e una zona cloud, quindi creare il modello cloud.
- In Code Stream, aggiungere un endpoint per il repository GitHub locale che ospita il modello cloud. Aggiungere quindi il modello cloud alla pipeline.

Questo esempio di caso d'uso illustra come utilizzare un modello cloud da un repository GitHub locale.

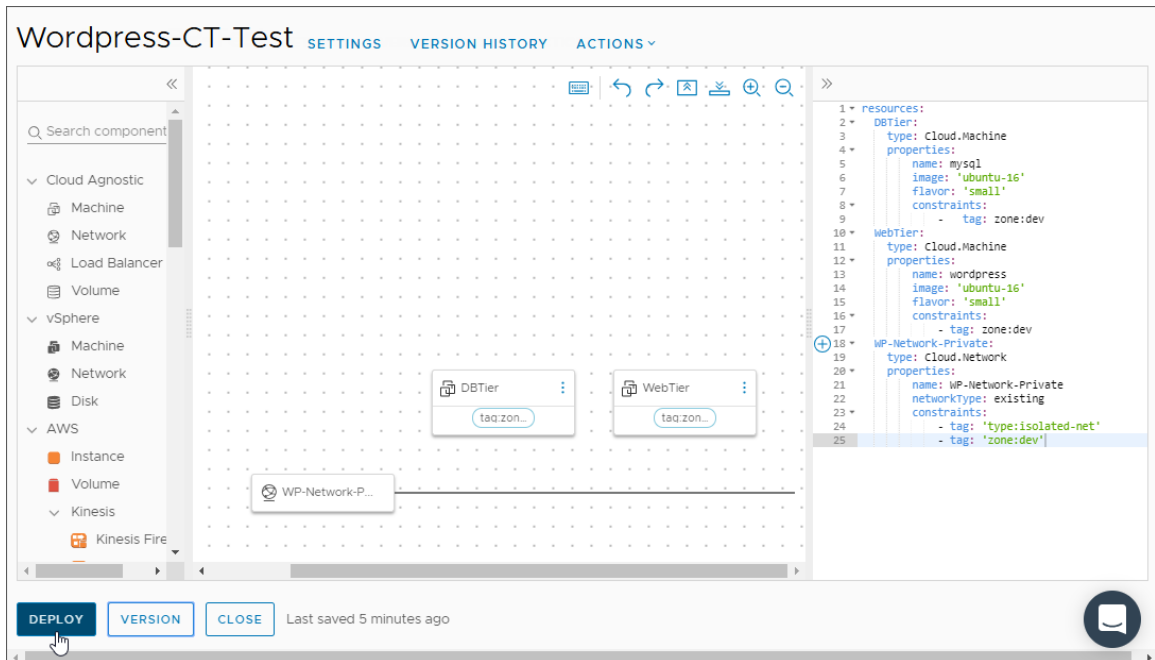


Prerequisiti

- Aggiungere un account cloud e una zona cloud nell'infrastruttura di vRealize Automation Cloud Assembly. Vedere la documentazione di vRealize Automation Cloud Assembly.
- Per creare il modello cloud nella procedura seguente, copiare il codice YAML di WordPress negli Appunti. Vedere il codice YAML del modello cloud nel caso d'uso di WordPress nella documentazione di vRealize Automation Cloud Assembly.
- Aggiungere il codice YAML per l'applicazione WordPress all'istanza di GitHub.
- Aggiungere un webhook per il trigger Git in modo che la pipeline possa estrarre il codice YAML ogni volta se ne confermano le modifiche. In Code Stream, fare clic su **Trigger > Git > Webhook per Git**.
- Per utilizzare un'attività del modello cloud, è necessario disporre di uno qualsiasi dei ruoli di Cloud Assembly.


Procedura

- 1 In Cloud Assembly eseguire i passaggi seguenti.
 - a Fare clic su **VMware Cloud Templates**, quindi creare un modello cloud e una distribuzione per l'applicazione WordPress.
 - b Incollare nel modello cloud il codice YAML di WordPress copiato negli Appunti e distribuirlo.



- 2 In Code Stream creare endpoint.
 - a Creare un endpoint Git per il repository GitHub locale in cui si trova il file YAML.
 - b Aggiungere un endpoint email che può notificare agli utenti lo stato della pipeline quando viene eseguita.

Add Endpoint

Project *	Codestream ▾
Type *	Email ▾
Name *	Enter value here 
Description	<input type="text"/>
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 Creare una pipeline e aggiungere notifiche relative all'esito della pipeline.

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 Aggiungere una fase per lo sviluppo e aggiungere un'attività Modello cloud.

- a Aggiungere un'attività modello cloud che distribuisce la macchina e configurarla in modo che utilizzi il codice YAML del modello cloud per l'applicazione 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 Aggiungere un'attività modello cloud che elimini definitivamente la macchina per liberare risorse.

5 Aggiungere una fase per la produzione e includere le attività di approvazione e distribuzione.

- a Aggiungere un'attività operazione utente per richiedere l'approvazione per rilasciare l'applicazione WordPress alla produzione.
- b Aggiungere un'attività modello cloud per distribuire la macchina e configurarla con il codice YAML del modello cloud per l'applicazione WordPress.

Quando si seleziona **Crea**, il nome della distribuzione deve essere univoco. Se il nome non viene specificato, Code Stream assegna un nome casuale univoco.

Ecco cosa è necessario sapere se si seleziona **Rollback** nel proprio caso d'uso: se si seleziona l'azione **Rollback** e si immette una **Versione di rollback**, la versione deve essere nel formato **n-X**. Ad esempio, **n-1**, **n-2**, **n-3** e così via. Se si crea e si aggiorna la distribuzione in una posizione diversa da Code Stream, il rollback non è consentito.

Quando si accede a Code Stream, viene recuperato un token utente, che ha una validità di 30 minuti. Per le durate della pipeline con esecuzione prolungata, quando l'esecuzione dell'attività che precede l'attività del modello cloud richiede 30 minuti o più, il token utente scade. Di conseguenza, l'attività del modello cloud non riesce.

Per garantire che la pipeline possa durare più di 30 minuti, è possibile immettere un token API facoltativo. Quando Code Stream richiama il modello cloud, il token API persiste e l'attività del modello cloud continua a utilizzare il token API.

Quando si utilizza il token API come variabile, viene crittografato. In caso contrario, viene utilizzato come testo normale.

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

- Task name:** Deploy CT
- Type:** VMware cloud template (dropdown)
- 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-- (dropdown)
 - Version:** --Select template Version-- (dropdown)
- Output Parameters:** (empty section)

6 Eseguire la pipeline.

Per verificare che ogni attività sia stata completata, fare clic sull'attività nell'esecuzione e verificare lo stato nei dettagli della distribuzione per visualizzare informazioni dettagliate sulle risorse.

7 In GitHub modificare la caratteristica dell'istanza del server WordPress da `small` a `medium`.

Quando si confermano le modifiche, la pipeline viene attivata. Estrae il codice aggiornato dal repository GitHub e crea l'applicazione.

```

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

```

8 Eseguire di nuovo la pipeline, verificare che venga completata e che abbia modificato la caratteristica dell'istanza di WordPress da `small` a `medium`.

Risultati

Congratulazioni! È stata automatizzata la versione dell'applicazione distribuita da un modello cloud YAML.

Operazioni successive

Per ulteriori informazioni su come utilizzare Code Stream, vedere [Capitolo 5 Tutorial sull'utilizzo di Code Stream](#).

Per ulteriori riferimenti, vedere [Altre risorse per amministratori e sviluppatori di Code Stream](#).

Come automatizzare il rilascio di un'applicazione in Code Stream in un cluster Kubernetes

In qualità di amministratore o sviluppatore di Code Stream, è possibile utilizzare Code Stream e VMware Tanzu Kubernetes Grid Integrated Edition (in precedenza denominato VMware Enterprise PKS) per automatizzare la distribuzione delle applicazioni software in un cluster Kubernetes. Questo caso d'uso menziona altri metodi che è possibile utilizzare per automatizzare il rilascio dell'applicazione.

In questo caso d'uso verrà creata una pipeline che include due fasi e si utilizzerà Jenkins per creare e distribuire l'applicazione.

- La prima fase è per lo sviluppo. Si utilizza Jenkins per estrarre il codice da un ramo nel repository GitHub, quindi lo si crea, lo si testa e lo si pubblica.

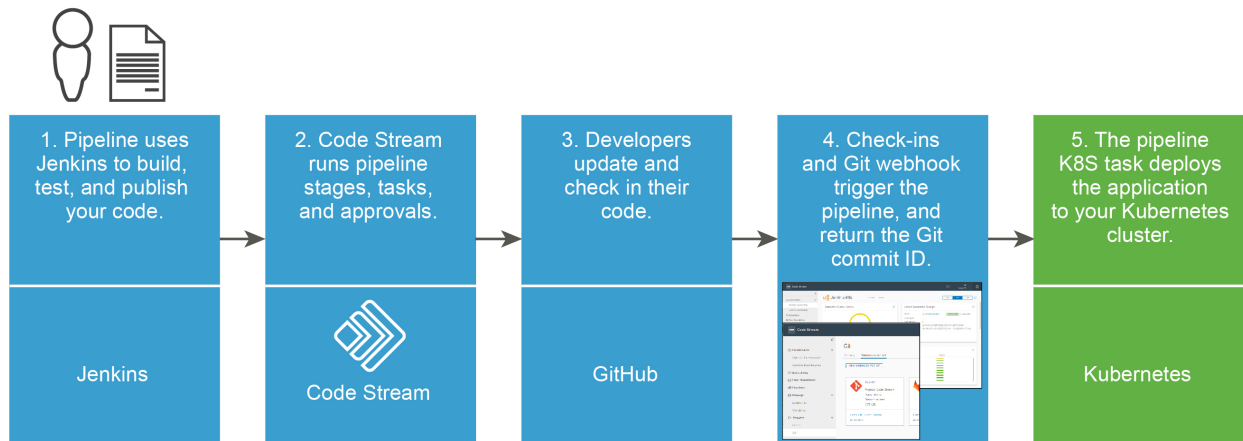
- La seconda fase è per la distribuzione. Esegue un'attività operazione utente che richiede l'approvazione da parte degli utenti chiave per consentire alla pipeline di distribuire l'applicazione nel cluster Kubernetes.

Quando si utilizza un endpoint API Kubernetes nell'area di lavoro della pipeline, Code Stream crea le risorse Kubernetes necessarie, come ConfigMap, Secret e Pod per eseguire l'attività di integrazione continua (CI) o l'attività personalizzata. Code Stream comunica con il contenitore tramite la NodePort.

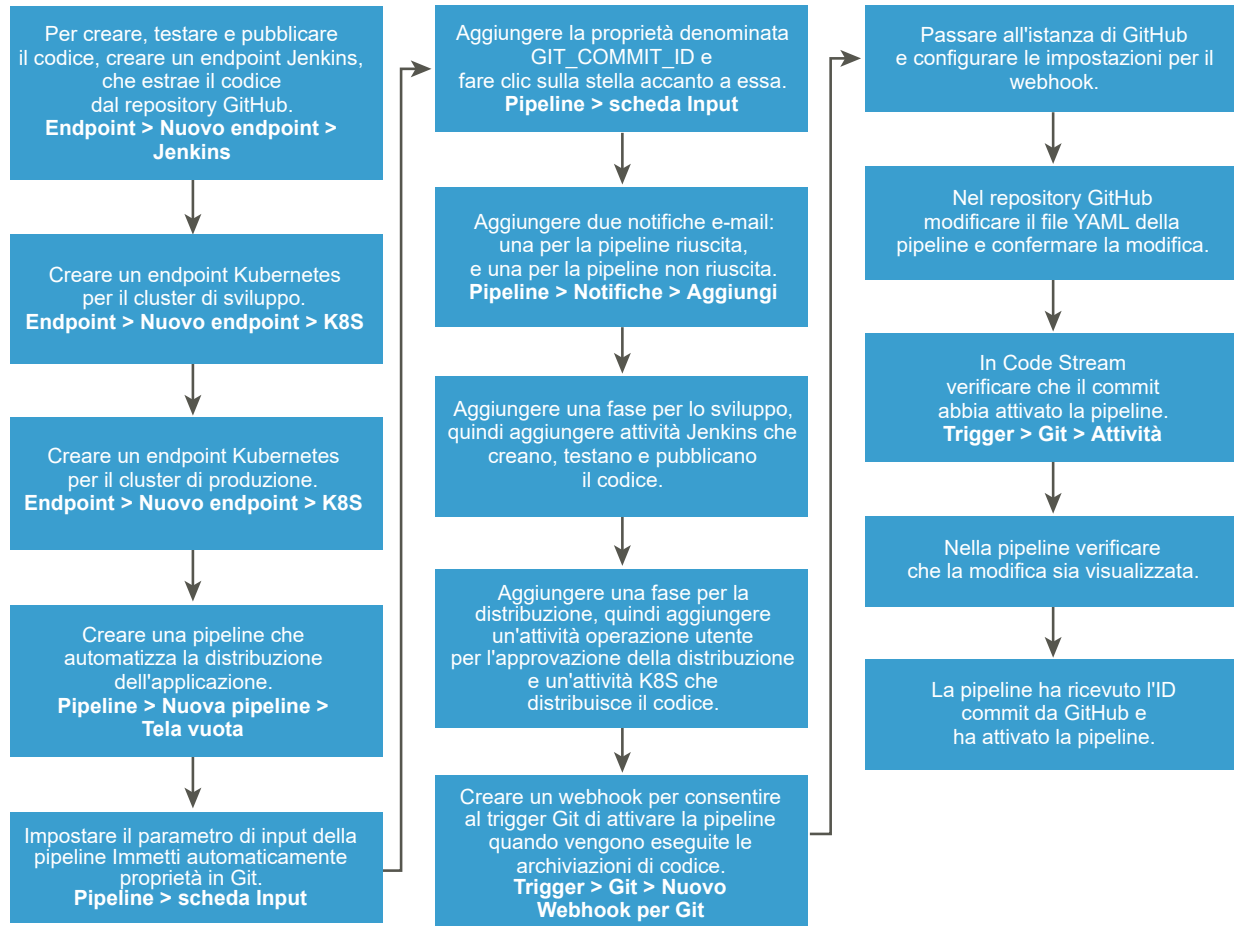
Per condividere i dati tra le esecuzioni della pipeline, è necessario fornire una richiesta volume persistente e Code Stream monterà la richiesta volume persistente nel contenitore per archiviare i dati e utilizzarla per le esecuzioni della pipeline successive.

L'area di lavoro della pipeline di Code Stream supporta Docker e Kubernetes per le attività di integrazione continua e le attività personalizzate.

Per ulteriori informazioni sulla configurazione dell'area di lavoro, vedere [Configurazione dell'area di lavoro della pipeline](#).



Affinché una pipeline possa creare, testare, pubblicare e distribuire l'applicazione, devono essere disponibili gli strumenti di sviluppo, le istanze di distribuzione e il file YAML della pipeline. La pipeline distribuirà l'applicazione nelle istanze di sviluppo e produzione dei cluster Kubernetes in AWS.



Altri metodi che automatizzano il rilascio dell'applicazione:

- Aniché creare l'applicazione utilizzando Jenkins, è possibile utilizzare la funzionalità di creazione nativa di Code Stream e un host di creazione Docker.
- Aniché distribuire l'applicazione in un cluster Kubernetes, è possibile distribuirla in un cluster Amazon Web Services (AWS).

Per ulteriori informazioni sull'utilizzo della funzionalità di creazione nativa di Code Stream e di un host Docker, vedere:

- Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart
- Pianificazione di una creazione nativa di integrazione e consegna continue in Code Stream prima dell'aggiunta manuale di attività

Prerequisiti

- Verificare che il codice dell'applicazione da distribuire si trovi in un repository GitHub funzionante.
- Verificare di disporre di un'istanza di Jenkins funzionante.
- Verificare di disporre di un server e-mail funzionante.

- In Code Stream creare un endpoint e-mail che si connetta al server e-mail.
- Configurare due cluster Kubernetes in Amazon Web Services (AWS), per lo sviluppo e la produzione, in cui la pipeline distribuirà l'applicazione.
- Verificare che il repository GitHub contenga il codice YAML per la propria pipeline e, in alternativa, un file YAML che definisca i metadati e le specifiche per l'ambiente in uso.

Procedura

- 1 In Code Stream fare clic su **Endpoint > Nuovo endpoint** e creare un endpoint Jenkins da utilizzare nella pipeline per estrarre il codice dal repository GitHub.
- 2 Per creare endpoint Kubernetes, fare clic su **Nuovo endpoint**.
 - a Creare un endpoint per il cluster Kubernetes di sviluppo.
 - b Creare un endpoint per il cluster Kubernetes di produzione.

L'URL del cluster Kubernetes potrebbe o meno includere un numero di porta.

Ad esempio:

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

```
https://api.kubernetesserver.fa2c1d78-9f00-4e30-8268-4ab81862080d.k8s-user.com
```
- 3 Creare una pipeline che distribuisca un contenitore dell'applicazione, ad esempio Wordpress, nel cluster Kubernetes di sviluppo e impostare le proprietà di input per la pipeline.
 - a Per consentire alla pipeline di riconoscere un commit di codice in GitHub che attiva la pipeline, nella pipeline fare clic sulla scheda **Input** e selezionare **Immetti automaticamente proprietà**.
 - b Aggiungere la proprietà denominata **GIT_COMMIT_ID**, quindi fare clic sulla stella accanto a tale proprietà.

Quando la pipeline viene eseguita, l'esecuzione visualizza l'ID commit restituito dal trigger Git.

Jenkins-K8s Enabled

Dev

- Build-DemoApp (Jenkins)
- Test-DemoApp (Jenkins)
- Publish-DemoApp (Jenkins)

Deploy

- Approve-Deployment (UserOperation)
- tpm-K8s-AWS (K8s)
- Sequential Task

Input

Pipeline Input Parameters

Auto inject properties: ☐ Gerrit ☒ Git ☐ None

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 Aggiungere notifiche per l'invio di un'e-mail quando la pipeline viene eseguita correttamente o non riesce.
 - a Nella pipeline fare clic sulla scheda **Notifiche**, quindi su **Aggiungi**.
 - b Per aggiungere una notifica email al termine dell'esecuzione della pipeline, selezionare **Email** e quindi **Completamento**. Quindi, selezionare il server e-mail, immettere gli indirizzi e-mail e fare clic su **Salva**.
 - c Per aggiungere un'altra notifica email da inviare quando si verifica un errore della pipeline, selezionare **Non riuscito** e fare clic su **Salva**.

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

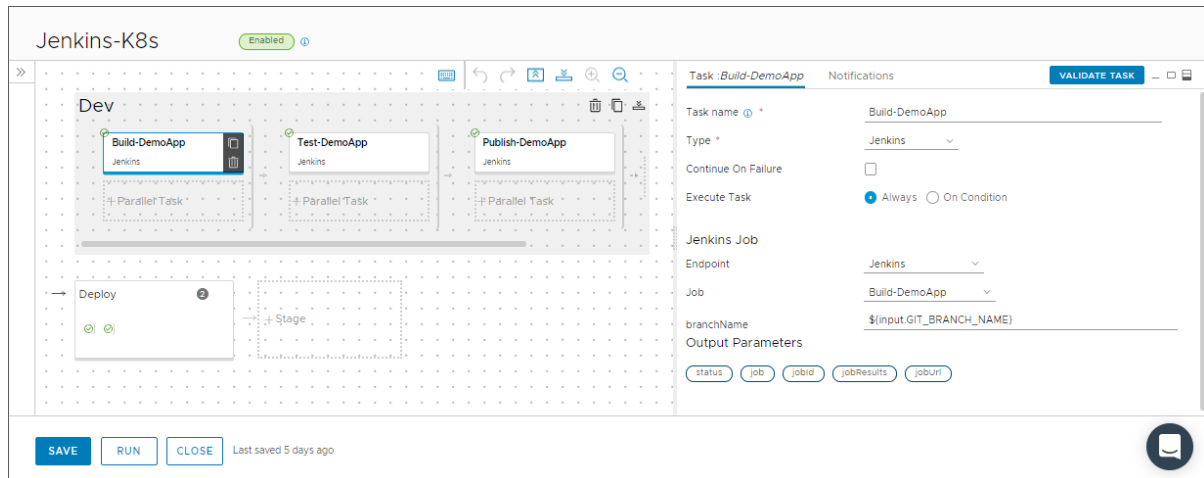
To ⓘ \$ * Email IDs of recipients

Subject \$ * Email Subject

Body ⓘ \$ *

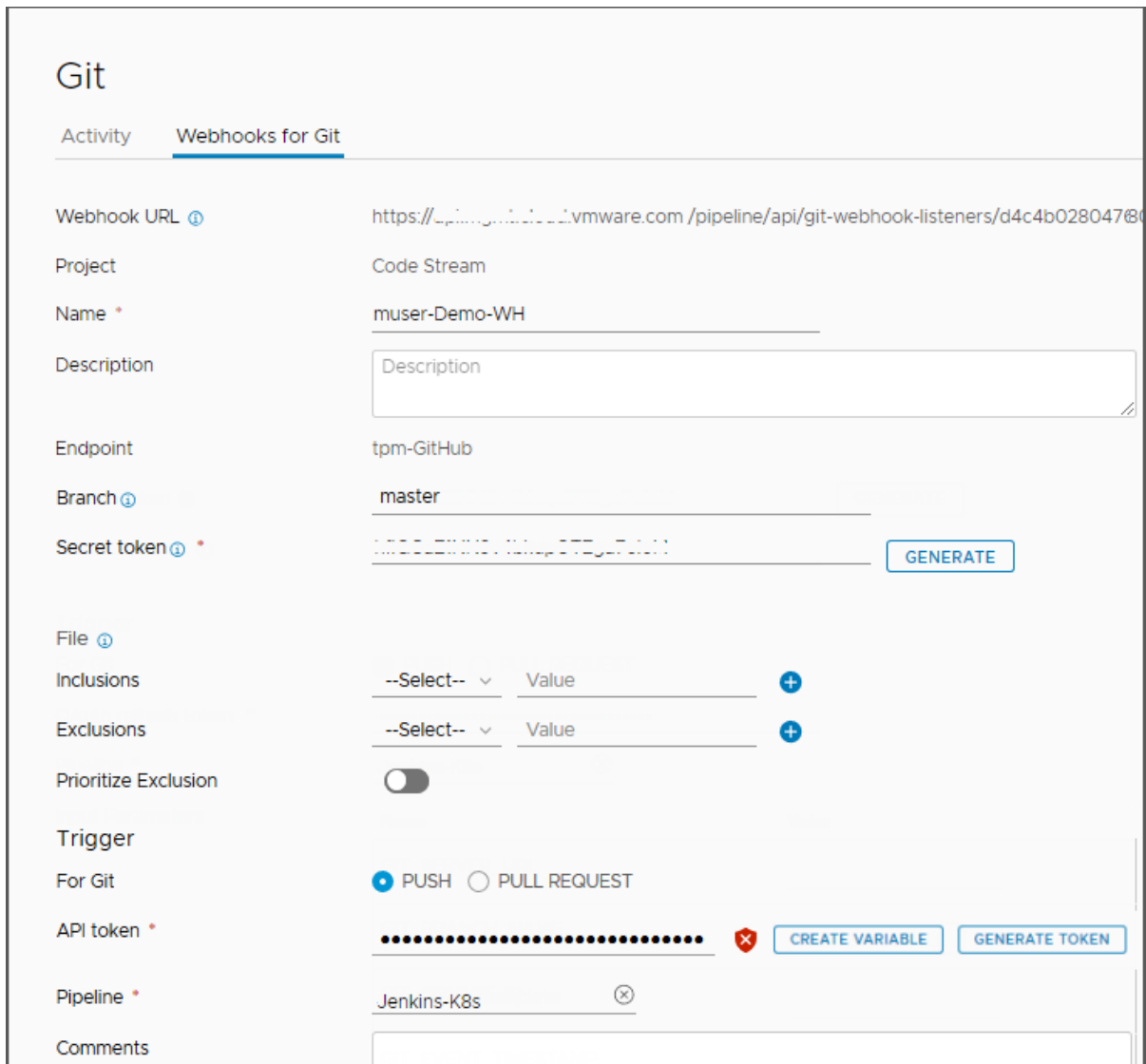
CANCEL **SAVE**

- 5 Aggiungere una fase di sviluppo alla pipeline e aggiungere attività che creano, testano e pubblicano l'applicazione. Quindi, convalidare ogni attività.
 - a Per creare l'applicazione, aggiungere un'attività Jenkins che utilizzi l'endpoint Jenkins ed esegua un processo di creazione dal server Jenkins. Per consentire alla pipeline di estrarre il codice, immettere il ramo Git nel formato: `${input.GIT_BRANCH_NAME}`
 - b Per testare l'applicazione, aggiungere un'attività Jenkins che utilizzi lo stesso endpoint Jenkins ed esegua un processo di test dal server Jenkins. Immettere lo stesso ramo Git.
 - c Per pubblicare l'applicazione, aggiungere un'attività Jenkins che utilizzi lo stesso endpoint Jenkins, quindi eseguire un processo di pubblicazione dal server Jenkins. Immettere lo stesso ramo Git.



- 6 Aggiungere una fase di distribuzione alla pipeline, quindi aggiungere un'attività che richiede un'approvazione per la distribuzione dell'applicazione e un'altra attività che distribuisce l'applicazione nel cluster Kubernetes. Quindi, convalidare ogni attività.
 - a Per richiedere un'approvazione per la distribuzione dell'applicazione, aggiungere un'attività operazione utente, aggiungere gli indirizzi e-mail degli utenti che devono approvarla e immettere un messaggio. Quindi, abilitare **Invia email**.
 - b Per distribuire l'applicazione, aggiungere un'attività Kubernetes. Quindi, nelle proprietà dell'attività Kubernetes, selezionare il cluster Kubernetes di sviluppo, selezionare l'azione **Crea** e selezionare l'origine payload **Definizione locale**. Selezionare il file YAML locale.

- 7 Aggiungere un webhook Git che consenta a Code Stream di utilizzare il trigger Git, che attiva la pipeline quando gli sviluppatori eseguono il commit del proprio codice.



- 8 Per eseguire il test della pipeline, passare al repository GitHub, aggiornare il file YAML dell'applicazione e confermare la modifica.
- In Code Stream verificare che il commit venga visualizzato.
 - Fare clic su **Trigger > Git > Attività**.
 - Cercare il trigger della pipeline.
 - Fare clic su **Dashboard > Dashboard della pipeline**.
 - Nel dashboard della pipeline individuare GIT_COMMIT_ID nell'area della modifica riuscita più recente.
- 9 Controllare il codice della pipeline e verificare che la modifica venga visualizzata.

Risultati

Congratulazioni! È stata automatizzata la distribuzione dell'applicazione software al cluster Kubernetes.

Esempio: YAML della pipeline di esempio che distribuisce un'applicazione in un cluster Kubernetes

Per il tipo di pipeline utilizzato in questo esempio, YAML è simile al seguente codice:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
eyJzeWlwaG9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJlc2VybmFtZSI6InRhbmRvLWJldGEyIiwicGFzc3dvcmQI Oi
JhRGstcmVOLWlUQil1IejciLCJlbWVpY2I6InRhbmRvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoizEdGdVoyOHRZbVYw
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

```

Operazioni successive

Per distribuire l'applicazione software al cluster Kubernetes di produzione, eseguire di nuovo i passaggi e selezionare il cluster di produzione.

Per ulteriori informazioni sull'integrazione di Code Stream con Jenkins, vedere [Come integrare Code Stream con Jenkins](#).

Come distribuire un'applicazione in Code Stream nella distribuzione blu-verde

Blu-verde è un modello di distribuzione che utilizza due host Docker che vengono distribuiti e configurati in modo identico in un cluster Kubernetes. Con il modello di distribuzione blu-verde, è possibile ridurre i tempi di inattività che si possono verificare nell'ambiente quando le pipeline in Code Stream distribuiscono le applicazioni.

Le istanze blu e verde nel modello di distribuzione hanno funzioni diverse. Solo un'istanza alla volta accetta il traffico attivo che distribuisce l'applicazione e ogni istanza accetta il traffico in un momento specifico. L'istanza blu riceve la prima versione dell'applicazione e l'istanza verde riceve la seconda.

Il bilanciamento del carico nell'ambiente blu-verde determina il routing del traffico attivo per la distribuzione dell'applicazione. Utilizzando il modello blu-verde, l'ambiente rimane operativo, gli utenti non notano alcun tempo di inattività e la pipeline integra e distribuisce in modo continuativo l'applicazione all'ambiente di produzione.

La pipeline creata in Code Stream rappresenta il modello di distribuzione blu-verde in due fasi. Una fase è per lo sviluppo e l'altra fase è per la produzione.

L'area di lavoro della pipeline di Code Stream supporta Docker e Kubernetes per le attività di integrazione continua e le attività personalizzate.

Per informazioni sulla configurazione dell'area di lavoro, vedere [Configurazione dell'area di lavoro della pipeline](#).

Tabella 5-2. Attività della fase di sviluppo per la distribuzione blu-verde

Tipo di attività	Attività
Kubernetes	Creare uno spazio dei nomi per la distribuzione blu-verde.
Kubernetes	Creare una chiave segreta per Docker Hub.
Kubernetes	Creare il servizio utilizzato per distribuire l'applicazione.
Kubernetes	Creare la distribuzione blu.
Polling	Verificare la distribuzione blu.
Kubernetes	Rimuovere lo spazio dei nomi.

Tabella 5-3. Attività della fase di produzione per la distribuzione blu-verde

Tipo di attività	Attività
Kubernetes	Verde ottiene i dettagli del servizio da blu.
Kubernetes	Recuperare i dettagli per il set di repliche verde.
Kubernetes	Creare la distribuzione verde e utilizzare la chiave segreta per estrarre l'immagine del contenitore.
Kubernetes	Aggiornare il servizio.
Polling	Verificare che la distribuzione sia stata completata nell'URL di produzione.
Kubernetes	Completare la distribuzione blu.
Kubernetes	Rimuovere la distribuzione blu.

Per distribuire l'applicazione nel proprio modello di distribuzione blu-verde, creare una pipeline in Code Stream che includa due fasi. La prima fase include le attività blu che distribuiscono l'applicazione all'istanza blu, mentre la seconda include attività verdi che distribuiscono l'applicazione all'istanza verde.

È possibile creare la pipeline utilizzando il modello di pipeline smart CICD. Il modello crea le fasi e le attività della pipeline per l'utente e include le selezioni della distribuzione.

Se si crea la pipeline manualmente, è necessario pianificare le fasi della pipeline. Per un esempio, vedere [Pianificazione di una creazione nativa di integrazione e consegna continue in Code Stream prima dell'aggiunta manuale di attività](#).

In questo esempio viene utilizzato il modello di pipeline smart CICD per creare la pipeline blu-verde.

Prerequisiti

- Verificare che sia possibile accedere a un cluster Kubernetes funzionante in AWS.

- Verificare di avere impostato un ambiente di distribuzione blu-verde e di aver configurato le istanze blu-verde in modo che siano identiche.
- Creare un endpoint Kubernetes in Code Stream che distribuisca l'immagine dell'applicazione nel cluster Kubernetes in AWS.
- Acquisire familiarità con l'utilizzo del modello di pipeline smart CICD. Vedere [Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart](#).

Procedura

- 1 Fare clic su **Pipeline > Nuova pipeline > Modelli smart > Modello CI/CD**.
- 2 Immettere le informazioni per la parte CI del modello di pipeline smart CICD e fare clic su **Avanti**.

Per ulteriori informazioni, vedere [Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart](#).
- 3 Completare la parte CD del modello di pipeline smart
 - a Selezionare gli ambienti per la distribuzione dell'applicazione. Ad esempio, **Sviluppo** e **Produzione**.
 - b Selezionare il servizio che la pipeline utilizzerà per la distribuzione.
 - c Nell'area di distribuzione selezionare l'endpoint del cluster per l'ambienti di sviluppo e l'ambiente di produzione.
 - d Per il modello di distribuzione della produzione, selezionare **Blu-verde** e fare clic su **Crea**.

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 *



Risultati

Congratulazioni! È stato utilizzato il modello di pipeline smart per creare una pipeline che distribuisca l'applicazione alle istanze blu-verde nel cluster di produzione Kubernetes in AWS.

Esempio: Codice YAML di esempio per alcune attività di distribuzione blu-verde

Il codice YAML visualizzato nelle attività della pipeline Kubernetes per la distribuzione blu-verde potrebbe essere simile agli esempi seguenti che creano lo spazio dei nomi, il servizio e la distribuzione. Se è necessario scaricare un'immagine da un repository di proprietà privata, il file YAML deve includere una sezione con il segreto di configurazione di Docker. Vedere la parte CD di [Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream](#) prima di utilizzare il modello di pipeline smart.

Quando il modello di pipeline smart avrà creato la pipeline, è possibile modificare le attività in base alle esigenze per la propria distribuzione.

Codice YAML per creare uno spazio dei nomi di esempio:

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

Codice YAML per creare un servizio di esempio:

```
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
```

Codice YAML per creare una distribuzione di esempio:

```
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
```

Operazioni successive

Per ulteriori informazioni su come utilizzare Code Stream, vedere [Capitolo 5 Tutorial sull'utilizzo di Code Stream](#).

Per eseguire il rollback di una distribuzione, vedere [Come eseguire il rollback di una distribuzione in Code Stream](#).

Per ulteriori riferimenti, vedere [Altre risorse per amministratori e sviluppatori di Code Stream](#).

Integrazione degli strumenti di creazione, test e distribuzione con Code Stream

In qualità di amministratore o di sviluppatore di DevOps, è possibile creare script personalizzati che estendano la funzionalità di Code Stream.

Grazie allo script, è possibile integrare Code Stream con gli strumenti di integrazione continua (CI) e di consegna continua (CD), nonché con le API che creano, testano e distribuiscono le applicazioni. Gli script personalizzati sono particolarmente utili se le API dell'applicazione non vengono esposte pubblicamente.

Lo script personalizzato può svolgere quasi tutte le operazioni necessarie per l'integrazione degli strumenti di creazione, test e distribuzione con Code Stream. Ad esempio, lo script può interagire con l'area di lavoro della pipeline per supportare le attività di integrazione continua che creano e testano l'applicazione e le attività di consegna continua che la distribuiscono. Può inviare un messaggio a Slack quando una pipeline termina e molto altro.

L'area di lavoro della pipeline di Code Stream supporta Docker e Kubernetes per le attività di integrazione continua e le attività personalizzate.

Per ulteriori informazioni sulla configurazione dell'area di lavoro, vedere [Configurazione dell'area di lavoro della pipeline](#).

È possibile scrivere lo script personalizzato in uno dei linguaggi supportati. Nello script è necessario includere la logica di business e definire input e output. I tipi di output possono includere numero, stringa, testo e password. È possibile creare più versioni di uno script personalizzato con logica di business, input e output diversi.

È necessario che la pipeline esegua una versione dello script in un'attività personalizzata. Gli script creati si trovano nell'istanza di Code Stream.

Quando una pipeline utilizza un'integrazione personalizzata, se si tenta di eliminare l'integrazione personalizzata, viene visualizzato un messaggio di errore che indica che non è possibile eliminarla.

L'eliminazione di un'integrazione personalizzata comporta la rimozione di tutte le versioni dello script personalizzato. Se esiste una pipeline con un'attività personalizzata che utilizza una versione qualsiasi dello script, la pipeline non riuscirà. Per assicurarsi che le pipeline esistenti vengano eseguite correttamente, è possibile deprecare e ritirare la versione dello script che non si desidera più utilizzare. Se tale versione non viene utilizzata da alcuna pipeline, è possibile eliminarla.

Tabella 5-4. Cosa fare dopo aver scritto lo script personalizzato

Cosa fare...	Ulteriori informazioni su questa azione...
Aggiungere un'attività personalizzata alla pipeline.	<p>L'attività personalizzata:</p> <ul style="list-style-type: none"> ■ Viene eseguita nello stesso contenitore di altre attività CI nella pipeline. ■ Include le variabili di input e output che lo script popola prima che la pipeline esegua l'attività personalizzata. ■ Supporta più tipi di dati e vari tipi di metadati che vengono definiti come input e output nello script.
Selezionare lo script nell'attività personalizzata.	Le proprietà di input e output vengono dichiarate nello script.
Salvare la pipeline, quindi abilitarla ed eseguirla.	Quando la pipeline è in esecuzione, l'attività personalizzata richiama la versione dello script specificata e con tale versione esegue la logica di business, che integra lo strumento di creazione, test e distribuzione con Code Stream.
Una volta eseguita la pipeline, esaminare le esecuzioni.	Verificare che la pipeline abbia fornito i risultati previsti.

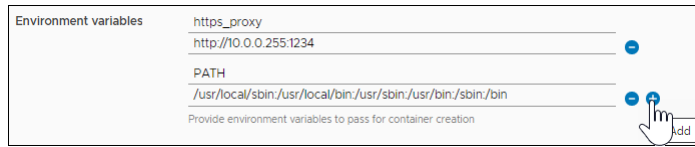
Quando si utilizza un'attività personalizzata che richiama una versione di Integrazione personalizzata, è possibile includere variabili di ambiente personalizzate come coppie nome-valore nella scheda **Area di lavoro** della pipeline. Quando l'immagine del generatore crea il contenitore dell'area di lavoro che esegue l'attività CI e distribuisce l'immagine, Code Stream passa le variabili di ambiente a tale contenitore.

Ad esempio, quando l'istanza di Code Stream richiede un proxy Web e si utilizza un host Docker per creare un contenitore per un'integrazione personalizzata, Code Stream esegue la pipeline e passa le variabili di impostazione del proxy Web a tale contenitore.

Tabella 5-5. Esempio di coppie nome-valore delle variabili di ambiente

Nome	Valore
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

Le coppie nome-valore vengono visualizzate nell'interfaccia utente nel modo seguente:



In questo esempio viene creata un'integrazione personalizzata che connette Code Stream all'istanza di Slack e invia un messaggio a un canale Slack.

Prerequisiti

- Per scrivere lo script personalizzato, verificare di disporre di uno di questi linguaggi: Python 2, Python 3, Node.js o di uno di questi linguaggi della shell: Bash, sh o zsh.
- Generare un'immagine del contenitore utilizzando il runtime Node.js o Python installato.

Procedura

1 Creare l'integrazione personalizzata.

- a Fare clic su **Integrazioni personalizzate > Nuovo** e immettere un nome pertinente.
- b Selezionare l'ambiente di runtime preferito.
- c Fare clic su **Crea**.

Lo script si apre e viene visualizzato il codice, che include l'ambiente di runtime richiesto. Ad esempio `runtime: "nodejs"`. Lo script deve includere il runtime, utilizzato dall'immagine del generatore, in modo che l'attività personalizzata aggiunta alla pipeline venga eseguita correttamente quando viene eseguita la pipeline. In caso contrario, l'attività personalizzata non riesce.

Le aree principali dello YAML dell'integrazione personalizzata includono runtime, codice, proprietà di input e proprietà di output. Questa procedura illustra diversi tipi e sintassi.

Chiavi YAML dell'integrazione personalizzata	Descrizione
runtime	<p>Ambiente di runtime dell'attività in cui Code Stream esegue il codice, che può essere una di queste stringhe senza distinzione tra maiuscole e minuscole:</p> <ul style="list-style-type: none"> ■ nodejs ■ python2 ■ python3 ■ shell <p>Se non viene specificato alcun valore, viene utilizzato shell per impostazione predefinita.</p>
code	Logica di business personalizzata da eseguire come parte dell'attività personalizzata.

Chiavi YAML dell'integrazione personalizzata	Descrizione
<code>inputProperties</code>	Array di proprietà di input da acquisire come parte della configurazione dell'attività personalizzata. Queste proprietà vengono in genere utilizzate nel codice.
<code>outputProperties</code>	Array di proprietà di output che è possibile esportare dall'attività personalizzata da propagare alla pipeline.

2 Dichiarare le proprietà di input nello script utilizzando i tipi di dati e i metadati disponibili.

Le proprietà di input vengono passate come contesto allo script nella sezione `code:` del linguaggio YAML.

Chiavi di input YAML per l'attività personalizzata	Descrizione	Obbligatoria
<code>type</code>	Tipi di input per il rendering: <ul style="list-style-type: none"> ■ <code>text</code> ■ <code>textarea</code> ■ <code>number</code> ■ <code>checkbox</code> ■ <code>password</code> ■ <code>select</code> 	Sì
<code>name</code>	Nome o stringa dell'input per l'attività personalizzata, che viene inserito nel codice YAML dell'integrazione personalizzata. Deve essere univoco per ogni proprietà di input definita per un'integrazione personalizzata.	Sì
<code>title</code>	Etichetta della stringa di testo della proprietà di input per l'attività personalizzata nella tela del modello di pipeline. Se viene lasciata vuota, per impostazione predefinita viene utilizzato il valore name .	No
<code>required</code>	Determina se un utente deve immettere la proprietà di input quando configura l'attività personalizzata. Impostare su <code>true</code> o su <code>false</code> . Quando è <code>true</code> , se un utente non specifica alcun valore quando configura l'attività personalizzata nella tela della pipeline, lo stato dell'attività rimane non configurata.	No
<code>placeholder</code>	Testo predefinito per l'area di immissione della proprietà di input quando non è presente alcun valore. Viene mappato all'attributo placeholder HTML. Supportata solo per determinati tipi di proprietà di input.	No
<code>defaultValue</code>	Valore predefinito che popola l'area di immissione della proprietà di input quando viene eseguito il rendering dell'attività personalizzata nella pagina del modello di pipeline.	No
<code>bindable</code>	Determina se la proprietà di input accetta le variabili con simbolo di dollaro quando si modella l'attività personalizzata nella tela della pipeline. Aggiunge l'indicatore <code>\$</code> accanto al titolo. Supportata solo per determinati tipi di proprietà di input.	No
<code>labelMessage</code>	Stringa che funge da descrizione comando della guida per gli utenti. Aggiunge un'icona descrizione comando <code>i</code> accanto al titolo di input.	No

Chiavi di input YAML per l'attività personalizzata	Descrizione	Obbligatoria
enum	<p>Acquisisce un array di valori che visualizza le opzioni delle proprietà di input selezionate. Supportata solo per determinati tipi di proprietà di input.</p> <p>Quando un utente seleziona un'opzione e la salva per l'attività personalizzata, il valore di inputProperty corrisponde a questo valore e viene visualizzato nella modellazione dell'attività personalizzata.</p> <p>Ad esempio, il valore 2015.</p> <ul style="list-style-type: none"> ■ 2015 ■ 2016 ■ 2017 ■ 2018 ■ 2019 ■ 2020 	No
options	<p>Acquisisce un array di oggetti utilizzando optionKey e optionValue.</p> <ul style="list-style-type: none"> ■ optionKey. Valore propagato alla sezione del codice dell'attività. ■ optionValue. Stringa che visualizza l'opzione nell'interfaccia utente. <p>Supportata solo per determinati tipi di proprietà di input.</p> <p>Opzioni:</p> <p>optionKey: key1. Quando viene selezionata e salvata per l'attività personalizzata, il valore di questa proprietà inputProperty corrisponde a key1 nella sezione del codice.</p> <p>optionValue: 'Etichetta per 1'. Valore visualizzato per key1 nell'interfaccia utente. Non viene visualizzato in alcun altro punto per l'attività personalizzata.</p> <p>optionKey: key2</p> <p>optionValue: 'Etichetta per 2'</p> <p>optionKey: key3</p> <p>optionValue: 'Etichetta per 3'</p>	No
minimum	Acquisisce un numero che funge da valore minimo valido per questa proprietà di input. Supportata solo per la proprietà di input di tipo number.	No
maximum	Acquisisce un numero che funge da valore massimo valido per questa proprietà di input. Supportata solo per la proprietà di input di tipo number.	No

Tabella 5-6. Metadati e tipi di dati supportati per script personalizzati

Tipi di dati supportati	Metadati supportati per l'input
<ul style="list-style-type: none"> ■ Stringa ■ Testo ■ Elenco: come elenco di qualsiasi tipo ■ Mappa: come map[string]any ■ Sicuri: rappresentati come casella di testo per la password, crittografati quando si salva l'attività personalizzata ■ Numero ■ Booleani: visualizzati come caselle di testo ■ URL: come stringa, con convalida aggiuntiva ■ Selezione, pulsante di opzione 	<ul style="list-style-type: none"> ■ type: Stringa Testo... ■ default: valore predefinito ■ options: elenco o mappa di opzioni da utilizzare con la selezione o il pulsante di opzione ■ min: valore o dimensione minimi ■ max: valore o dimensione massimi ■ title: nome dettagliato della casella di testo ■ placeholder: segnaposto dell'interfaccia utente ■ description: diventa una descrizione comando

Ad esempio:

```
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 Dichiarare le proprietà di output nello script.

Lo script acquisisce le proprietà di output dalla sezione `code`: della relativa business logic, dove si dichiara il contesto per l'output.

Quando la pipeline viene eseguita, è possibile immettere il codice di risposta per l'output dell'attività. Ad esempio **200**.

Chiavi che Code Stream supporta per ogni **outputProperty**.

Chiave	Descrizione
type	Attualmente include un singolo valore di label .
nome	Chiave emessa dal blocco di codice dello YAML dell'integrazione personalizzata.
title	Etichetta nell'interfaccia utente che visualizza outputProperty .

Ad esempio:

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

- 4 Per interagire con l'input e l'output dello script personalizzato, recuperare una proprietà di input o impostare una proprietà di output utilizzando **context**.

Per una proprietà di input: `(context.getInput("key"))`

Per una proprietà di output: `(context.setOutput("key", "value"))`

Per Node.js:

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

Per Python:

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

Per Shell:

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

- 5 Nella sezione `code`: dichiarare tutta la logica di business per l'integrazione personalizzata.

Ad esempio, con l'ambiente di runtime 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 Prima di eseguire la versione e rilasciare lo script di integrazione personalizzato, scaricare il file di contesto per Python o Node.js e verificare la logica di business inclusa nello script.
 - a Posizionare il puntatore nello script, quindi fare clic sul pulsante del file di contesto nella parte superiore della tela. Ad esempio, se lo script è in linguaggio Python fare clic su **CONTEXT.PY**.
 - b Modificare il file e salvarlo.
 - c Nel sistema di sviluppo eseguire e testare lo script personalizzato con l'aiuto del file di contesto.
- 7 Applicare una versione allo script di integrazione personalizzato.
 - a Fare clic su **Versione**.
 - b Inserire le informazioni sulla versione.
 - c Fare clic su **Versione di rilascio** per poter selezionare lo script nell'attività personalizzata.
 - d Per creare la versione, fare clic su **Crea**.

Creating Version

Version *

1.0

Description

New

Change Log

New for 1.0

Release Version @

☒

CANCEL

CREATE

8 Per salvare lo script, fare clic su **Salva**.

9 Nella pipeline, configurare l'area di lavoro.

Questo esempio utilizza un'area di lavoro di Docker.

- a Fare clic sulla scheda **Area di lavoro**.
- b Selezionare l'host Docker e l'URL immagine generatore.

The screenshot shows the 'Workspace' configuration page for a pipeline named 'Demo-customTask-nodejs', which is in an 'Enabled' state. The page has a tabbed interface with 'Workspace', 'Input', 'Model', and 'Output' tabs. The 'Workspace' tab is active, showing a list of configuration fields:

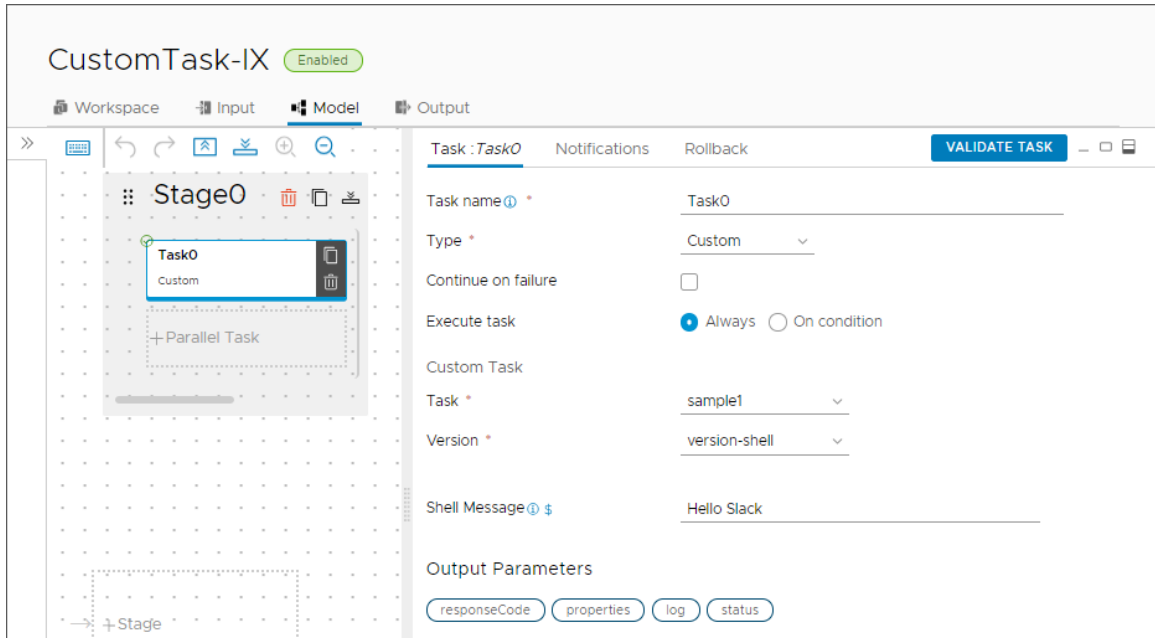
- Host**: Set to 'Docker-saas' (with a dropdown arrow).
- Builder image URL**: Set to 'node:latest'.
- Image registry**: Set to '--Select Container Registry Endpoint--' (with a dropdown arrow).
- Working directory**: An empty text field.
- Cache**: An empty text field with a blue plus icon to its right.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

At the bottom of the 'Workspace' tab, there is a text box with the following information: '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 Aggiungere un'attività personalizzata alla pipeline e configurarla.

- a Fare clic sulla scheda **Modello**.
- b Aggiungere un'attività, selezionare il tipo come **Personalizzato** e immettere un nome pertinente.

- c Selezionare lo script di integrazione personalizzato e la versione.
- d Per visualizzare un messaggio personalizzato in Slack, immettere il testo del messaggio.
Il testo immesso sostituisce `defaultValue` nello script di integrazione personalizzato. Ad esempio:



- 11 Salvare e abilitare la pipeline.
 - a Fare clic su **Salva**.
 - b Nella scheda Pipeline fare clic su **Attiva pipeline** in modo che il cerchio venga spostato a destra.
- 12 Eseguire la pipeline.
 - a Fare clic su **Esegui**.
 - b Osservare l'esecuzione della pipeline.

- c Verificare che l'output includa il codice di stato, il codice di risposta, lo stato e l'output dichiarato previsti.

È stato definito **statusCode** come proprietà di output. Ad esempio, uno **statusCode** di 200 potrebbe indicare un post creato correttamente in Slack e un **responseCode** di 0 potrebbe indicare che lo script è riuscito senza errori.

- d Per verificare l'output nei registri di esecuzione, fare clic su **Esecuzioni**, fare clic sul collegamento alla pipeline, quindi sull'attività e controllare i dati registrati. Ad esempio:

The screenshot displays the execution details for a task named "Task1" within a pipeline "custom-int-demo #5". The task is marked as "COMPLETED" with a status of "Execution Completed". The output shows "statusCode" as 200 and "Response code" as 0. A terminal log snippet is visible at the bottom.

Task name	Task1 VIEW OUTPUT JSON
Type	Custom
Status	COMPLETED 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>

[View Full Log](#)

13 Se si verifica un errore, risolvere il problema ed eseguire nuovamente la pipeline.

Ad esempio, se manca un file o un modulo nell'immagine di base, è necessario creare un'altra immagine di base che includa il file mancante. Quindi specificare il file Docker e inserire l'immagine attraverso la pipeline.

Risultati

Congratulazioni! È stato creato uno script di integrazione personalizzata che connette Code Stream all'istanza di Slack e invia un messaggio a un canale Slack.

Operazioni successive

Continuare a creare integrazioni personalizzate per supportare l'utilizzo di attività personalizzate nelle pipeline, in modo da poter estendere la funzionalità di Code Stream nell'automazione del ciclo di vita del rilascio del software.

Come utilizzare le proprietà delle risorse di un'attività del modello cloud nell'attività successiva

Quando si utilizza un'attività del modello cloud in Code Stream, ci si chiede spesso come utilizzare l'output di questa attività in un'attività successiva nella pipeline. Per utilizzare l'output di un'attività del modello cloud, ad esempio una macchina cloud, è necessario sapere come trovare le proprietà delle risorse nei dettagli della distribuzione dell'attività del modello cloud e l'indirizzo IP della macchina cloud.

Ad esempio, i dettagli della distribuzione di un modello cloud di VMware includono la risorsa macchina cloud e il relativo indirizzo IP. Nella pipeline, è possibile utilizzare la macchina cloud e l'indirizzo IP come variabile per associare un'attività del modello cloud a un'attività REST.

Il metodo utilizzato per trovare l'indirizzo IP per la macchina cloud non è tipico, perché è necessario che la distribuzione del modello cloud di VMware venga completata prima che i dettagli della distribuzione siano disponibili. È quindi possibile utilizzare le risorse della distribuzione del modello cloud di VMware per associare le attività della pipeline.

- Le proprietà delle risorse visualizzate in un'attività del modello cloud nella pipeline sono definite nel modello cloud di VMware in Cloud Assembly.
- È possibile che non si sappia quando una distribuzione di tale modello cloud sia terminata.
- Un'attività del modello cloud in Code Stream può visualizzare le proprietà di output del modello cloud di VMware solo al termine della distribuzione.

Questo esempio può essere particolarmente utile se si distribuisce un'applicazione e si richiamano varie API. Ad esempio, se si utilizza un'attività del modello cloud che richiama un modello cloud di VMware che distribuisce un'applicazione WordPress con una REST API, è possibile individuare l'indirizzo IP della macchina distribuita nei dettagli della distribuzione e utilizzare l'API per testarlo.

L'attività del modello cloud supporta l'utilizzo del binding della variabile mediante la visualizzazione dei dettagli con completamento automatico. È possibile scegliere la modalità di binding della variabile desiderata.

Questo esempio illustra come:

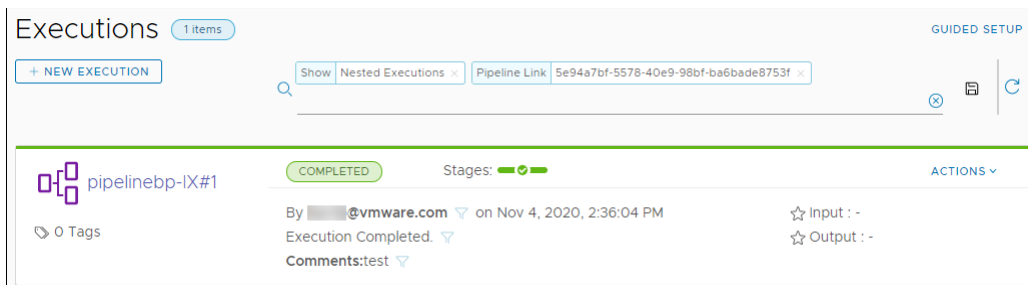
- Trovare i dettagli della distribuzione e le proprietà delle risorse per l'attività del modello cloud in una pipeline eseguita e completata.
- Individuare l'indirizzo IP della macchina cloud nella sezione delle risorse dei dettagli della distribuzione.
- Aggiungere un'attività REST successiva all'attività del modello cloud nella pipeline.
- Eseguire il binding dell'attività del modello cloud all'attività REST utilizzando l'indirizzo IP della macchina cloud nell'URL dell'attività REST.
- Eseguire la pipeline e controllare che il binding dall'attività del modello cloud all'attività REST funzioni.

Prerequisiti

- Verificare che sia presente un modello cloud di VMware funzionante con versione.
- Verificare che la distribuzione del modello cloud di VMware sia riuscita in Cloud Assembly.
- Verificare che sia presente una pipeline che includa un'attività del modello cloud che utilizza tale modello cloud di VMware.
- Verificare che la pipeline sia stata eseguita e completata correttamente.

Procedura

- 1 Nella pipeline, individuare l'indirizzo IP della macchina cloud nella sezione delle risorse dei dettagli della distribuzione dell'attività del modello cloud.
 - a Fare clic su **Azioni > Visualizza esecuzioni**.
 - b In un'esecuzione della pipeline riuscita, fare clic sul collegamento all'esecuzione della pipeline.



- c Sotto il nome della pipeline, fare clic sul collegamento all'**Attività**.

Project	bhawesh
Execution	pipelinebp-IX #1
Status	COMPLETED
Message	Execution Completed.

- d Nell'area Output, individuare i dettagli della distribuzione.

Task name: TaskO [VIEW OUTPUT JSON](#)

Type: VMware cloud template

Status: COMPLETED

Message: Execution Completed.

Duration: 0 milliseconds (Nov 4, 2020, 2:36:13 PM - Nov 4, 2020, 2:52:50 PM)

Precondition: -

Continue on failure: No

Output

Deployment: [deployment_c7185c47-1c12-40c5-9451-cbbbc4b16c89](#)

Deployment details

```

1 {
2   "id": "c7185c47-1c12-40c5-9451-cbbbc4b16c89",
3   "name": "deployment_c7185c47-1c12-40c5-9451-cbbbc4b16c89",
4   "description": "Pipeline Service triggered operation",
5   "orgId": "434f6917-4e34-4537-b6c0-30f3630871bc",
6   "blueprintId": "8d1dd801-3a32-4f30-adde-27f8163dfe6f",
7   "blueprintVersion": "4",
8   "createdAt": "2020-11-04T21:36:14.500036Z",
9   "createdBy": "kernb@vmware.com",
10  "lastUpdatedAt": "2020-11-04T21:52:45.243028Z",
11  "lastUpdatedBy": "kernb@vmware.com",
12  "inputs": {},
13  "simulated": false,
14  "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
15  "resources": {
16    "Cloud_Machine_1[0]": {
17      "id": "/resources/compute/f5a846f3-c97c-4145-9e28-951c36bd721c",
18      "name": "Cloud_Machine_1[0]",
19      "powerState": "ON"

```

Input

Action: Create Deployment

Cloud template: bhawesh

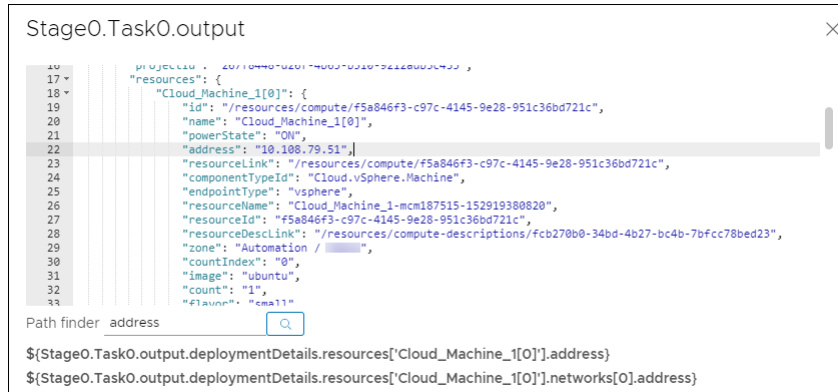
Cloud template version: 4

- e Nella sezione relativa alle risorse dei dettagli della distribuzione, individuare il nome della macchina cloud.

Includere quindi la sintassi del nome della macchina cloud nell'URL dell'attività REST.

- f Per individuare l'espressione di binding per la proprietà di output dell'attività del modello cloud, fare clic su **Visualizza JSON di output**, cercare la proprietà dell'indirizzo e individuare l'indirizzo IP della macchina cloud.

L'espressione di binding viene visualizzata sotto la proprietà e l'icona della ricerca nell'output JSON.



Nella proprietà della risorsa indirizzo viene visualizzato l'indirizzo IP della macchina cloud.
Ad esempio:

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

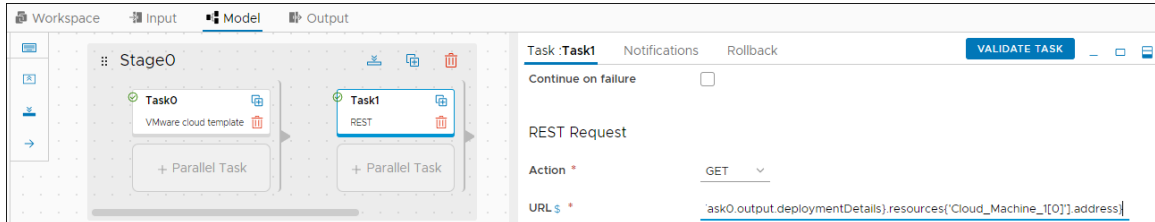
- 2 Tornare al modello della pipeline e immettere l'URL nell'attività REST.

- a Fare clic su **Azioni > Visualizza pipeline**.
- b Fare clic sull'attività REST.

- c Nell'area URL della richiesta REST, immettere **\$**, selezionare **Fase, Attività, output, Dettagli distribuzione** e immettere **resources**.

La funzione di digitazione con completamento automatico è disponibile fino al punto in cui è necessario immettere **resources**.

- d Immettere il resto della risorsa della macchina cloud dei dettagli della distribuzione come: `{ 'Cloud_Machine_1[0]' }.address`



Per la voce della macchina cloud, è necessario utilizzare la notazione con parentesi quadre come mostrato.

Il formato completo dell'URL è: \$

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

- 3 Eseguire la pipeline e verificare che l'attività REST utilizzi la macchina cloud e l'indirizzo IP dell'output dell'attività del modello cloud come URL da testare.

Risultati

Congratulazioni! Nei dettagli della distribuzione e nell'output JSON di un'attività del modello cloud, sono stati trovati il nome e l'indirizzo IP della macchina cloud che sono stati utilizzati per eseguire il binding dell'output dell'attività del modello cloud all'input dell'URL dell'attività REST nella pipeline.

Operazioni successive

Continuare a esplorare utilizzando le variabili di binding dalle risorse nell'attività del modello cloud alle altre attività nella pipeline.

Come utilizzare una REST API per integrare Code Stream con altre applicazioni

Code Stream fornisce un plug-in REST, che consente di integrare Code Stream con altre applicazioni che utilizzano una REST API, per consentire lo sviluppo e la distribuzione continuativi di applicazioni software che devono interagire tra loro. Il plug-in REST richiama un'API, che invia e riceve informazioni tra Code Stream e un'altra applicazione.

Con il plug-in REST, è possibile:

- Integrare in una pipeline di Code Stream sistemi esterni basati sulla REST API.
- Integrare una pipeline di Code Stream come parte del flusso di sistemi esterni.

Il plug-in REST funziona con qualsiasi REST API e supporta i metodi GET, POST, PUT, PATCH e DELETE per inviare o ricevere informazioni tra Code Stream e altre applicazioni.

Tabella 5-7. Preparazione di una pipeline per la comunicazione tramite la REST API

Cosa fare	Cosa accade di conseguenza
Aggiungere un'attività REST alla pipeline.	L'attività REST comunica le informazioni tra le applicazioni e può fornire informazioni sullo stato per un'attività successiva nella fase della pipeline.
Nell'attività REST selezionare l'azione REST e includere l'URL.	L'attività della pipeline richiama l'URL quando la pipeline viene eseguita. Per le azioni POST, PUT e PATCH, è necessario includere un payload. Nel payload è possibile associare le proprietà di pipeline e attività quando la pipeline viene eseguita.
Si consideri questo esempio.	<p>Esempio di utilizzo del plug-in REST:</p> <p>È possibile aggiungere un'attività REST per creare un tag su un commit Git per una build e fare in modo che l'attività invii una richiesta per ottenere l'ID di archiviazione dal repository. L'attività può inviare un payload al repository e creare un tag per la build e il repository può restituire la risposta con il tag.</p>

In modo analogo all'utilizzo del plug-in REST per richiamare un'API, è possibile includere un'attività di polling nella pipeline per richiamare una REST API ed eseguirne il polling finché non viene completata e finché l'attività della pipeline non soddisfa i criteri di uscita.

È inoltre possibile utilizzare le REST API per importare ed esportare una pipeline, nonché utilizzare gli script di esempio per eseguire una pipeline.

Questa procedura consente di ottenere un URL semplice.

Procedura

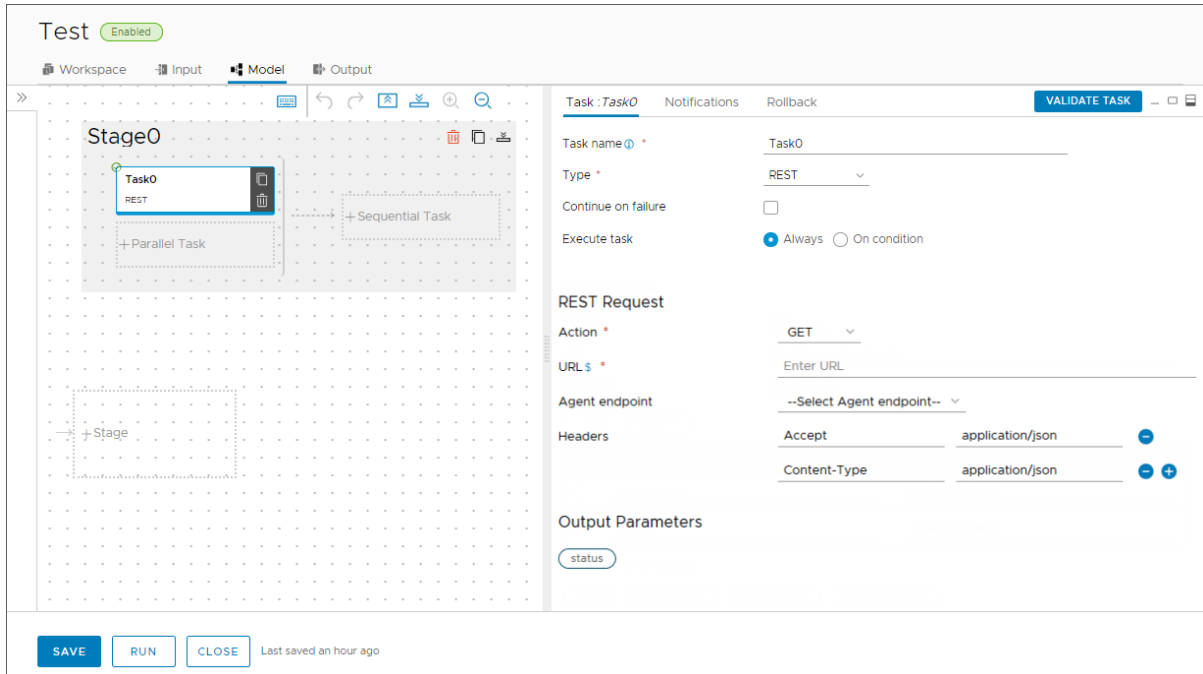
- 1 Per creare una pipeline, fare clic su **Pipeline > Nuova pipeline > Tela vuota**.
- 2 Nella fase della pipeline fare clic su **+ Attività sequenziale**.
- 3 Nel riquadro delle attività aggiungere l'attività REST:
 - a Immettere un nome per l'attività.
 - b Nel menu a discesa Tipo selezionare **REST**.
 - c Nell'area della richiesta REST selezionare **GET**.

Per fare in modo che l'attività REST richieda dati di un'altra applicazione, selezionare il metodo GET. Per inviare dati a un'altra applicazione, selezionare il metodo POST.

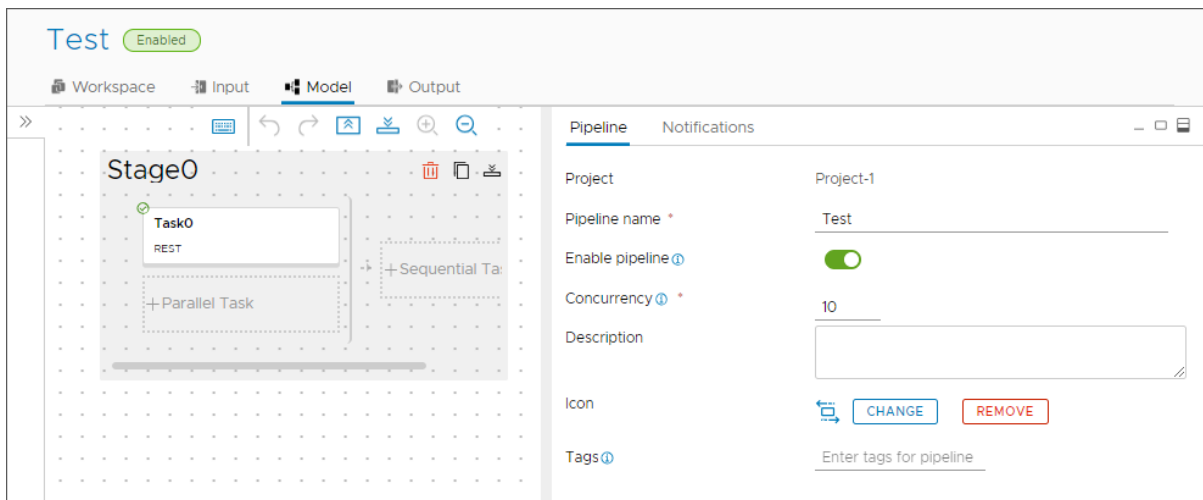
- d Immettere l'URL che identifica l'endpoint della REST API. Ad esempio `https://www.google.com`.

Per consentire a un'attività REST di importare dati da un'altra applicazione, è possibile includere la variabile payload. Ad esempio, per un'azione di importazione, è possibile immettere `${Stage0.export.responseBody}`. Se le dimensioni dei dati della risposta superano i 5 MB, l'attività REST potrebbe non riuscire.

- e Per fornire l'autorizzazione per l'attività, fare clic su **Aggiungi intestazioni** e immettere una chiave di intestazione e un valore.



- 4 Per salvare la pipeline, fare clic su **Salva**.
- 5 Nella scheda Pipeline fare clic su **Attiva pipeline**.



- 6 Fare clic su **Salva**, quindi su **Chiudi**.

- 7 Fare clic su **Esegui**.
- 8 Per osservare l'esecuzione della pipeline, fare clic su **Esecuzioni**.

The screenshot displays the 'Executions' page 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. The entry is named 'Test#1' and has a status of 'RUNNING'. It includes a progress bar for 'Stages' and an 'ACTIONS' dropdown menu. The execution was performed 'By system-user on 11/26/2018 3:11 PM'. The input and output are both listed as 'n/a'.

Test#1	RUNNING	Stages: <div><div></div></div>	ACTIONS ▾
	By system-user on 11/26/2018 3:11 PM	☆ Input : n/a ☆ Output : n/a	

9 Per verificare che l'attività REST restituisca le informazioni previste, esaminare l'esecuzione della pipeline e i risultati dell'attività.

- Al termine della pipeline, per verificare che l'altra applicazione abbia restituito i dati richiesti, fare clic sul collegamento all'esecuzione della pipeline.
- Fare clic sull'attività REST nella pipeline.
- Nell'esecuzione della pipeline fare clic sull'attività, osservare i dettagli e verificare che l'attività REST restituisca i risultati previsti.

I dettagli dell'attività mostrano il codice di risposta, il corpo, le chiavi di intestazione e i valori.

[< BACK](#)

Test #2 COMPLETED 0 ACTIONS

Stage0
Task0

Task name Task0 [VIEW OUTPUT JSON](#)

Type REST

Status COMPLETED Execution Completed.

Duration 1s (11/26/2018 3:45 PM - 11/26/2018 3:45 PM)

Continue on failure ☐

Execute task ☒ Always ☐ On condition

Response

Code 200

Body

```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="aHwW/ydugkGr9CHU6QQGzg==">(function(){window.google={kEI:'cnf8W6KpJIEVkwXx-aLoDA',kEXPI:'0,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,6056,636,2239,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284,2,579,727,612,1820,58,2,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483,698,59,318,273,108,167,323,744,101,1119,38,363,557,438,135,145,155,497,2,718,383,978,487,47,1080,901,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,247,109,1049,14,758,7,127,179,9,21,261,1413,5977597,12,1861,681,134,43,5997424,90,2800095,4,1572,549,332,445,1,2,80,1,900,583,6,307,1,8,1,2,2132,1,1,1,1,414,1,748,141,297,169,301,24,2,8,96,50,2,47,22307501',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&&(a.getAttribute||!b).getAttribute("eid")){a=p
```

Headers

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 Per visualizzare l'output JSON, fare clic su **VISUALIZZA OUTPUT JSON**.

```

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
          =RTUkVjVhyg9KvAZR1S8yCCSEw8WosYfn9MwDFQ1N5fNd5DvXUM5B3J8PyKMX1Z_zRNp3usxttMpd7YiqRUOSfMkTC7cTERbd
          UmOnj3cTppHe3PHIXJPGHnTSZEweb3cxtjVIhVolS85ezVXaTSRYFcG0B_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={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
          ,2,579,727,612,1820,58,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
          ,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

```

Risultati

Congratulazioni! È stata configurata un'attività REST che ha richiamato una REST API e ha inviato informazioni tra Code Stream e un'altra applicazione utilizzando il plug-in REST.

Operazioni successive

Continuare a utilizzare le attività REST nelle pipeline per eseguire comandi e integrare Code Stream con altre applicazioni per poter sviluppare e consegnare le applicazioni software. È consigliabile utilizzare le attività di polling che eseguono il polling dell'API fino al completamento e fino a quando l'attività della pipeline non soddisfa i criteri di uscita.

Come sfruttare la pipeline come codice in Code Stream

Come amministratore o sviluppatore DevOps, è possibile che si desideri creare una pipeline in Code Stream utilizzando il codice YAML anziché l'interfaccia utente. Quando si creano pipeline come codice, è possibile utilizzare un editor qualsiasi e inserire commenti nel codice della pipeline.

Nel codice della pipeline, è possibile fare riferimento alle configurazioni esterne, ad esempio le variabili di ambiente e le credenziali di sicurezza. Quando si aggiornano le variabili utilizzate nel codice della pipeline, è possibile aggiornarle senza dover aggiornare il codice della pipeline.

È possibile utilizzare il codice YAML della pipeline come modello per clonare e creare altre pipeline, nonché condividere i modelli con altri utenti.

È possibile archiviare i modelli di codice della pipeline in un repository di controllo dell'origine, che ne esegue il controllo della versione e tiene traccia degli aggiornamenti. Utilizzando un sistema di controllo dell'origine, è possibile eseguire facilmente il backup del codice della pipeline e ripristinarlo, se necessario.

Prerequisiti

- Verificare di disporre di un editor di codice.
- Se si intende archiviare il codice della pipeline in un repository di controllo dell'origine, verificare che sia possibile accedere a un'istanza funzionante.

Procedura

- 1 Nell'editor di codice, creare un file.
- 2 Copiare e incollare il codice della pipeline di esempio e aggiornarlo in base alle esigenze specifiche della pipeline.
- 3 Per includere un endpoint nel codice della pipeline, copiare e incollare il codice dell'endpoint di esempio e aggiornarlo in modo che rifletta l'endpoint.

Quando si utilizza un endpoint API Kubernetes nell'area di lavoro della pipeline, Code Stream crea le risorse Kubernetes necessarie, come ConfigMap, Secret e Pod per eseguire l'attività di integrazione continua (CI) o l'attività personalizzata. Code Stream comunica con il contenitore tramite la NodePort.

L'area di lavoro della pipeline di Code Stream supporta Docker e Kubernetes per le attività di integrazione continua e le attività personalizzate.

Per ulteriori informazioni sulla configurazione dell'area di lavoro, vedere [Configurazione dell'area di lavoro della pipeline](#).

- 4 Salvare il codice.
- 5 Per archiviare e controllare la versione del codice della pipeline, controllare il codice nel repository del controllo dell'origine.
- 6 Quando si crea una pipeline di integrazione e consegna continue, è necessario importare il file YAML Kubernetes.

Per importare il file YAML Kubernetes, selezionarlo nell'area Consegna continua del modello di pipeline smart e fare clic su **Elabora**. In alternativa, utilizzare l'API.

Risultati

Utilizzando gli esempi di codice, è stato creato il codice YAML che rappresenta la pipeline e gli endpoint.

Esempio: Codice YAML di esempio per una pipeline e gli endpoint

In questo esempio di codice YAML sono incluse le sezioni che rappresentano l'area di lavoro per la build nativa, le fasi, le attività e le notifiche di Code Stream e molto altro in una pipeline.

Per esempi di codice per i plug-in supportati, vedere [Capitolo 6 Connessione di Code Stream agli endpoint](#)

```

---
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}
---
```

Questo codice YAML rappresenta un endpoint Jenkins di esempio.

```

---
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
---
```

Questo codice YAML rappresenta un endpoint Kubernetes di esempio.

```

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

Operazioni successive

Eseguire la pipeline e apportare le modifiche necessarie. Vedere [Come eseguire una pipeline e visualizzare i risultati](#).

Connessione di Code Stream agli endpoint

6

Code Stream si integra con gli strumenti di sviluppo tramite i plug-in. I plug-in supportati includono Jenkins, Bamboo, vRealize Operations, Bugzilla, Team Foundation Server, Git e molti altri.

È inoltre possibile sviluppare plug-in personalizzati che integrano Code Stream con altre applicazioni di sviluppo.

Per integrare Code Stream con Jira, non è necessario un plug-in esterno, perché Code Stream include la funzionalità di creazione dei ticket Jira come tipo di notifica. Per creare ticket Jira sullo stato della pipeline, è necessario aggiungere un endpoint Jira.

Questo capitolo include i seguenti argomenti:

- [Che cosa sono gli endpoint in Code Stream](#)
- [Come integrare Code Stream con Jenkins](#)
- [Come integrare Code Stream con Git](#)
- [Come integrare Code Stream con Gerrit](#)
- [Come integrare Code Stream con vRealize Orchestrator](#)

Che cosa sono gli endpoint in Code Stream

Un endpoint è un'istanza di un'applicazione DevOps che si connette a Code Stream e fornisce i dati per l'esecuzione delle pipeline, ad esempio un'origine dati, un repository o un sistema di notifica.

Il modo in cui si possono utilizzare gli endpoint dipende dal ruolo di cui si dispone in Code Stream.

- Gli amministratori e gli sviluppatori possono creare, aggiornare, eliminare e visualizzare gli endpoint.
- Gli amministratori possono contrassegnare un endpoint come limitato ed eseguire le pipeline che utilizzano questo tipo di endpoint.
- Gli utenti che dispongono del ruolo di visualizzatore possono visualizzare gli endpoint, ma non possono crearli, aggiornarli o eliminarli.

Per ulteriori informazioni, vedere [Come gestire accesso utente e approvazioni in Code Stream](#).

Per connettere Code Stream a un endpoint, eseguire i passaggi seguenti.

- 1 Aggiungere un'attività nella pipeline
- 2 Configurare l'attività in modo che comunichi con l'endpoint.
- 3 Verificare che Code Stream sia in grado di connettersi all'endpoint facendo clic su **Convalida**.
- 4 Quindi, quando si esegue la pipeline, l'attività si connette all'endpoint per eseguire l'attività.

Per informazioni sui tipi di attività che utilizzano questi endpoint, vedere [Tipi di attività disponibili in Code Stream](#).

Tabella 6-1. Endpoint supportati da Code Stream

Endpoint	Che cosa offre	Versioni supportate	Requisiti
Bamboo	Crea piani di creazione.	6.9.*	
Docker	Le creazioni native possono utilizzare gli host Docker per la distribuzione.		Quando una pipeline include un'immagine da Docker Hub, è necessario verificare che l'immagine disponga di <code>cURL</code> o <code>wget</code> incorporato prima di eseguire la pipeline. Quando la pipeline viene eseguita, Code Stream scarica un file binario che utilizza <code>cURL</code> o <code>wget</code> per eseguire i comandi.
Registro Docker	Registra le immagini del contenitore in modo che un host della build Docker possa estrarle.	2.7.1	
Gerrit	Si connette a un server Gerrit per le revisioni e il trigger	2.14.*	
Git	Attiva le pipeline quando gli sviluppatori aggiornano il codice e lo archiviano nel repository.	Git Hub Enterprise 2.1.8 Git Lab Enterprise 11.9.12-ee	
Jenkins	Crea artefatti del codice.	1.6.* e 2.*	
Jira	Crea un ticket JIRA quando un'attività della pipeline non riesce.	8.3.*	

Tabella 6-1. Endpoint supportati da Code Stream (continua)

Endpoint	Che cosa offre	Versioni supportate	Requisiti
Kubernetes	Automatizza i passaggi che distribuiscono, scalano e gestiscono le applicazioni incluse in contenitori.	Tutte le versioni sono supportate per Cloud Assembly 8.4 e versioni successive 1.18 per Cloud Assembly 8.3 e versioni precedenti	Quando si utilizza un endpoint API Kubernetes nell'area di lavoro della pipeline, Code Stream crea le risorse Kubernetes necessarie, come ConfigMap, Secret e Pod per eseguire l'attività di integrazione continua (CI) o l'attività personalizzata. Code Stream comunica con il contenitore tramite la NodePort. Per ulteriori informazioni sulla configurazione dell'area di lavoro, vedere Configurazione dell'area di lavoro della pipeline .
PowerShell	Crea attività che eseguono gli script PowerShell nelle macchine Windows o Linux.	4 e 5	
SSH	Crea attività che eseguono gli script SSH nelle macchine Windows o Linux.	7.0	
TFS, Team Foundation Server	Gestisce il codice sorgente, le creazioni automatiche, i test e le attività correlate.	2015 e 2017	
vRealize Orchestrator	Organizza e automatizza i workflow nel processo di creazione.	7.* e 8.*	

Esempio di codice YAML per un endpoint GitHub

Questo esempio di codice YAML definisce un endpoint GitHub a cui è possibile fare riferimento in un'attività 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
---
```


Come integrare Code Stream con Jenkins

Code Stream fornisce un plug-in Jenkins che attiva i processi Jenkins che creano e testano il codice sorgente. Il plug-in Jenkins esegue test case e può utilizzare script personalizzati.

Per eseguire un processo Jenkins nella pipeline, utilizzare un server Jenkins e aggiungere l'endpoint Jenkins in Code Stream. Quindi, creare una pipeline e aggiungervi un'attività Jenkins.

Quando si utilizzano l'attività Jenkins e un endpoint Jenkins in Code Stream, è possibile creare una pipeline che supporti processi con più rami in Jenkins. Il processo con più rami include singoli processi in ogni ramo di un repository Git. Quando si creano pipeline in Code Stream che supportano processi con più rami:

- L'attività Jenkins può eseguire processi Jenkins che si trovano in più cartelle nel server Jenkins.
- È possibile sostituire il percorso della cartella nella configurazione dell'attività Jenkins in modo che utilizzi un percorso di cartella diverso, che sostituisce il percorso predefinito definito nell'endpoint Jenkins in Code Stream.
- Le pipeline con più rami in Code Stream rilevano i file di processo Jenkins di tipo `.groovy` in un repository Git o in un repository GitHub e iniziano a creare processi per ogni ramo analizzato nel repository.
- È possibile sostituire il percorso predefinito definito nell'endpoint Jenkins con un percorso specificato nella configurazione dell'attività Jenkins ed eseguire un processo e una pipeline associati a tutti i rami di un processo Jenkins principale.

Prerequisiti

- Configurare un server Jenkins che esegua la versione 1.561 o successiva.
- Verificare di essere un membro di un progetto in Code Stream. Se non si è membri, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).
- Verificare che nel server Jenkins sia presente un processo in modo che l'attività della pipeline possa eseguirlo.

Procedura

- 1 Aggiungere e convalidare un endpoint Jenkins.
 - a Fare clic su **Endpoint > Nuovo endpoint**.
 - b Selezionare un progetto e come tipo di endpoint, selezionare **Jenkins**. Quindi, immettere un nome e una descrizione.
 - c Se l'endpoint è un componente business critical nell'infrastruttura, abilitare **Contrassegna come limitato**.
 - d Immettere l'URL del server Jenkins.

- e Immettere il nome utente e la password per accedere al server Jenkins. Quindi, immettere le informazioni rimanenti.

Tabella 6-2. Informazioni rimanenti per l'endpoint Jenkins

Voce endpoint	Descrizione
Percorso cartella	<p>Percorso della cartella che raggruppa i processi. Jenkins può eseguire tutti i processi della cartella. È possibile creare sottocartelle. Ad esempio:</p> <ul style="list-style-type: none"> ■ <code>folder_1</code> può includere <code>job_1</code> ■ <code>folder_1</code> può includere <code>folder_2</code>, che può includere <code>job_2</code> <p>Quando si crea un endpoint per <code>folder_1</code>, il percorso della cartella è <code>job/folder_1</code> e nell'endpoint è indicato solo <code>job_1</code>.</p> <p>Per ottenere l'elenco dei processi nella cartella secondaria denominata <code>folder_2</code>, è necessario creare un altro endpoint che utilizzi il percorso della cartella come <code>/job/folder_1/job/folder_2/</code>.</p>
Percorso della cartella per i processi Jenkins con più rami	<p>Per supportare i processi Jenkins con più rami, nell'attività Jenkins immettere il percorso completo che include l'URL del server Jenkins e il percorso del processo completo. Quando si include il percorso di una cartella nell'attività Jenkins, tale percorso sostituisce il percorso visualizzato nell'endpoint Jenkins. Con il percorso della cartella personalizzata nell'attività Jenkins, Code Stream visualizza solo i processi presenti in tale cartella.</p> <ul style="list-style-type: none"> ■ Ad esempio: <code>https://server.yourcompany.com/job/project</code> ■ Se la pipeline deve anche attivare il processo Jenkins principale, utilizzare: <code>https://server.yourcompany.com/job/project/job/main</code>
URL	URL host del server Jenkins. Immettere l'URL nel formato <code>protocol://host:port</code> . Ad esempio: <code>http://192.10.121.13:8080</code>
Intervallo di polling	Durata dell'intervallo in cui Code Stream esegue il polling del server Jenkins per gli aggiornamenti.

Tabella 6-2. Informazioni rimanenti per l'endpoint Jenkins (continua)

Voce endpoint	Descrizione
Numero tentativi richiesta	Numero di tentativi di richiesta di creazione pianificata per il server Jenkins.
Tempo di attesa per riprova	Numero di secondi di attesa prima di riprovare la richiesta di creazione per il server Jenkins.

- f Fare clic su **Convalida** e verificare che l'endpoint si connetta a Code Stream. Se non si connette, correggere eventuali errori, quindi fare clic su **Salva**.

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 Per creare il codice, creare una pipeline e aggiungere un'attività che utilizzi l'endpoint Jenkins.
 - a Fare clic su **Pipeline > Nuova pipeline > Tela vuota**.
 - b Fare clic sulla fase predefinita.
 - c Nell'area Attività immettere un nome per l'attività.

- d Selezionare il tipo di attività come **Jenkins**.
- e Selezionare l'endpoint Jenkins creato.
- f Dal menu a discesa selezionare un processo dal server Jenkins che verrà eseguito dalla pipeline.
- g Immettere i parametri per il processo.
- h Immettere il token di autenticazione per il processo Jenkins.

The screenshot displays the 'Build and Deploy' task configuration in the vRealize Automation Code Stream interface. The task is titled 'Build and Deploy' and is currently 'Enabled'. The task is configured with the following details:

- 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

The task is part of a stage named 'Stage0'. The stage contains two tasks: 'Build' and 'Test', both of which are Jenkins tasks. The 'Build' task is selected, and its configuration is shown on the right. The 'Test' task is also a Jenkins task. Below the stage, there is a '+ Stage' button to add a new stage. At the bottom of the interface, there are buttons for 'SAVE', 'RUN', and 'CLOSE', along with a status message 'Last saved a month ago' and a chat icon.

3 Abilitare ed eseguire la pipeline e visualizzare l'esecuzione della pipeline.

The screenshot displays the 'Build and Deploy #28' pipeline execution page. At the top, a green 'COMPLETED' status is shown next to a '0' icon and an 'ACTIONS' dropdown. Below this, a 'Stage0' progress bar indicates the completion of four tasks: 'Build', 'Test', 'Approval for Deployment', and 'Deployment', followed by 'Wait for application to start'. The 'Build' task is highlighted with a blue box.

Below the progress bar, the 'Task name' is 'Build', with a 'VIEW OUTPUT JSON' link. The 'Type' is 'Jenkins'. The 'Status' is 'COMPLETED' with the message 'Execution Completed.'. The 'Duration' is '11s (08/06/2018 12:27 AM - 08/06/2018 12:27 AM)'. The 'Continue On Failure' checkbox is unchecked. The 'Execute Task' options are 'Always' (selected) and 'On Condition'.

The 'Jenkins Job' section lists the following details:

- Endpoint: aa
- Job Name: add_numbers
- Job ID: 1428
- Job URL: http://.../job/add_numbers/1428/

The 'Job Result' section contains a table with the following data:

Key	Value
junitResponse.failCount	0
junitResponse.skipCount	0
junitResponse.totalCount	0
junitResponse.successCount	0
jacocoResponse.lineCoverage	0
jacocoResponse.classCoverage	0

4 Esaminare lo stato e i dettagli di esecuzione nel dashboard della pipeline.

È possibile identificare eventuali errori e il motivo per cui si sono verificati. È inoltre possibile visualizzare le tendenze relative alla durata, al completamento e agli errori di esecuzione della 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 their corresponding status bars (green for completed, red for failed, blue for running, yellow for waiting). A legend at the bottom of this section defines the colors: green for Completed, red for Failed, blue for Running, and yellow for Waiting.

Below the 'Recent Executions' section is the 'Execution Details' section, which contains a table with the following columns: Execution#, Status, Status Message, Duration, and Updated On.

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 of the 'Execution Details' section, there are navigation icons: a monitor icon, a play icon, a stop icon, and a refresh icon.

Risultati

Congratulazioni! È stata effettuata l'integrazione di Code Stream con Jenkins mediante l'aggiunta di un endpoint, la creazione di una pipeline e la configurazione di un'attività Jenkins che crea il codice.

Esempio: Esempio YAML per un'attività di creazione Jenkins

Per il tipo di attività di creazione Jenkins utilizzato in questo esempio, il codice YAML è simile al seguente, con le notifiche attivate:

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

Operazioni successive

Esaminare le altre sezioni per ulteriori informazioni. Vedere [Capitolo 6 Connessione di Code Stream agli endpoint](#).

Come integrare Code Stream con Git

Code Stream offre un modo per attivare una pipeline se si verifica una modifica del codice nel repository GitHub, GitLab o Bitbucket. Il trigger Git utilizza un endpoint Git nel ramo del repository che si desidera monitorare. Code Stream si connette all'endpoint Git tramite un webhook.

Per definire un endpoint Git in Code Stream, selezionare un progetto e immettere il ramo del repository Git in cui si trova l'endpoint. Il progetto raggruppa la pipeline con l'endpoint e altri oggetti correlati. Quando si sceglie il progetto nella definizione del webhook, si selezionano l'endpoint e la pipeline da attivare.

Nota Se si definisce un webhook con l'endpoint e successivamente si modifica l'endpoint, non è possibile modificare i dettagli dell'endpoint nel webhook. Per modificare i dettagli dell'endpoint, è necessario eliminare e ridefinire il webhook con l'endpoint. Vedere [Come utilizzare il trigger Git in Code Stream per eseguire una pipeline](#).

È possibile creare più webhook per rami diversi utilizzando lo stesso endpoint Git e specificando valori diversi per il nome del ramo nella pagina di configurazione del webhook. Per creare un altro webhook per un altro ramo nello stesso repository Git, non è necessario clonare l'endpoint Git più volte per più rami. È infatti sufficiente specificare il nome del ramo nel webhook in modo da poter riutilizzare l'endpoint Git. Se il ramo nel webhook Git è lo stesso ramo dell'endpoint, non è necessario specificare il nome del ramo nella pagina del webhook Git.

Prerequisiti

- Verificare che sia possibile accedere al repository GitHub, GitLab o Bitbucket a cui si intende connettersi.
- Verificare di essere un membro di un progetto in Code Stream. In caso contrario, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).

Procedura

- 1 Definire un endpoint Git.
 - a Fare clic su **Endpoint > Nuovo endpoint**.
 - b Selezionare un progetto e, per il tipo di endpoint, selezionare **Git**. Immettere un nome e una descrizione.

- c Se l'endpoint è un componente business critical nell'infrastruttura, abilitare **Contrassegna come limitato**.

Quando si utilizza un endpoint limitato in una pipeline, un amministratore può eseguire la pipeline e deve approvare l'esecuzione della pipeline. Se un endpoint o una variabile sono contrassegnati come limitati e un utente non amministratore attiva la pipeline, la pipeline si interrompe in corrispondenza di tale attività e attende che un amministratore la riprenda.

Un amministratore del progetto può avviare una pipeline che includa endpoint o variabili con limitazioni se queste risorse si trovano nel progetto in cui l'utente è un amministratore del progetto.

Quando un utente che non è un amministratore tenta di eseguire una pipeline che include una risorsa limitata, la pipeline si arresta in corrispondenza dell'attività che utilizza la risorsa limitata. Un amministratore deve quindi riprendere la pipeline.

Per ulteriori informazioni sulle risorse limitate e i ruoli personalizzati che includono l'autorizzazione denominata **Gestisci pipeline limitate**, vedere:

- [Come gestire accesso utente e approvazioni in Code Stream](#)
- [Capitolo 2 Impostazione di Code Stream per modellare il processo di rilascio](#)

- d Selezionare uno dei tipi di server Git supportati.
- e Immettere l'URL del repository con il gateway API per il server nel percorso. Ad esempio:

Per GitHub, immettere : `https://api.github.com/vmware-example/repo-example`

Per BitBucket, immettere: `https://api.bitbucket.org/{user}/{repo name}`

o `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`

- f Immettere il ramo nel repository in cui si trova l'endpoint.
- g Selezionare il tipo di autenticazione e immettere il nome utente per GitHub, GitLab o BitBucket. Immettere quindi il token privato associato al nome utente.
- Password. Per creare un webhook in un secondo momento, è necessario immettere il token privato per la password. I webhook per Git non supportano gli endpoint creati utilizzando l'autenticazione di base.

Utilizzare le variabili segrete per nascondere e crittografare le informazioni riservate. Utilizzare una variabile limitata per le stringhe, le password e gli URL che devono essere nascosti e crittografati, nonché per limitare l'uso nelle esecuzioni. Ad esempio, utilizzare una variabile segreta per una password o un URL. È possibile utilizzare variabili segrete e limitate in qualsiasi tipo di attività nella pipeline.

- Token privato. Questo token è specifico per Git e fornisce l'accesso a un'azione specifica. Vedere https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html. È inoltre possibile creare una variabile per il token privato.

- 2 Fare clic su **Convalida** e verificare che l'endpoint si connetta a Code Stream.

Se non si connette, correggere gli eventuali errori, quindi fare clic su **Crea**.

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

Operazioni successive

Per ulteriori informazioni, rivedere le altre sezioni. Vedere [Come utilizzare il trigger Git in Code Stream per eseguire una pipeline](#).

Come integrare Code Stream con Gerrit

Code Stream consente di attivare una pipeline se si verifica una revisione del codice nel progetto Gerrit. La definizione del trigger per Gerrit include il progetto Gerrit e le pipeline da eseguire per tipi di evento diversi.

Il trigger per Gerrit utilizza un listener Gerrit nel server Gerrit da monitorare. Per definire un endpoint Gerrit in Code Stream, è necessario selezionare un progetto e immettere l'URL del server Gerrit. È quindi possibile specificare l'endpoint quando si crea un listener Gerrit in tale server.

Se si utilizza un server Gerrit come endpoint di Code Stream in un'istanza di vRealize Automation in cui è abilitato FIPS, è necessario verificare che il file di configurazione Gerrit includa le chiavi di autenticazione dei messaggi corrette. Se il file di configurazione del server Gerrit non include le chiavi di autenticazione del messaggio corrette, il server non può essere avviato correttamente e viene visualizzato il messaggio: `PrivateKey/PassPhrase is incorrect`

Prerequisiti

- Verificare che sia possibile accedere al server Gerrit a cui si intende connettersi.
- Verificare di essere un membro di un progetto in Code Stream. Se non si è membri, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).

Procedura

1 Definire un endpoint Gerrit.

- a Fare clic su **Configura > Endpoint** e quindi su **Nuovo endpoint**.
- b Selezionare un progetto e, per il tipo di endpoint, selezionare **Gerrit**. Quindi, immettere un nome e una descrizione.
- c Se l'endpoint è un componente business critical nell'infrastruttura, abilitare **Contrassegna come limitato**.
- d Immettere l'URL del server Gerrit.

Per utilizzare la porta predefinita, è possibile specificare un numero di porta con l'URL o lasciare il valore vuoto.

- e Immettere un nome utente e una password per il server Gerrit.

Se la password deve essere crittografata, fare clic su **Crea variabile** e selezionare il tipo:

- Segreta. La password viene risolta quando un utente con un ruolo qualsiasi esegue la pipeline.
- Limitata. La password viene risolta quando un utente con ruolo di amministratore esegue la pipeline.

Come valore, immettere la password che deve essere sicura, ad esempio la password di un server Jenkins.

- f Per la chiave privata, immettere la chiave SSH utilizzata per accedere in modo sicuro al server Gerrit.

Questa chiave è la chiave privata RSA che si trova nella directory `.ssh`.

- g (Facoltativo) Se alla chiave privata è associata una frase d'accesso, immetterla.

Per crittografare la passphrase, fare clic su **Crea variabile** e selezionare il tipo:

- Segreta. La password viene risolta quando un utente con un ruolo qualsiasi esegue la pipeline.
- Limitata. La password viene risolta quando un utente con ruolo di amministratore esegue la pipeline.

Come valore, immettere la passphrase che deve essere sicura, ad esempio la passphrase per un server SSH.

- 2 Fare clic su **Convalida** e verificare che l'endpoint Gerrit in Code Stream si connetta al server Gerrit.

Se non si connette, correggere gli eventuali errori, quindi fare di nuovo clic su **Convalida**.

New endpoint

Project	test		
Type	Gerrit		
Name *	Gerrit-Demo-Endpoint		
Description	<div></div>		
Mark restricted	<input type="checkbox"/> non-restricted		
URL *	http://example-gerrit.mycompany.com:8080		
Username *	CS_user		
Password *	<div> <div></div> <div>CREATE VARIABLE</div> </div>		
Private Key *	<div> <div>-----BEGIN RSA PRIVATE KEY-----</div> <div>Proc-Type: 4,ENCRYPTED</div> <div>DEK-Info: AES-128-CBC,F00CE0B6526AF67DC77ADCD0962DBF92</div> <div></div> </div>		
Pass Phrase ⓘ	<div> <div></div> <div>CREATE VARIABLE</div> </div>		
<div> <div>CREATE</div> <div>VALIDATE</div> <div>CANCEL</div> </div>			

- 3 Fare clic su **Crea**.

- 4 Verificare che nell'ambiente vRealize Automation sia abilitato FIPS oppure fare in modo che il processo Jenkins crei l'ambiente con FIPS abilitato utilizzando l'URL di Jenkins.
 - a Per eseguire il comando dalla riga di comando, connettersi all'appliance vRealize Automation 8.x tramite SSH e accedere come utente root. Ad esempio, connettersi all'URL del nome di dominio completo, ad esempio `https://cava-1-234-567.yourcompanyFQDN.com` sulla porta 22, 5480 o 443.
 - b Per verificare FIPS in vRealize Automation, eseguire il comando `vracli security fips`.
 - c Assicurarsi che il comando restituisca `FIPS mode: strict`.
- 5 Se il server Gerrit è un endpoint in un'istanza di vRealize Automation in cui è abilitato FIPS, assicurarsi che il file di configurazione Gerrit includa le chiavi MAC (Message Authentication Code) corrette.
 - a Aprire Gerrit e creare una coppia di chiavi SSH.
 - b Individuare il file di configurazione del server Gerrit in `'$site_path'/etc/gerrit.config`.
 - c Verificare che il file di configurazione del server Gerrit includa una o più chiavi MAC (Message Authentication Code), eccetto per `hmac-MD5`.

Nota In modalità FIPS, `hmac-MD5` non è un algoritmo MAC supportato. Per assicurarsi che il server Gerrit venga avviato correttamente, il file di configurazione del server Gerrit deve escludere questo algoritmo. Se il server Gerrit non si avvia correttamente, viene visualizzato il messaggio `PrivateKey/PassPhrase is incorrect`

I nomi delle chiavi MAC (Message Authentication Code) supportati che iniziano con un segno più (+) sono abilitati. I nomi delle chiavi MAC che iniziano con un trattino (-) vengono rimossi dall'elenco dei MAC predefiniti. Per impostazione predefinita, i MAC supportati sono disponibili in un Code Stream per il server Gerrit:

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

Operazioni successive

Per ulteriori informazioni, rivedere le altre sezioni. Vedere [Come utilizzare il trigger Gerrit in Code Stream per eseguire una pipeline](#).

Come integrare Code Stream con vRealize Orchestrator

Code Stream può essere integrato con vRealize Orchestrator (vRO) per estenderne la funzionalità mediante l'esecuzione dei workflow di vRO. vRealize Orchestrator include molti workflow predefiniti che possono essere integrati con strumenti di terze parti. Questi workflow consentono di automatizzare e gestire i processi DevOps, automatizzare le operazioni in massa e molto altro.

Ad esempio, è possibile utilizzare un workflow in un'attività vRO nella propria pipeline per abilitare un utente, rimuovere un utente, spostare macchine virtuali, eseguire l'integrazione con i framework di test per testare il codice quando vengono eseguite le pipeline e molto altro. È possibile trovare esempi di codice per i workflow vRealize Orchestrator in code.vmware.com.

Con un workflow vRealize Orchestrator, la pipeline può eseguire un'azione quando crea, testa e distribuisce l'applicazione. È possibile includere workflow predefiniti nella pipeline oppure si possono creare e utilizzare workflow personalizzati. Ciascun workflow include input, attività e output.

Per eseguire un workflow vRO nella pipeline, questo deve essere visualizzato nell'elenco dei workflow disponibili nell'attività vRO che viene inclusa nella pipeline.

Prima che il workflow possa essere visualizzato nell'attività vRO nella pipeline, è necessario che un amministratore esegua i seguenti passaggi in vRealize Orchestrator:

- 1 Applicare il tag `CODESTREAM` al workflow vRO.
- 2 Contrassegnare il workflow vRO come globale.

Prerequisiti

- Verificare che, in qualità di amministratore, sia possibile accedere a un'istanza locale di vRealize Orchestrator. Per ricevere assistenza, consultare l'amministratore e la [documentazione di vRealize Orchestrator](#).
- Verificare di essere un membro di un progetto in Code Stream. In caso contrario, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).
- In Code Stream creare una pipeline e aggiungere una fase.

Procedura

- 1 In qualità di amministratore, preparare un workflow di vRealize Orchestrator per l'esecuzione della pipeline.
 - a In vRealize Orchestrator, trovare il workflow che è necessario utilizzare nella pipeline, ad esempio un workflow per abilitare un utente.
Se è necessario un workflow che non esiste, è possibile crearlo.
 - b Nella barra di ricerca, immettere **Tag workflow** per trovare il workflow denominato Tag workflow.

- c Nella scheda denominata `Tag workflow`, fare clic su **Esegui**. Viene visualizzata l'area di configurazione.
- d Nell'area di testo `Tagged workflow`, immettere il nome del workflow da utilizzare nella pipeline di Code Stream, quindi selezionarlo nell'elenco.
- e Nelle aree di testo `Tag` e `Value`, immettere `CODESTREAM` in lettere maiuscole.
- f Fare clic sulla casella di controllo **Tag globale**.
- g Fare clic su **Esegui** per collegare il tag denominato `CODESTREAM` al workflow che è necessario selezionare nella pipeline di Code Stream.
- h Nel riquadro di spostamento, fare clic su **Workflow** e verificare che il tag denominato `CODESTREAM` sia presente nella scheda del workflow che verrà eseguito dalla pipeline.

Dopo aver effettuato l'accesso a Code Stream e aver aggiunto un'attività di `vRO` alla pipeline, nell'elenco dei workflow viene visualizzato il workflow con tag.

- 2 In Code Stream creare un endpoint per l'istanza di vRealize Orchestrator.
 - a Fare clic su **Endpoint > Nuovo endpoint**.
 - b Selezionare un progetto.
 - c Immettere un nome pertinente.

- d Immettere l'URL dell'endpoint di vRealize Orchestrator.

Utilizzare il seguente formato: **https://vro-appliance.yourdomain.local:8281**

Non utilizzare il seguente formato: **https://vro-appliance.yourdomain.local:8281/vco/api**

L'URL di un'istanza di vRealize Orchestrator integrata nell'appliance vRealize Automation è il nome di dominio completo per l'appliance senza porta. Per esempio: **https://vro-appliance.yourdomain.local/vco**

Per le appliance vRealize Orchestrator esterne a partire da vRealize Automation 8.x, il nome di dominio completo per l'appliance è **https://vro-appliance.yourdomain.local**

Per le appliance vRealize Orchestrator esterne incluse con vRealize Automation 7.x, il nome di dominio completo per l'appliance è **https://vro-appliance.yourdomain.local:8281/vco**

Se si verifica un problema quando si aggiunge l'endpoint, potrebbe essere necessario importare una configurazione YAML con un'impronta digitale del certificato SHA-256 con i due punti rimossi. Ad esempio, **B0:01:A2:72 . . .** diventa **B001A272 . . .**. Il codice YAML di esempio è simile 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 Fare clic su **Accetta certificato** nel caso in cui l'URL immesso necessiti di un certificato.
- f Immettere il nome utente e la password per il server vRealize Orchestrator.

Se si utilizza un utente non locale per l'autenticazione, è necessario omettere la parte del dominio del nome utente. Ad esempio, per eseguire l'autenticazione con **svc_vro@yourdomain.local**, è necessario immettere **svc_vro** nell'area di testo **Nome utente**.

3 Preparare la pipeline per eseguire l'attività vRO.

- a Aggiungere un'attività vRO alla fase della pipeline.
- b Immettere un nome pertinente.
- c Nell'area Proprietà workflow selezionare l'endpoint vRealize Orchestrator.

- d Selezionare il workflow che è stato contrassegnato come CODESTREAM in vRealize Orchestrator.

Se si seleziona un workflow personalizzato creato dall'utente, potrebbe essere necessario immettere i valori dei parametri di input.

- e Per **Esegui attività**, fare clic su **Su condizione**.

The screenshot shows the configuration window for a task named "vRO workflow". The window has tabs for "Task :vRO workflow", "Notifications", and "Rollback". A "VALIDATE TASK" button is in the top right. The configuration fields are as follows:

Field	Value
Task name ⓘ *	vRO workflow
Type *	vRO
Duration	NaNps (-)
Continue on failure	<input type="checkbox"/>
Execute task	<input type="radio"/> Always <input checked="" type="radio"/> On condition
Condition \$	<input type="text"/>
Workflow Properties	
Endpoint	vROEP
Workflow	Test
Greeting	Hello!
Output Parameters	
<input type="button" value="status"/> <input type="button" value="properties"/>	

- f Immettere le condizioni che si applicano quando viene eseguita la pipeline.

Quando eseguire pipeline...	Selezionare le condizioni...
Su condizione	<p>Esegue l'attività della pipeline solo se la condizione definita viene valutata come true. Se la condizione è false, l'attività viene ignorata.</p> <p>L'attività vRO consente di includere un'espressione booleana, che utilizza i seguenti operandi e operatori.</p> <ul style="list-style-type: none"> ■ Variabili di pipeline come <code>\${pipeline.variableName}</code>. Quando si immettono le variabili, utilizzare solo le parentesi graffe. ■ Variabili di output dell'attività, ad esempio <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code>. ■ Variabili di associazione delle pipeline predefinite, ad esempio <code>\${releasePipelineName}</code>. ■ Valori booleani con distinzione tra maiuscole e minuscole, ad esempio <code>true</code>, <code>false</code>, <code>'true'</code>, <code>'false'</code>. ■ Valori interi o decimali senza virgolette. ■ Valori stringa utilizzati con virgolette singole o doppie, ad esempio <code>"test"</code>, <code>'test'</code>. ■ Tipi stringa e numerici di valori, quali <code>==</code> <code>Equals</code> e <code>!=</code> <code>Not Equals</code>. ■ Operatori relazionali quali <code>></code>, <code>>=</code>, <code><</code> e <code><=</code>. ■ Logica booleana, ad esempio <code>&&</code> e <code> </code>. ■ Operatori aritmetici quali <code>+</code>, <code>-</code>, <code>*</code> e <code>/</code>. ■ Espressioni nidificate che utilizzano parentesi tonde. ■ Le stringhe che includono il valore letterale ABCD vengono valutate come false e l'attività viene ignorata. ■ Gli operatori unari non sono supportati. <p>Una condizione di esempio potrebbe essere <code>\${Stage1.task1.output} == "Passed" \${pipeline.variableName} == 39</code></p>
Sempre	Se si seleziona Sempre , la pipeline esegue l'attività senza condizioni.

- g Immettere un messaggio di saluto.

- h Fare clic su **Convalida attività** e correggere gli eventuali errori che si verificano.

- 4 Salvare, abilitare ed eseguire la pipeline.

- 5 Dopo l'esecuzione della pipeline, esaminare i risultati.

- a Fare clic su **Esecuzioni**.

- b Fare clic sulla pipeline.

- c Fare clic sull'attività.

- d Esaminare i risultati, il valore di input e le proprietà.

È possibile identificare l'ID di esecuzione del workflow, chi ha risposto all'attività e quando, nonché tutti i commenti che ha incluso.

Risultati

Congratulazioni! È stato contrassegnato un workflow vRealize Orchestrator per l'utilizzo in Code Stream ed è stata aggiunta un'attività vRO nella pipeline di Code Stream in modo che esegua un workflow che automatizza un'azione nell'ambiente DevOps.

Esempio: Formato output dell'attività vRO

Il formato di output per un'attività vRO è simile a questo esempio.

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

Operazioni successive

Continuare a includere le attività di workflow vRO nelle pipeline in modo da poter automatizzare le attività negli ambienti di sviluppo, test e produzione.

Attivazione di pipeline in Code Stream

7

È possibile fare in modo che Code Stream attivi una pipeline quando si verificano determinati eventi.

Ad esempio:

- Il trigger Docker può eseguire una pipeline quando viene creato o aggiornato un nuovo artefatto.
- Il trigger Git può attivare una pipeline quando gli sviluppatori aggiornano il codice.
- Il trigger per Gerrit può attivare una pipeline quando gli sviluppatori rivedono il codice.

Questo capitolo include i seguenti argomenti:

- [Come utilizzare il trigger Docker in Code Stream per eseguire una pipeline di consegna continua](#)
- [Come utilizzare il trigger Git in Code Stream per eseguire una pipeline](#)
- [Come utilizzare il trigger Gerrit in Code Stream per eseguire una pipeline](#)

Come utilizzare il trigger Docker in Code Stream per eseguire una pipeline di consegna continua

Amministratori e sviluppatori di Code Stream possono utilizzare il trigger Docker in Code Stream. Il trigger Docker esegue una pipeline di consegna continua (CD) standalone ogni volta che viene creato o aggiornato un artefatto di creazione. Il trigger Docker esegue la pipeline CD, che inserisce l'artefatto nuovo o aggiornato come immagine del contenitore in un repository Docker Hub. La pipeline CD può essere eseguita nell'ambito delle creazioni automatiche.

Ad esempio, per distribuire in modo continuativo l'immagine del contenitore aggiornata tramite la pipeline CD, utilizzare il trigger Docker. Quando l'immagine del contenitore viene selezionata nel registro Docker, il webhook in Docker Hub segnala a Code Stream che l'immagine è stata modificata. Questa notifica attiva la pipeline della consegna continua da eseguire con l'immagine del contenitore aggiornata e carica l'immagine nel repository di Docker Hub.

Per utilizzare il trigger Docker, è necessario eseguire diversi passaggi in Code Stream.

Tabella 7-1. Come utilizzare il trigger Docker

Cosa fare...	Ulteriori informazioni su questa azione...
Creare un endpoint del registro Docker.	<p>Per consentire a Code Stream di attivare la pipeline, è necessario disporre di un endpoint del registro Docker. Se l'endpoint non esiste, è possibile selezionare un'opzione che lo crei quando si aggiunge il webhook per il trigger Docker.</p> <p>L'endpoint del registro Docker include l'URL del repository Docker Hub.</p>
Aggiungere parametri di input alla pipeline che immettono automaticamente i parametri Docker al momento dell'esecuzione della pipeline.	<p>È possibile inviare i parametri Docker nella pipeline. I parametri possono includere il nome del proprietario, l'immagine, il nome del repository, lo spazio dei nomi del repository e il tag dell'evento Docker.</p> <p>Nella pipeline CD includere parametri di input che il webhook Docker passa alla pipeline prima che venga attivata.</p>
Creare un webhook Docker.	<p>Quando si crea il webhook Docker in Code Stream, viene creato anche un webhook corrispondente in Docker Hub. Il webhook Docker in Code Stream si connette a Docker Hub tramite l'URL che viene incluso nel webhook.</p> <p>I webhook comunicano tra loro e attivano la pipeline quando viene creato o aggiornato un artefatto in Docker Hub.</p> <p>Se si aggiorna o si elimina il webhook Docker in Code Stream, viene aggiornato o eliminato anche il webhook in Docker Hub.</p>
Aggiungere e configurare un'attività Kubernetes nella pipeline.	Quando si crea o aggiorna un artefatto nel repository Docker Hub, la pipeline viene attivata. Quindi, distribuisce l'artefatto tramite la pipeline all'host Docker nel cluster Kubernetes.
Includere una definizione YAML locale nell'attività.	<p>La definizione YAML applicata all'attività di distribuzione include l'immagine del contenitore Docker. Se è necessario scaricare un'immagine da un repository di proprietà privata, il file YAML deve includere una sezione con il segreto di configurazione di Docker. Vedere la parte CD di Pianificazione di una creazione nativa di integrazione continua e consegna continua in Code Stream prima di utilizzare il modello di pipeline smart</p>

Quando viene creato o aggiornato un artefatto nel repository Docker Hub, il webhook in Docker Hub invia una notifica al webhook in Code Stream, che attiva la pipeline. Si verificano le seguenti azioni:

- 1 Docker Hub invia una richiesta POST all'URL nel webhook.
- 2 Code Stream esegue il trigger Docker.
- 3 Il trigger Docker avvia la pipeline CD.
- 4 La pipeline CD inserisce l'artefatto nel repository Docker Hub.

- 5 Code Stream attiva il webhook Docker, che esegue una pipeline CD che distribuisce l'artefatto nell'host Docker.

In questo esempio vengono creati un endpoint Docker e un webhook Docker in Code Stream, che distribuisce l'applicazione nel cluster Kubernetes di sviluppo. I passaggi includono il codice di esempio per il payload che Docker invia all'URL nel webhook, il codice API che utilizza e il codice di autenticazione con il token sicuro.

Prerequisiti

- Verificare che nella propria istanza di Code Stream esista una pipeline di consegna continua (CD). Verificare inoltre che includa una o più attività Kubernetes che distribuiscono l'applicazione. Vedere [Capitolo 4 Pianificazione di creazione, integrazione e consegna native del codice in Code Stream](#).
- Verificare che sia possibile accedere a un cluster Kubernetes esistente in cui la pipeline CD può distribuire l'applicazione per lo sviluppo.
- Verificare di essere un membro di un progetto in Code Stream. In caso contrario, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).

Procedura

- 1 Creare un endpoint del registro Docker.
 - a Fare clic su **Endpoint**.
 - b Fare clic su **Nuovo endpoint**.
 - c Iniziare a digitare il nome del progetto esistente.
 - d Selezionare il tipo come **Registro Docker**.
 - e Immettere un nome pertinente.
 - f Selezionare il tipo di server come **DockerHub**.
 - g Immettere l'URL del repository Docker Hub.
 - h Immettere il nome e la password che possono accedere al repository.

New endpoint

Project *	<input type="text" value="AWS_PGProj"/>
Type *	<input type="text" value="Docker Registry"/>
Name *	<input type="text" value="dockerhub-endpoint"/>
Description	<input type="text"/>
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 Nella pipeline CD impostare le proprietà di input per l'immissione automatica dei parametri Docker al momento dell'esecuzione della pipeline.

sm-1 Enabled

Workspace **Input** Model Output

Input Parameters ⓘ

Auto inject parameters ☐ Gerrit ☐ Git ☒ Docker ☐ None

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

Starred ⓘ	Name
<input type="checkbox"/>	DOCKER_EVENT_OWNER_NAME
<input type="checkbox"/>	DOCKER_IMAGE
<input type="checkbox"/>	DOCKER_REPO_NAME
<input type="checkbox"/>	DOCKER_REPO_NAMESPACE
<input type="checkbox"/>	DOCKER_TAG

- 3 Creare un webhook Docker.
 - a Fare clic su **Trigger > Docker**.
 - b Fare clic su **Nuovo webhook per Docker**.
 - c Selezionare un progetto.
 - d Immettere un nome pertinente.
 - e Selezionare l'endpoint del registro Docker.

Se l'endpoint ancora non esiste, crearlo facendo clic su **Crea endpoint**.

- f Selezionare la pipeline con i parametri di Docker inviati per il webhook da attivare. Vedere [Passaggio 2](#).

Se la pipeline è stata configurata con parametri di input aggiunti personalizzati, nell'elenco Parametri di input vengono visualizzati i parametri e i valori. È possibile immettere valori per i parametri di input che verranno passati alla pipeline con l'evento trigger. In alternativa, è possibile lasciare vuoti i valori oppure utilizzare i valori predefiniti, se specificati.

Per ulteriori informazioni sui parametri nella scheda di input, vedere [Pianificazione di una creazione nativa di integrazione e consegna continue in Code Stream prima dell'aggiunta manuale di attività](#).

g Immettere il token dell'API.

Il token dell'API CSP autentica l'utente per le connessioni API esterne con Code Stream.
Per ottenere il token API:

- 1 Fare clic su **Genera Token**.
- 2 Immettere l'indirizzo e-mail associato al nome utente e alla password, quindi fare clic su **Genera**.

Il token generato è valido per sei mesi. È anche noto come token di aggiornamento.

- Per mantenere il token come una variabile per un utilizzo futuro, fare clic su **Crea variabile**, assegnare un nome alla variabile e fare clic su **Salva**.
- Per mantenere il token come valore di testo per un utilizzo futuro, fare clic su **Copia** e incollare il token in un file di testo da salvare in locale.

È possibile scegliere di creare una variabile e archiviare il token in un file di testo per un utilizzo futuro.

- 3 Fare clic su **Chiudi**.

h Immettere l'immagine della build.

- i Immettere un tag.

Docker

Activity **Webhooks for Docker**

Webhook URL ⓘ `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` ⓘ

API token * `.....` ⓘ **CREATE VARIABLE** **GENERATE TOKEN**

Image ⓘ `Image`

Tag ⓘ `Tags`

SAVE **CANCEL**

- j Fare clic su **Salva**.

Viene visualizzata la scheda webhook con il webhook Docker abilitato. Se si desidera eseguire un inserimento fittizio nel repository di Docker Hub senza attivare il webhook Docker ed eseguire una pipeline, fare clic su **Disabilita**.

- 4 Nella pipeline CD configurare l'attività di distribuzione Kubernetes.
 - a Nelle proprietà dell'attività Kubernetes selezionare il cluster Kubernetes di sviluppo.
 - b Selezionare l'azione **Crea**.

- c Selezionare **Definizione locale** per l'origine payload.
- d Selezionare il file YAML locale.

Ad esempio, Docker Hub potrebbe inviare questa definizione YAML locale come payload all'URL nel 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#\nVOLUME\u0009\u0009[/var/cache/apt-cacher-ng]\nRUN\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n#\nEXPOSE\n\u0009\u000993142\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"
  }
}
```

L'API che crea il webhook in Docker Hub utilizza

questo modulo: https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/

Il corpo del codice JSON è simile a:

```
{
  "name": "demo_webhook",
```

```
"webhooks": [
{
"name": "demo_webhook",
"hook_url": "http://www.google.com"
}
]
}
```

Per ricevere gli eventi dal server Docker Hub, lo schema di autenticazione per il webhook Docker creato in Code Stream utilizza un meccanismo di autenticazione basato su un elenco di elementi consentiti con un token di stringa casuale per il webhook. Filtra gli eventi in base al token sicuro, che è possibile aggiungere a `hook_url`.

Code Stream può verificare qualsiasi richiesta dal server Docker Hub utilizzando il token di sicurezza configurato. Ad esempio: `hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""`

- 5 Creare un artefatto Docker nel repository Docker Hub. In alternativa, aggiornare un artefatto esistente.
- 6 Per controllare che il trigger si sia verificato e per vedere l'attività nel webhook Docker, fare clic su **Trigger > Docker > Attività**.

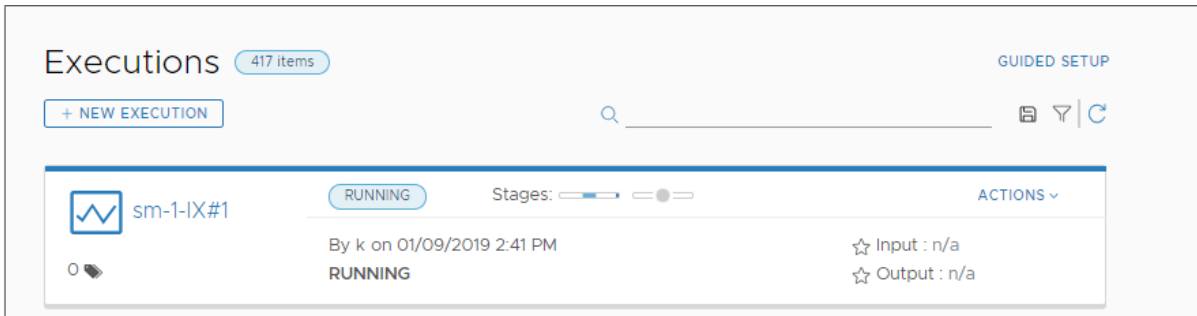
Docker

GUIDED SETUP

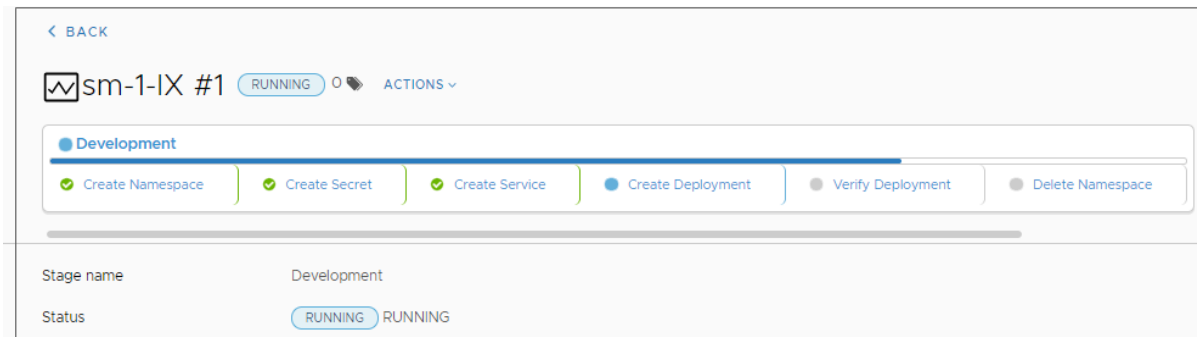
Activity
Webhooks for Docker

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 Fare clic su **Esecuzioni** e osservare la pipeline durante l'esecuzione.



- 8 Fare clic sulla fase in esecuzione e visualizzare le attività durante l'esecuzione della pipeline.



Risultati

Congratulazioni! È possibile configurare il trigger Docker per eseguire la pipeline di consegna continua in modo continuativo. La pipeline può ora caricare artefatti Docker nuovi e aggiornati nel repository di Docker Hub.

Operazioni successive

Verificare che l'artefatto nuovo o aggiornato venga distribuito nell'host Docker nel cluster Kubernetes di sviluppo.

Come utilizzare il trigger Git in Code Stream per eseguire una pipeline

Amministratori e sviluppatori di Code Stream possono integrare Code Stream con il ciclo di vita di Git utilizzando il trigger Gerrit. Quando si modifica il codice in GitHub, GitLab o Bitbucket Enterprise, l'evento comunica con Code Stream tramite un webhook e attiva una pipeline. Il webhook funziona con le versioni Enterprise locali di GitLab, GitHub e Bitbucket quando sia Cloud Assembly sia la versione Enterprise sono raggiungibili nella stessa rete.

Quando si aggiunge il webhook per Git in Code Stream, crea un webhook anche nel repository GitHub, GitLab o Bitbucket. Se si aggiorna o si elimina il webhook in un secondo momento, tale azione comporta l'aggiornamento o l'eliminazione del webhook anche in GitHub, GitLab o Bitbucket.

La definizione del webhook deve includere un endpoint Git nel ramo del repository che si desidera monitorare. Per creare il webhook, Code Stream utilizza l'endpoint Git. Se l'endpoint non esiste, è possibile crearlo quando si aggiunge il webhook. In questo esempio si presuppone l'esistenza di un endpoint Git predefinito in GitHub.

Nota Per creare un webhook, l'endpoint Git deve utilizzare un token privato per l'autenticazione. Non può utilizzare una password.

È possibile creare più webhook per rami diversi utilizzando lo stesso endpoint Git e specificando valori diversi per il nome del ramo nella pagina di configurazione del webhook. Per creare un altro webhook per un altro ramo nello stesso repository Git, non è necessario clonare l'endpoint Git più volte per più rami. È infatti sufficiente specificare il nome del ramo nel webhook in modo da poter riutilizzare l'endpoint Git. Se il ramo nel webhook Git è lo stesso ramo dell'endpoint, non è necessario specificare il nome del ramo nella pagina del webhook Git.

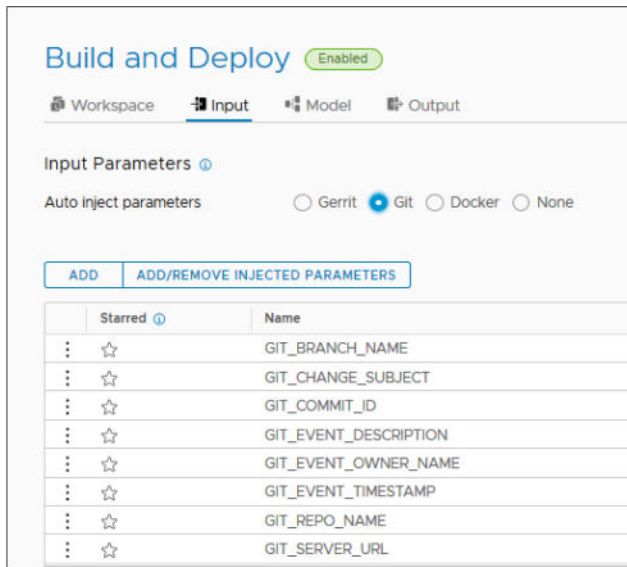
In questo esempio viene illustrato come utilizzare il trigger Git con un repository di GitHub, ma i prerequisiti includono i preparativi necessari se viene utilizzato un altro tipo di server Git.

Prerequisiti

- Verificare di essere un membro di un progetto in Code Stream. In caso contrario, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).
- Assicurarsi di disporre di un endpoint Git nel ramo GitHub che si desidera monitorare. Vedere [Come integrare Code Stream con Git](#).
- Verificare di disporre dei diritti necessari per creare un webhook nel repository Git.
- Se si configura un webhook in GitLab, modificare le impostazioni di rete predefinite in GitLab Enterprise per attivare le richieste in uscita e consentire la creazione di webhook locali.

Nota Questa modifica è necessaria solo per GitLab Enterprise. Queste impostazioni non si applicano a GitHub o Bitbucket.

- a Accedere all'istanza di GitLab Enterprise come amministratore.
- b Passare alle impostazioni di rete utilizzando un URL, ad esempio `http://{gitlab-server}/admin/application_settings/network`.
- c Espandere **Richieste in uscita** e fare clic su:
 - Consenti richieste alla rete locale dal webhook e dai servizi.
 - Consenti richieste alla rete locale dall'hook di sistema.
- Per le pipeline che si desidera attivare, verificare di aver impostato le proprietà di input per inviare i parametri Git quando viene eseguita la pipeline.



Per informazioni sulla creazione dei parametri di input, vedere [Pianificazione di una creazione nativa di integrazione e consegna continue in Code Stream](#) prima dell'aggiunta manuale di attività.

Procedura

- 1 In Code Stream, fare clic su **Trigger > Git**.
- 2 Fare clic sulla scheda **Webhook per Git**, quindi su **Nuovo webhook per Git**.
 - a Selezionare un progetto.
 - b Immettere un nome e una descrizione significativi per il webhook.

- c Selezionare un endpoint Git configurato per il ramo che si desidera monitorare.

Quando si crea il webhook, la definizione del webhook include i dettagli dell'endpoint corrente.

- Se in seguito si modifica il tipo Git, il tipo di server Git o l'URL del repository Git nell'endpoint, il webhook non sarà più in grado di attivare una pipeline poiché tenterà di accedere al repository Git utilizzando i dettagli dell'endpoint originale. È necessario eliminare il webhook e crearlo di nuovo con l'endpoint.
- Se in seguito si modifica il tipo di autenticazione, il nome utente o il token privato nell'endpoint, il webhook continuerà a funzionare.
- Se si utilizza un repository BitBucket, l'URL del repository deve essere in uno di questi formati: `https://api.bitbucket.org/{user}/{repo name}` o `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`.

Nota Se in precedenza è stato creato un webhook utilizzando un endpoint Git che utilizza una password per l'autenticazione di base, è necessario eliminare e ridefinire il webhook con un endpoint Git che utilizza un token privato per l'autenticazione.

Vedere [Come integrare Code Stream con Git](#).

- d (Facoltativo) Immettere il ramo di cui si desidera monitorare il webhook.

Se il ramo non viene specificato, il webhook monitora il ramo configurato per l'endpoint Git.

- e (Facoltativo) Generare un token segreto per il webhook.

Se si utilizza un token segreto, Code Stream genera un token di stringa casuale per il webhook. Quindi, quando il webhook riceve i dati dell'evento Git, invia i dati con il token segreto. Code Stream utilizza le informazioni per stabilire se le chiamate provengono dall'origine prevista, ad esempio il ramo, il repository o l'istanza configurati di GitHub. Il token segreto offre un livello aggiuntivo di sicurezza. utilizzato per verificare che i dati dell'evento Git provengano dall'origine corretta.

- f (Facoltativo) Specificare inclusioni o esclusioni di file come condizioni per il trigger.
- Inclusioni di file. Se uno qualsiasi dei file in un commit corrisponde ai file specificati nel regex o nei percorsi di inclusione, il commit attiva le pipeline. Con un regex specificato, Code Stream attiva le pipeline solo quando i nomi di file nell'insieme di modifiche corrispondono all'espressione fornita. Il filtro regex è utile quando si configura un trigger per più pipeline in un singolo repository.
 - Esclusioni di file. Quando tutti file in un commit corrispondono ai file specificati nel regex o nei percorsi di esclusione, le pipeline non vengono attivate.
 - Assegnazione di priorità alle esclusioni. Quando è attivata, l'opzione Assegna priorità all'esclusione garantisce che le pipeline non vengano attivate anche se uno qualsiasi dei file di un commit corrisponde ai file specificati nel regex o nei percorsi di esclusione. L'opzione è disattivata per impostazione predefinita.

Se le condizioni soddisfano sia l'inclusione di file sia l'esclusione di file, le pipeline non vengono attivate.

Nell'esempio seguente sia le inclusioni che le esclusioni sono condizioni per l'attivazione.

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	<input type="checkbox"/>		

- Per le inclusioni dei file, un commit con una modifica a `runtime/src/main/a.java` o a qualsiasi file Java attiva le pipeline configurate nella configurazione dell'evento.
 - Per le esclusioni dei file, solo un commit con modifiche in entrambi i file non attiva le pipeline configurate nelle configurazioni dell'evento.
- g Per l'evento Git, selezionare una richiesta **Push** o **Pull**.

h Immettere il token dell'API.

Il token dell'API CSP autentica l'utente per le connessioni API esterne con Code Stream.
Per ottenere il token API:

- 1 Fare clic su **Genera Token**.
- 2 Immettere l'indirizzo e-mail associato al nome utente e alla password, quindi fare clic su **Genera**.

Il token generato è valido per sei mesi. È anche noto come token di aggiornamento.

- Per mantenere il token come una variabile per un utilizzo futuro, fare clic su **Crea variabile**, assegnare un nome alla variabile e fare clic su **Salva**.
- Per mantenere il token come valore di testo per un utilizzo futuro, fare clic su **Copia** e incollare il token in un file di testo da salvare in locale.

È possibile scegliere di creare una variabile e archiviare il token in un file di testo per un utilizzo futuro.

- 3 Fare clic su **Chiudi**.

i Selezionare la pipeline per il webhook da attivare.

Se la pipeline include parametri di input aggiunti personalizzati, nell'elenco Parametri di input vengono visualizzati i parametri e i valori. È possibile immettere valori per i parametri di input che vengono passati alla pipeline con l'evento trigger. Oppure è possibile lasciare vuoti i valori o utilizzare i valori predefiniti, se sono specificati.

Per informazioni sui parametri di input di invio automatico per i trigger Git, vedere i [Prerequisiti](#).

j Fare clic su **Crea**.

Il webhook viene visualizzato come nuova scheda.

3 Fare clic sulla scheda del webhook.

Quando viene visualizzato nuovamente il modulo dei dati del webhook, nella parte alta è presente un URL. Il webhook Git si connette al repository GitHub tramite l'URL del webhook.

Git

Activity

Webhooks for Git

Webhook URL ⓘ

https://ca10118e-4b34-4389-8000-000000000000/codestream/api/git-webhook-listeners/963b2287-527f-4e9b-b000-000000000000

Project

test

Name *

test-webhook

Description

Description

Endpoint

DemoApp-Git

Branch ⓘ

master

Secret token ⓘ *

GYH0cBWZx4dUn47Y/KA8H/BOkts=

GENERATE

File ⓘ

Inclusions

--Select-- ▾ Value +

Exclusions

--Select-- ▾ Value +

Prioritize Exclusion

☐

Trigger

For Git

☒ PUSH ☐ PULL REQUEST

API token *

.....

⛔

CREATE VARIABLE

GENERATE TOKEN

Pipeline *

CICD-2 ⓘ

Comments

Execution trigger delay ⓘ

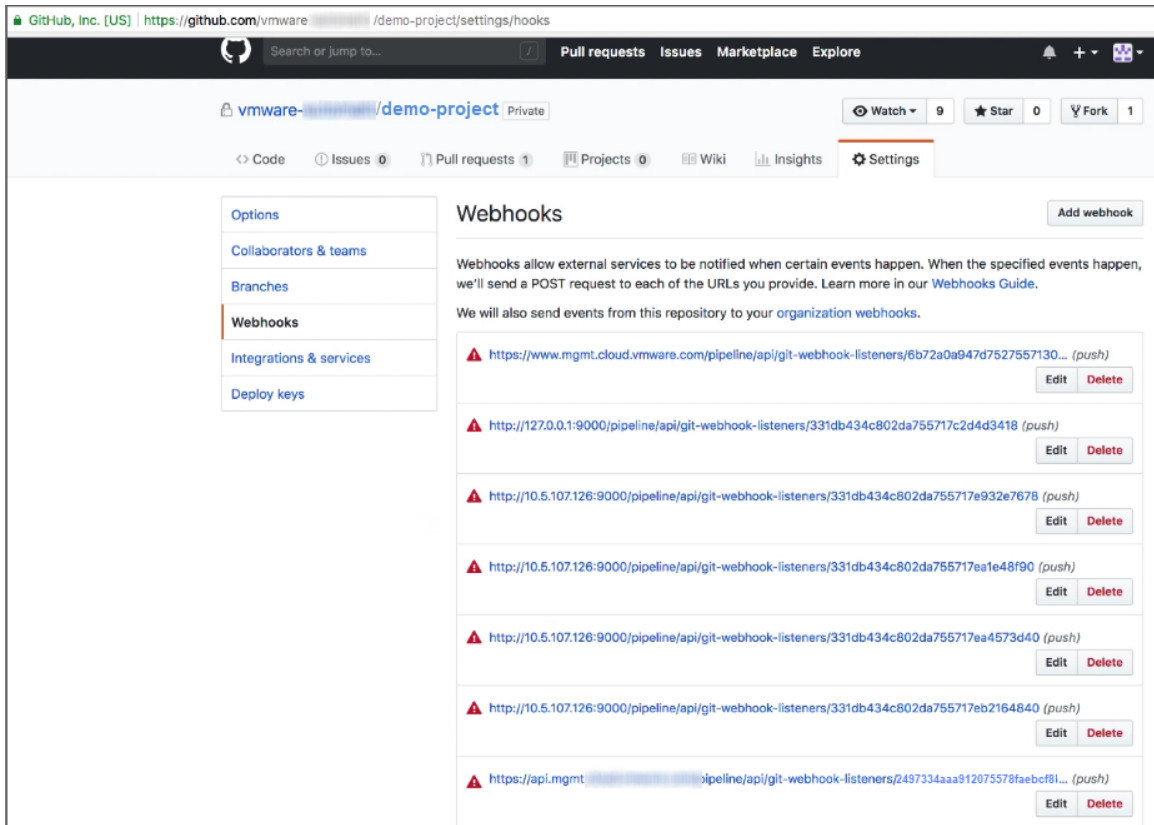
1

SAVE

CANCEL

- 4 In una nuova finestra del browser, aprire il repository GitHub che si connette tramite il webhook.
 - a Per visualizzare il webhook aggiunto in Code Stream, fare clic sulla scheda **Impostazioni** e selezionare **Webhook**.

Nella parte inferiore dell'elenco dei webhook viene visualizzato lo stesso URL del webhook.



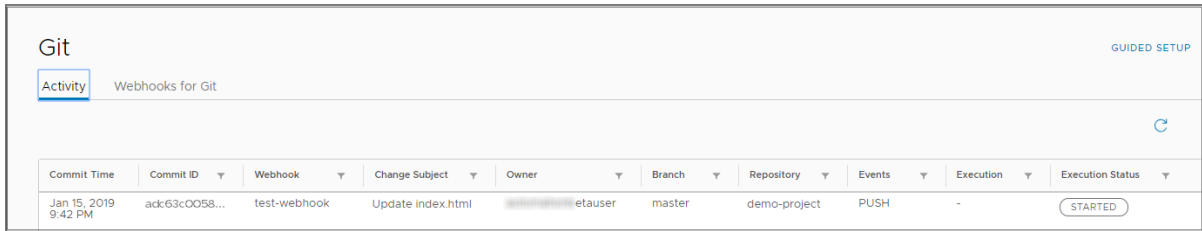
- b Per apportare una modifica al codice, fare clic sulla scheda **Codice** e selezionare un file nel ramo. Dopo aver modificato il file, eseguire il commit della modifica.
 - c Per verificare che l'URL del webhook funzioni, fare clic sulla scheda **Impostazioni** e selezionare nuovamente **Webhook**.

Nella parte inferiore dell'elenco dei webhook viene visualizzato un segno di spunta verde accanto all'URL del webhook.



- 5 Tornare a Code Stream per visualizzare l'attività sul webhook Git. Fare clic su **Trigger** > **Git** > **Attività**.

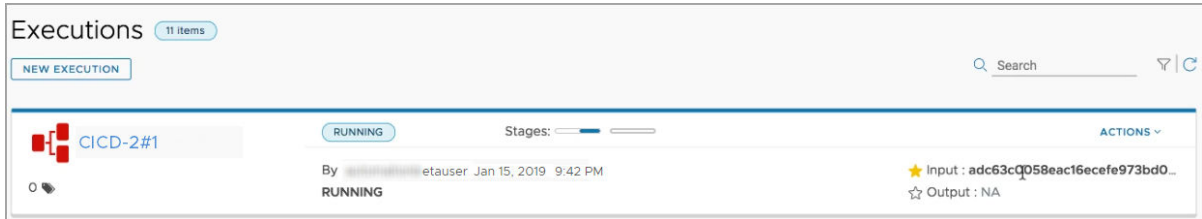
In Stato esecuzione verificare che l'esecuzione della pipeline sia stata avviata.



Commit Time	Commit ID	Webhook	Change Subject	Owner	Branch	Repository	Events	Execution	Execution Status
Jan 15, 2019 9:42 PM	ack:63c0058...	test-webhook	Update index.html	etauser	master	demo-project	PUSH	-	STARTED

6 Fare clic su **Esecuzioni** e tenere traccia della pipeline durante l'esecuzione.

Per osservare l'esecuzione della pipeline, è possibile fare clic su Aggiorna.



Risultati

Congratulazioni! Il trigger per Git è stato utilizzato correttamente.

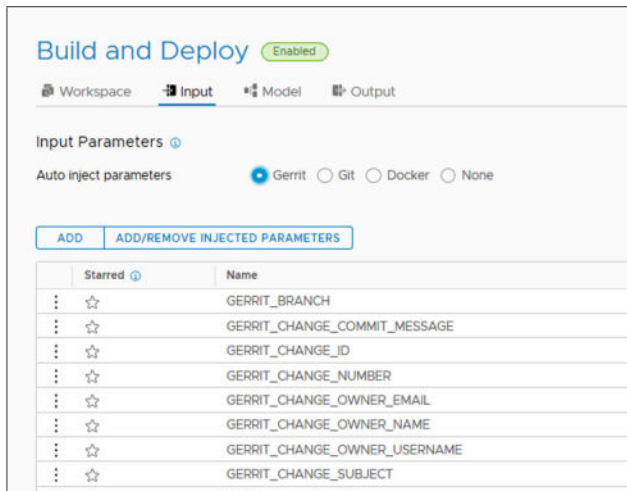
Come utilizzare il trigger Gerrit in Code Stream per eseguire una pipeline

Amministratori e sviluppatori di Code Stream possono integrare Code Stream con il ciclo di vita di revisione del codice Gerrit utilizzando il trigger Gerrit. L'evento attiva una pipeline da eseguire quando si crea un set di patch, si pubblicano bozze, si uniscono modifiche al codice nel progetto Gerrit o si inseriscono direttamente le modifiche nel ramo Git.

Quando si aggiunge il trigger Gerrit, si seleziona un listener Gerrit, un progetto Gerrit nel server Gerrit e si configurano gli eventi Gerrit. In questo esempio, si configura innanzitutto un listener Gerrit e quindi si utilizza tale listener in un trigger Gerrit con due eventi in tre pipeline diverse.

Prerequisiti

- Verificare di essere un membro di un progetto in Code Stream. In caso contrario, chiedere a un amministratore di Code Stream di essere aggiunti come membri di un progetto. Vedere [Come aggiungere un progetto in Code Stream](#).
- Verificare che in Code Stream sia presente un endpoint Gerrit configurato. Vedere [Come integrare Code Stream con Gerrit](#).
- Assicurarsi di conoscere la versione di Gerrit.
- Affinché le pipeline vengano attivate, impostare le proprietà di input della pipeline come **Gerrit**, in modo da consentire alla pipeline di ricevere i parametri Gerrit come input quando viene eseguita.



Per informazioni sulla creazione dei parametri di input, vedere [Pianificazione di una creazione nativa di integrazione e consegna continue in Code Stream prima dell'aggiunta manuale di attività](#).

Procedura

- 1 In Code Stream, fare clic su **Trigger > Gerrit**.
- 2 (Facoltativo) Fare clic sulla scheda **Listener**, quindi su **Nuovo listener**.

Nota Se il listener Gerrit che si intende utilizzare per il trigger Gerrit è già definito, ignorare questo passaggio.

- a Selezionare un progetto.
- b Immettere un nome per il listener Gerrit.
- c Selezionare un endpoint Gerrit

- d Immettere il token dell'API.

Il token dell'API CSP autentica l'utente per le connessioni API esterne con Code Stream. Per ottenere il token API:

- 1 Fare clic su **Genera Token**.
- 2 Immettere l'indirizzo e-mail associato al nome utente e alla password, quindi fare clic su **Genera**.

Il token generato è valido per sei mesi. È anche noto come token di aggiornamento.

- Per mantenere il token come una variabile per un utilizzo futuro, fare clic su **Crea variabile**, assegnare un nome alla variabile e fare clic su **Salva**.
- Per mantenere il token come valore di testo per un utilizzo futuro, fare clic su **Copia** e incollare il token in un file di testo da salvare in locale.

È possibile scegliere di creare una variabile e archiviare il token in un file di testo per un utilizzo futuro.

- 3 Fare clic su **Chiudi**.

Se è stata creata una variabile, nel token dell'API viene visualizzato il nome della variabile immesso utilizzando il binding del dollaro. Se il token è stato copiato, il token dell'API mostra il token mascherato.

The screenshot shows the 'Gerrit' configuration page with the 'Listeners' tab selected. The form contains the following fields and controls:

- Project**: test1 (with a clear button)
- Name**: Gerrit-Demo-Listener
- Endpoint**: corporate-gerrit (with a dropdown arrow)
- API token**: \${var.CSuser API Token} (with a green checkmark icon)

At the bottom of the form, there are five buttons: **CREATE** (dark blue), **VALIDATE** (light blue), **CANCEL** (light blue), **CREATE VARIABLE** (light blue), and **GENERATE TOKEN** (light blue).

- e Per convalidare i dettagli del token e dell'endpoint, fare clic su **Convalida**.

Il token scade dopo 90 giorni.

- f Fare clic su **Crea**.
- g Nella scheda del listener, fare clic su **Connetti**.

Il listener avvia il monitoraggio di tutte le attività nel server Gerrit e ascolta tutti i trigger abilitati in tale server. Per interrompere l'ascolto di un trigger in tale server, disattivare il trigger.

Nota Per aggiornare un endpoint Gerrit connesso a un listener, è necessario disconnettere il listener prima di aggiornare l'endpoint.

- Fare clic su **Configura > Trigger > Gerrit**.
 - Fare clic sulla scheda **Listener**.
 - Fare clic su **Disconnetti** nel listener connesso all'endpoint che si desidera aggiornare.
-

- 3 Fare clic sulla scheda **Trigger** e quindi su **Nuovo trigger**.

- 4 Selezionare un progetto nel server Gerrit.

- 5 Immettere un nome.

Il nome del trigger Gerrit deve essere univoco.

- 6 Selezionare un listener Gerrit configurato.

Utilizzando il listener Gerrit, Code Stream fornisce un elenco dei progetti Gerrit disponibili nel server.

- 7 Selezionare un progetto nel server Gerrit.

- 8 Immettere nel repository il ramo che verrà monitorato dal listener Gerrit.

- 9 (Facoltativo) Specificare inclusioni o esclusioni di file come condizioni per il trigger.

- Specificare le inclusioni di file che attivano la pipeline. Quando uno qualsiasi dei file in un commit corrisponde ai file specificati nel regex o nei percorsi di inclusione, le pipeline vengono attivate. Con un regex specificato, Code Stream attiva solo le pipeline i cui nomi file nell'insieme di modifiche corrispondono all'espressione specificata. Il filtro regex è utile quando si configura un trigger per più pipeline in un singolo repository.
- È possibile specificare esclusioni di file che impediscano l'attivazione delle pipeline. Quando tutti i file in un commit corrispondono ai file specificati nel regex o nei percorsi di esclusione, le pipeline non vengono attivate.
- Quando è attivata, l'opzione **Assegna priorità all'esclusione** fa in modo che le pipeline non vengano attivate. Le pipeline non vengono attivate anche se uno qualsiasi dei file in un commit corrisponde ai file specificati nel regex o nei percorsi di esclusione. L'opzione **Assegna priorità all'esclusione** è disattivata per impostazione predefinita.

Se le condizioni soddisfano sia l'inclusione dei file sia l'esclusione dei file, le pipeline non vengono attivate.

Nell'esempio seguente, sia le inclusioni sia le esclusioni di file sono condizioni per l'attivazione.

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	<input type="checkbox"/>		

- Per le inclusioni di file, un commit con una modifica in `runtime/src/main/a.java` o in un file Java qualsiasi attiva le pipeline impostate nella configurazione dell'evento.
- Per le esclusioni di file, un commit con modifiche solo in entrambi i file non attiva le pipeline impostate nella configurazione dell'evento.

10 Fare clic su **Nuova configurazione**.

- Per un evento Gerrit, selezionare **Set di patch creato**, **Bozza pubblicata** o **Unione modifiche**. Oppure, per un push diretto in Git che escluda Gerrit, selezionare **Push Git diretto**.

Nota A partire dalla versione 2.15 di Gerrit, le modifiche bozza e i set di modifiche bozza non sono più supportati. Pertanto, se si esegue Gerrit 2.15 o una versione successiva, **Bozza pubblicata** non è un evento supportato.

- Selezionare la pipeline che verrà attivata.

Se la pipeline include parametri di input aggiunti personalizzati, nell'elenco Parametri di input vengono visualizzati i parametri e i valori. È possibile immettere valori per i parametri di input da trasmettere alla pipeline con l'evento trigger. Oppure è possibile lasciare vuoti i valori o utilizzare i valori predefiniti.

Nota Se sono definiti valori predefiniti:

- Tutti i valori immessi per i parametri di input sostituiranno i valori predefiniti definiti nel modello della pipeline.
- I valori predefiniti nella configurazione del trigger non cambieranno al variare dei valori dei parametri nel modello della pipeline.

Per informazioni sui parametri di input di invio automatico per i trigger Gerrit, vedere i [Prerequisiti](#).

- c Per **Set di patch creato**, **Bozza pubblicata** e **Unione modifiche**, alcune azioni vengono visualizzate con etichette per impostazione predefinita. È possibile modificare l'etichetta o aggiungere commenti. Quindi, quando la pipeline viene eseguita, l'etichetta o il commento vengono visualizzati nella scheda **Attività** come **Azione intrapresa** per la pipeline.

La configurazione dell'evento Gerrit consente di inserire commenti utilizzando una variabile per il commento Success o il commento Failure. Ad esempio: `${var.success}` e `${var.failure}`.

- d Fare clic su **Salva**.

Per aggiungere più eventi trigger in più pipeline, fare di nuovo clic su **Nuova configurazione**.

Nell'esempio seguente è possibile visualizzare gli eventi per tre pipeline:

- Se si verifica un evento **Unione modifiche** nel progetto Gerrit, viene attivata la pipeline denominata **Gerrit-Pipeline**.
- Se si verifica un evento **Set di patch creato** nel progetto Gerrit, vengono attivate le pipeline denominate **Gerrit-Trigger-Pipeline** e **Gerrit-Demo-Pipeline**.

Gerrit

GUIDED SETUP

Activity
Triggers
Listeners

Project *

Name *

Gerrit Listener *

Gerrit project *

Branch *

test1 ⓧ

Gerrit-Demo-Trigger

Gerrit-Demo-Listener ▼

Gerrit-Demo-Project ▼

master

File ①

Inclusions

Exclusions

Prioritize Exclusion

-- Select Type -- ▼

-- Select Type -- ▼

☐

value

value

+

+

+ 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 Fare clic su **Crea**.

Il trigger Gerrit viene visualizzato come nuova sezione nella scheda **Attivazioni** ed è impostato come **Disabilitato** per impostazione predefinita.

12 Nella scheda del trigger fare clic su **Abilita**.

Una volta abilitato, il trigger può utilizzare il listener Gerrit, che avvia il monitoraggio degli eventi che si verificano nel ramo del progetto Gerrit.

Per creare un trigger con le stesse condizioni di inclusione file o di esclusione file, ma con un repository diverso da quello incluso quando il trigger è stato creato, nella scheda del trigger fare clic su **Azioni > Clona**. Quindi, nel trigger clonato, fare clic su **Apri** e modificare i parametri.

Risultati

Congratulazioni! È stato configurato un trigger Gerrit con due eventi in tre pipeline diverse.

Operazioni successive

Dopo aver eseguito il commit di una modifica del codice nel progetto Gerrit, osservare la scheda **Attività** per l'evento Gerrit in Code Stream. Verificare che l'elenco di attività includa voci che corrispondono a ogni esecuzione della pipeline nella configurazione del trigger.

Quando si verifica un evento, solo le pipeline nel trigger Gerrit correlate a tale tipo di evento specifico possono essere eseguite. In questo esempio, se viene creato un set di patch, vengono eseguite solo **Gerrit-Trigger-Pipeline** e **Gerrit-Demo-Pipeline**.

Le informazioni nelle colonne della scheda **Attività** descrivono ciascun evento del trigger Gerrit. È possibile selezionare le colonne visualizzate facendo clic sull'icona della colonna disponibile sotto la tabella.

- Le colonne **Modifica oggetto** ed **Esecuzione** sono vuote quando il trigger è un push Git diretto.
- La colonna **Trigger Gerrit** include il trigger che ha creato l'evento.
- La colonna **Listener** è disattivata per impostazione predefinita. Quando è selezionata, la colonna indica il listener Gerrit che ha ricevuto l'evento. Un singolo listener può essere visualizzato come associato a più trigger.
- La colonna **Tipo di trigger** è disattivata per impostazione predefinita. Quando è selezionata, la colonna indica il tipo di trigger AUTOMATIC o MANUAL.
- Le altre colonne includono **Tempo di commit**, **Numero modifica**, **Stato**, **Messaggio**, **Azione intrapresa**, **Utente**, **Progetto Gerrit**, **Ramo** ed **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	111Dummy	Gerrit-Pipeline #1	COMPLETED	Execution Completed.	Verified +1	Orlando.Spryann@gm	test1	Gerrit-Demo-Trigger	master	Change Merged
⋮	Nov 12, 2019, 12:50:04 PM	19570 /6	1111Dummy	Gerrit-Pipeline #2	WAITING	Stage0.Task0: Execution Waiting for User Action.		Orlando.Spryann@gm	test1	Gerrit-Demo-Trigger	master	Change Merged
			1111Dummy	Gerrit-Demo-Pipeline #1	FAILED	Stage0.Task0: User Operation request has been rejected by Fritz.	Verified -1	Orlando.Spryann@gm	test1	Gerrit-Demo-Trigger	master	Patchset created
			1111Dummy	Gerrit-Trigger-Pipeline #1	WAITING	Stage0.Task0: Execution Waiting for User Action.		Orlando.Spryann@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

Per controllare l'attività per un'esecuzione di pipeline completata o non riuscita, fare clic sui tre puntini a sinistra di una voce nella schermata Attività.

- Se la pipeline non viene eseguita a causa di un errore nel modello della pipeline o di un altro problema, correggere l'errore e selezionare **Esegui nuovamente** per eseguire di nuovo la pipeline.
- Se la pipeline non viene eseguita a causa di un problema di connettività di rete o di un altro problema, selezionare **Riprendi** per riavviare la stessa esecuzione della pipeline e risparmiare tempo di esecuzione.
- Utilizzare **Visualizza esecuzione** per aprire la visualizzazione dell'esecuzione della pipeline. Vedere [Come eseguire una pipeline e visualizzare i risultati](#).
- Utilizzare **Elimina** per eliminare la voce dalla schermata Attività.

Se un evento Gerrit non riesce ad attivare una pipeline, è possibile fare clic su **Trigger manuale**, selezionare il trigger Gerrit, immettere l'ID modifica e fare clic su **Esegui**.

Monitoraggio delle pipeline in Code Stream



Un amministratore o uno sviluppatore di Code Stream devono conoscere le prestazioni delle proprie pipeline in Code Stream. Occorre sapere quanto efficacemente le pipeline rilasciano il codice nelle fasi di sviluppo, test e produzione.

Per ottenere i dettagli, utilizzare i dashboard di Code Stream per monitorare le tendenze e i risultati dell'esecuzione di una pipeline. È possibile utilizzare i dashboard della pipeline predefiniti per monitorare una singola pipeline oppure creare dashboard personalizzati per il monitoraggio di più pipeline.

- Le metriche della pipeline includono le statistiche, ad esempio i tempi medi, disponibili nel dashboard della pipeline.
- Per visualizzare le metriche di più pipeline, utilizzare i dashboard personalizzati.

Questo capitolo include i seguenti argomenti:

- [Elementi visualizzati nel dashboard della pipeline in Code Stream](#)
- [Come utilizzare dashboard personalizzati per tenere traccia degli indicatori di prestazioni chiave per la pipeline in Code Stream](#)

Elementi visualizzati nel dashboard della pipeline in Code Stream

Un dashboard della pipeline è una visualizzazione dei risultati di una pipeline specifica che è stata eseguita, ad esempio le tendenze, i problemi principali e le modifiche riuscite. Code Stream crea il dashboard della pipeline quando si crea una pipeline.

Il dashboard contiene i widget che mostrano i risultati dell'esecuzione della pipeline.

Widget dei conteggi degli stati di esecuzione della pipeline

È possibile visualizzare il numero totale di esecuzioni di una pipeline in un determinato periodo di tempo raggruppate per stato: Completata, Non riuscita o Annullata. Per conoscere come cambia lo stato dell'esecuzione della pipeline in periodi di tempo più lunghi o più brevi, modificare la durata nella visualizzazione.

Widget delle statistiche di esecuzione della pipeline

Le statistiche di esecuzione della pipeline includono i tempi medi per il ripristino, la consegna o la mancata riuscita di una pipeline nel tempo.

Gli stati seguenti si applicano a tutte le esecuzioni della pipeline:

- Completata
- Non riuscita
- In attesa
- In esecuzione
- Annullata
- In coda
- Non avviata
- Rollback in corso
- Rollback completato
- Rollback non riuscito
- In pausa

Tabella 8-1. Calcolo dei tempi medi

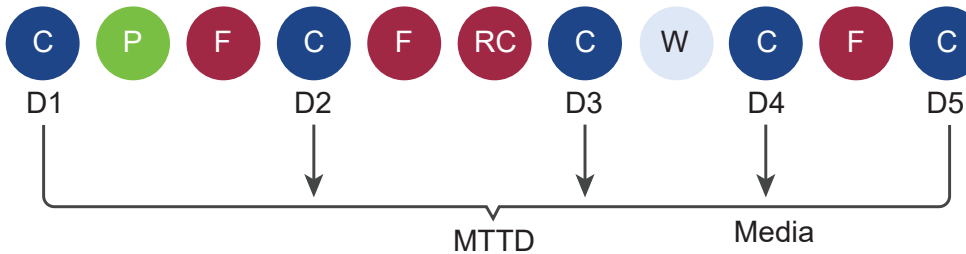
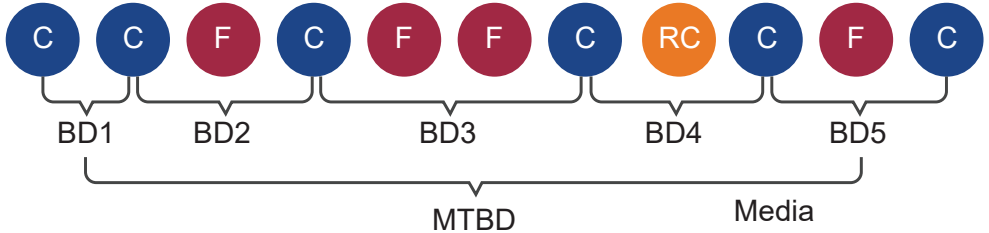
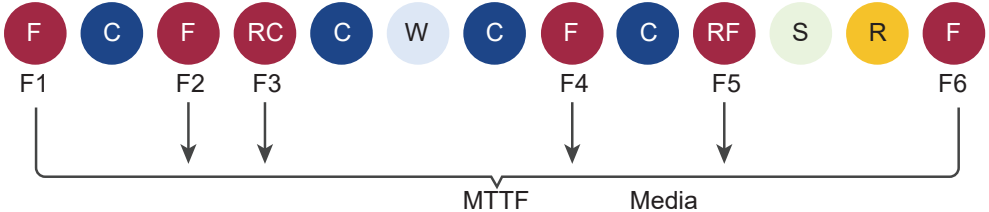
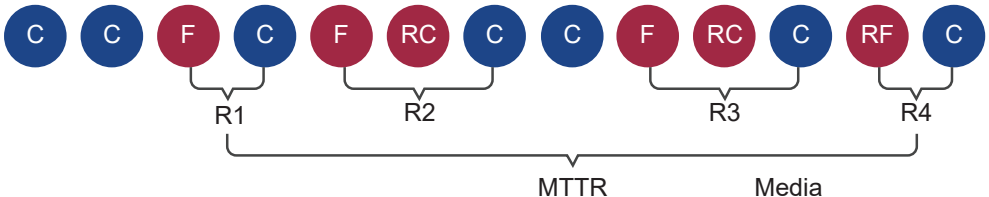
Cosa viene calcolato...	Che cosa significa...
CI media	Tempo medio trascorso nella fase di integrazione continua, misurato in base al tempo impiegato nel tipo di attività CI.
Tempo medio di consegna (MTTD)	<p>Durata media di tutte le esecuzioni COMPLETED in un determinato periodo di tempo. D1, D2 e così via rappresentano la quantità di tempo per la consegna di ciascuna esecuzione COMPLETED.</p> 

Tabella 8-1. Calcolo dei tempi medi (continua)

Cosa viene calcolato...	Che cosa significa...
Tempo medio tra le consegne (MTBD)	<p>Tempo medio trascorso tra le consegne riuscite in un determinato periodo di tempo. Il tempo trascorso tra due esecuzioni COMPLETED consecutive è il tempo intercorso tra consegne riuscite, come BD1, BD2 e così via. MTBD indica la frequenza di aggiornamento di un ambiente di produzione.</p> 
Tempo medio fallimento (MTTF)	<p>Durata media delle esecuzioni che terminano con gli stati FAILED, ROLLBACK_COMPLETED o ROLLBACK_FAILED in un determinato periodo di tempo. F1, F2 e così via rappresentano il tempo trascorso prima che l'esecuzione termini con stato FAILURE, ROLLBACK_COMPLETED o ROLLBACK_FAILED.</p> 
Tempo medio per il ripristino (MTTR)	<p>Tempo medio per il ripristino da un fallimento in un determinato periodo di tempo. Il tempo di ripristino da un fallimento è il tempo trascorso tra un'esecuzione con stato finale FAILED, ROLLBACK_COMPLETED o ROLLBACK_FAILED e l'esecuzione riuscita immediatamente successiva con stato COMPLETED. R1, R2 e così via rappresentano la quantità di tempo per il ripristino dopo ogni esecuzione FAILED o ROLLBACK_FAILED.</p> 

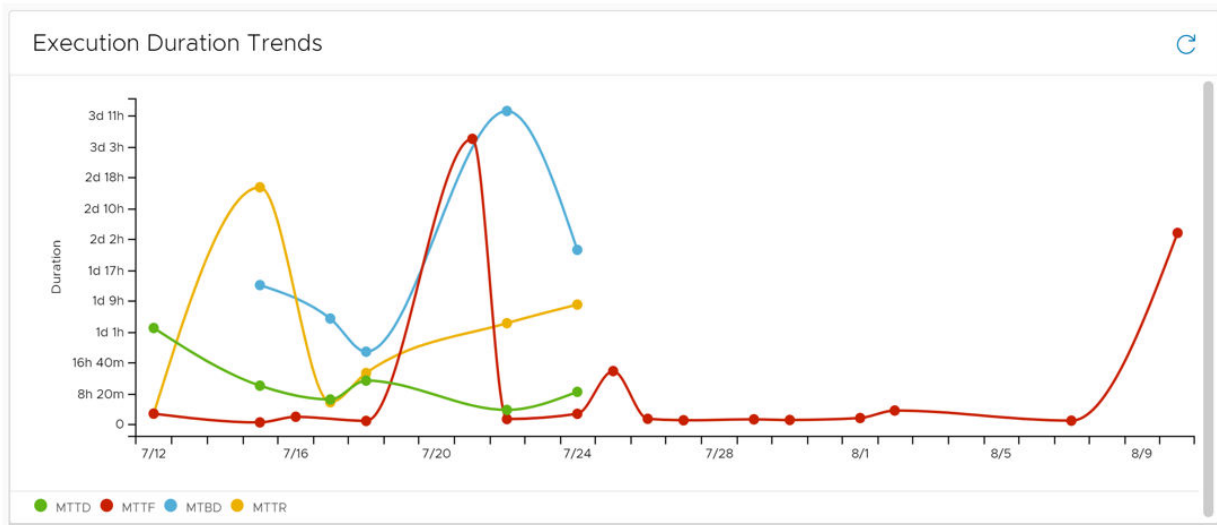
Widget Fasi non riuscite principali e Attività

Due widget mostrano le fasi non riuscite principali e le attività in una pipeline. Ogni valore indica il numero e la percentuale di errori per gli ambienti di sviluppo e post-sviluppo per ogni pipeline e progetto, come media settimanale o mensile. È possibile visualizzare gli errori principali per risolvere i problemi nel processo di automazione del rilascio.

È possibile configurare la visualizzazione per una determinata durata, ad esempio gli ultimi sette giorni, e prendere nota delle principali attività non riuscite durante quel periodo di tempo. Se si modifica l'ambiente o la pipeline e si esegue di nuovo la pipeline, quindi si controllano le principali attività non riuscite in un periodo di tempo più lungo, ad esempio gli ultimi 14 giorni, è possibile che le attività principali non riuscite siano cambiate. Questo risultato indica che la modifica del processo di automazione del rilascio ha migliorato la percentuale di esecuzioni riuscite della pipeline.

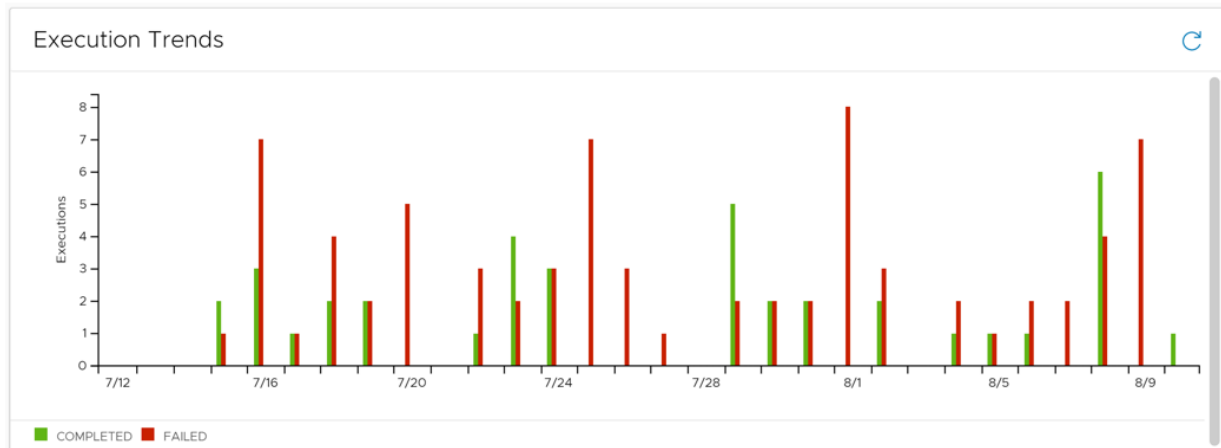
Widget Tendenze durata esecuzioni della pipeline

Le tendenze della durata delle esecuzioni della pipeline mostrano i valori MTDD, MTTF, MTBD e MTTR in un determinato periodo di tempo.



Widget Tendenze esecuzioni della pipeline

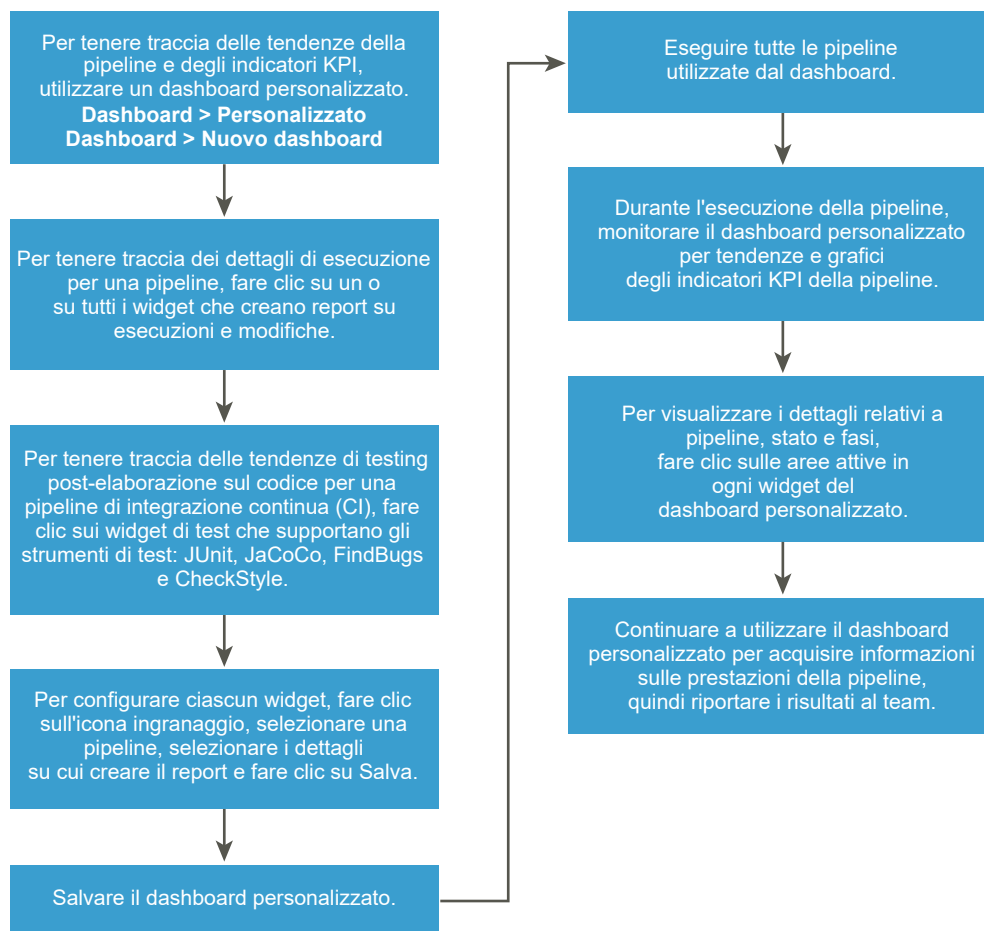
Il widget sulle tendenze delle esecuzioni della pipeline mostra le esecuzioni giornaliere totali di una pipeline, raggruppate per stato, in un determinato periodo di tempo. Ad eccezione del giorno corrente, nella maggior parte dei conteggi delle aggregazioni giornaliere vengono mostrate solo le esecuzioni COMPLETED e FAILED.



Come utilizzare dashboard personalizzati per tenere traccia degli indicatori di prestazioni chiave per la pipeline in Code Stream

Gli amministratori e gli sviluppatori di Code Stream possono creare dashboard personalizzati per visualizzare i risultati desiderati per una o più pipeline eseguite. Ad esempio, è possibile creare un dashboard a livello di progetto con indicatori KPI e metriche raccolti da più pipeline. Se viene segnalato un avviso di esecuzione o un errore, è possibile utilizzare il dashboard per risolvere l'errore.

Per tenere traccia delle tendenze e degli indicatori di prestazioni chiave per le pipeline utilizzando un dashboard personalizzato, aggiungere i widget al dashboard e configurarli per la creazione di report sulle pipeline.



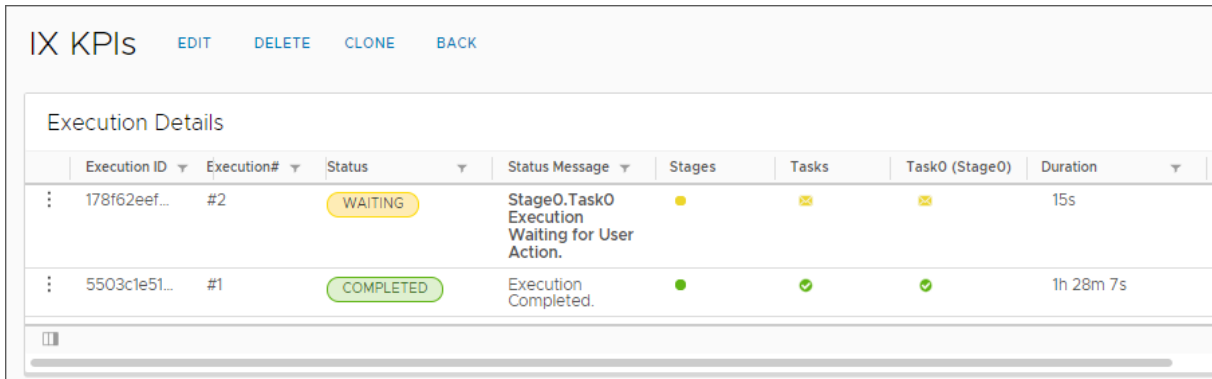
Prerequisiti

- Verificare che siano presenti una o più pipeline. Nell'interfaccia utente fare clic su **Pipeline**.
- Verificare che le pipeline che si intende monitorare siano state eseguite correttamente. Fare clic su **Esecuzioni**.

Procedura

- 1 Per creare un dashboard personalizzato, fare clic su **Dashboard > Dashboard personalizzati > Nuovo dashboard**.
- 2 Per personalizzare il dashboard in modo che crei report su determinate tendenze e indicatori di prestazioni chiave specifici per la pipeline, fare clic su un widget.

Ad esempio, per visualizzare i dettagli relativi allo stato, alle fasi, alle attività, al tempo e all'autore dell'esecuzione della pipeline, fare clic sul widget **Dettagli esecuzione**. In alternativa, per una pipeline di integrazione continua (CI), è possibile tenere traccia delle tendenze sulla post-elaborazione utilizzando i widget per JUnit, JaCoCo, FindBugs e CheckStyle.



IX KPIs								
EDIT DELETE CLONE BACK								
Execution Details								
	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 Configurare ciascun widget aggiunto.
 - a Nel widget fare clic sull'icona dell'ingranaggio.
 - b Selezionare una pipeline, impostare le opzioni disponibili e selezionare le colonne da visualizzare.
 - c Per salvare la configurazione del widget, fare clic su **Salva**.
 - d Per salvare il dashboard personalizzato, fare clic su **Salva**, quindi su **Chiudi**.
- 4 Per visualizzare ulteriori informazioni sulla pipeline, fare clic sulle aree attive dei widget.

Ad esempio, nel widget **Dettagli di esecuzione** fare clic su una voce nella colonna Stato per visualizzare ulteriori informazioni sull'esecuzione della pipeline. Oppure fare clic sul collegamento attivo nel widget **Ultima modifica riuscita** per visualizzare un riepilogo della fase e dell'attività della pipeline.

Risultati

Congratulazioni! È stato creato un dashboard personalizzato che monitora le tendenze e gli indicatori KPI per le pipeline.

Operazioni successive

Continuare a monitorare le prestazioni delle pipeline in Code Stream e condividere i risultati con il proprio manager e i team per migliorare il processo di rilascio delle applicazioni.

Ulteriori informazioni su Code Stream

9

Amministratori e sviluppatori di Code Stream possono ottenere informazioni più approfondite su Code Stream e sulle sue funzionalità nei modi descritti di seguito.

È possibile utilizzare questa documentazione per ulteriori informazioni sulle pipeline e le relative esecuzioni, su come aggiungere endpoint, su come aggiungere progetti e altro ancora.

Conoscere le autorizzazioni fornite dai ruoli. Scoprire come utilizzare le risorse limitate e richiedere approvazioni per le pipeline. Vedere [Come gestire accesso utente e approvazioni in Code Stream](#).

Capire l'importanza della funzione di ricerca per trovare processi o componenti specifici nelle pipeline, nelle esecuzioni o negli endpoint.

Questo capitolo include i seguenti argomenti:

- [Che cos'è la ricerca in Code Stream](#)
- [Altre risorse per amministratori e sviluppatori di Code Stream](#)

Che cos'è la ricerca in Code Stream

È possibile utilizzare la ricerca per individuare elementi specifici o altri componenti. Ad esempio può essere utile cercare le pipeline attivate o disattivate, poiché, se una pipeline è disattivata, non può essere eseguita.

Cosa è possibile cercare

È possibile effettuare ricerche in:

- Progetti
- Endpoint
- Pipeline
- Esecuzioni
- Dashboard della pipeline, dashboard personalizzati
- Server e trigger Gerrit
- Webhook Git
- Webhook Docker

È possibile eseguire la ricerca con filtro basato su colonne in:

- Operazioni utente
- Variabili
- Attività trigger per Gerrit, Git e Docker

È possibile eseguire una ricerca con filtro basato su griglia nella pagina **Attività** per qualsiasi trigger.

Come funziona la ricerca

I criteri per la ricerca variano in base alla pagina in cui ci si trova. Ogni pagina prevede criteri di ricerca diversi.

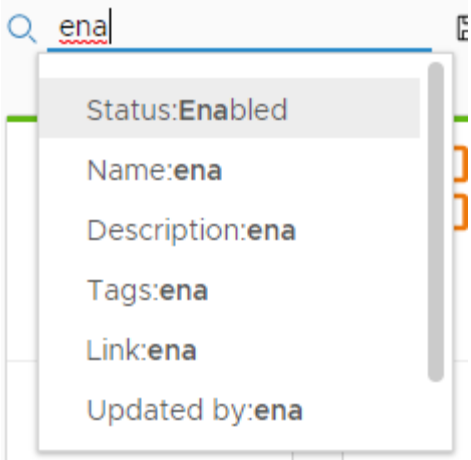
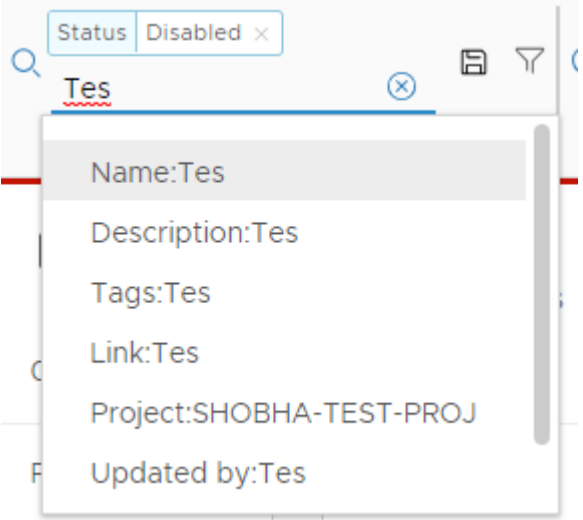
Dove si cerca	Criteri da utilizzare per la ricerca
Dashboard della pipeline	Progetto, Nome, Descrizione, Tag, Collegamento
Dashboard personalizzati	Progetto, Nome, Descrizione, Collegamento (UUID di un elemento nel dashboard)
Esecuzioni	Nome, Commenti, Motivo, Tag, Indice, Stato, Progetto, Mostra, Autore esecuzione, Eseguito da me, Collegamento (UUID dell'esecuzione) e Parametri di input, Parametri di output o Messaggio di stato utilizzando questo formato: <key>:<value>
Pipeline	Nome, Descrizione, Stato, Tag, Autore creazione, Creato da me, Autore aggiornamento, Aggiornato da me, Progetto
Progetti	Nome, Descrizione
Endpoint	Nome, Descrizione, Tipo, Aggiornato da, Progetto
Trigger Gerrit	Nome, Stato, Progetto
Server Gerrit	Nome, URL server, Progetto
Webhook Git	Nome, Tipo server, Repository, Ramo, Progetto

Dove:

- Il collegamento è l'UUID di una pipeline, di un'esecuzione o di un widget in un dashboard.
- La notazione e gli esempi di Parametri di input, Parametri di output e Messaggio di stato includono:
 - Notazione: `input.<inputKey>:<inputValue>`
Esempio: `input.GERRIT_CHANGE_OWNER_EMAIL:joe_user`
 - Notazione: `output.<outputKey>:<outputValue>`
Esempio: `output.BuildNo:29`
 - Notazione: `statusMessage:<value>`
Esempio: `statusMessage:Execution failed`

- Lo stato dipende dalla pagina di ricerca.
 - Per le esecuzioni, i valori possibili includono: completata, non riuscita, rollback non riuscito o annullata.
 - Per le pipeline, i valori di stato possibili includono: abilitato, disabilitato o rilasciato.
 - Per i trigger, i valori di stato possibili includono: abilitato o disabilitato.
- Eseguito, Creato o Aggiornato da me si riferisce all'utente connesso.

La ricerca viene visualizzata in alto a destra di ogni pagina valida. Quando si inizia a digitare nello spazio per la ricerca, Code Stream conoscendo il contesto della pagina, suggerisce le opzioni per la ricerca.

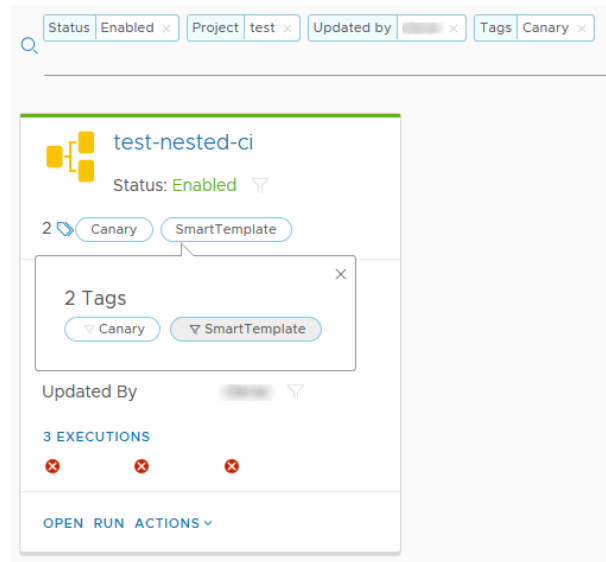
Metodi che è possibile utilizzare per la ricerca	Come immettere la stringa di ricerca
<p>Digitare una parte del parametro di ricerca.</p> <p>Ad esempio, digitare abi per aggiungere un filtro di stato che elenca tutte le pipeline abilitate.</p>	
<p>Per ridurre il numero di elementi trovati, aggiungere un filtro.</p> <p>Ad esempio, digitare Tes per aggiungere un filtro per il nome. Il filtro agisce come AND insieme al filtro Stato:disabilitato esistente per mostrare solo le pipeline disattivate il cui nome include Tes.</p> <p>Quando si aggiunge un altro filtro, vengono visualizzate le opzioni rimanenti: Nome, Descrizione, Tag, Collegamento, Progetto e Autore aggiornamento.</p>	

Metodi che è possibile utilizzare per la ricerca

Per ridurre il numero di voci visualizzate, fare clic sull'icona del filtro sulle proprietà di una pipeline o di un'esecuzione della pipeline.

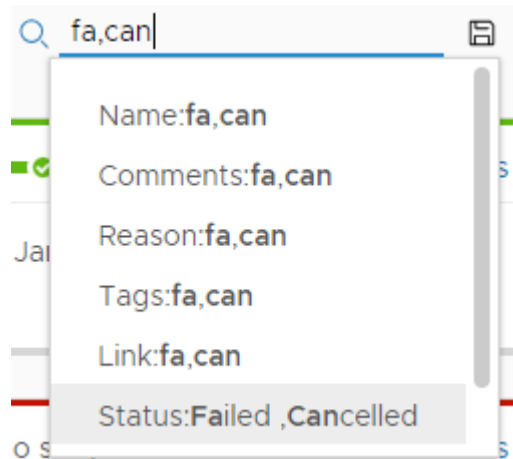
- Per le pipeline, **Stato**, **Tag**, **Progetto** e **Autore aggiornamento** sono tutti dotati di un'icona di filtro.
- Per le esecuzioni, **Tag**, **Autore esecuzione** e **Messaggio di stato** sono tutti dotati di un'icona di filtro.

Ad esempio, nella scheda della pipeline, fare clic sull'icona per aggiungere il filtro per il tag **SmartTemplate** ai filtri esistenti per: **Status:Enabled**, **Project:test**, **Updated by:user** e **Tags:Canary**.

Come immettere la stringa di ricerca

Utilizzare un separatore virgola per includere tutti gli elementi in due stati di esecuzione.

Ad esempio, digitare **fa,can** per creare un filtro di stato che funga da OR per elencare tutte le esecuzioni non riuscite o annullate.

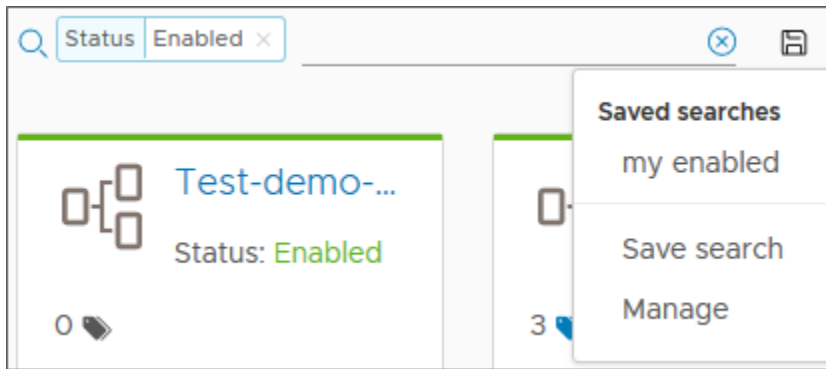


Metodi che è possibile utilizzare per la ricerca	Come immettere la stringa di ricerca
<p>Digitare un numero per includere tutti gli elementi all'interno di un intervallo di indice.</p> <p>Ad esempio, digitare 35 e selezionare < per elencare tutte le esecuzioni con un numero di indice inferiore a 35.</p>	
<p>Le pipeline modellate come attività diventano esecuzioni nidificate e non sono elencate con tutte le esecuzioni per impostazione predefinita.</p> <p>Per visualizzare le esecuzioni nidificate, digitare nested e selezionare il filtro Mostra.</p>	

Come salvare una ricerca preferita

È possibile salvare le ricerche preferite per utilizzarle in ogni pagina facendo clic sull'icona del disco accanto all'area di ricerca.

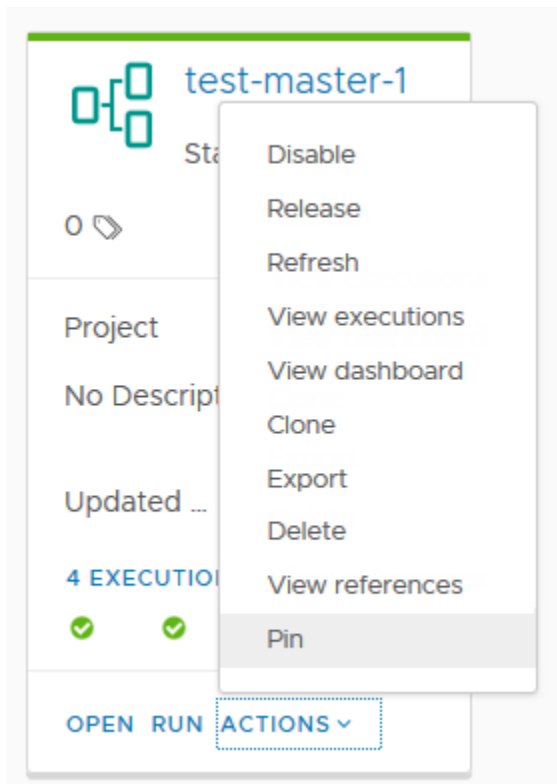
- È possibile salvare una ricerca digitando i parametri per la ricerca e facendo clic sull'icona per assegnare alla ricerca un nome, ad esempio **my enabled**.
- Dopo aver salvato una ricerca, fare clic sull'icona per accedervi. È inoltre possibile selezionare **Gestisci** per rinominare, eliminare o spostare la ricerca nell'elenco delle ricerche salvate.



Le ricerche sono associate al nome utente e vengono visualizzate solo nelle pagine per le quali viene applicata la ricerca. Ad esempio, se è stata salvata una ricerca denominata **my enabled** per **Stato:abilitato** nella pagina della pipeline, la ricerca **my enabled** non è disponibile nella pagina dei trigger Gerrit, anche se **Stato:abilitato** è una ricerca valida per un trigger.

È possibile salvare una pipeline preferita?

Se sul dashboard è presente una pipeline preferita, è possibile bloccarla in modo che compaia sempre in primo piano nella pagina delle pipeline o dei dashboard. Nella scheda della pipeline, fare clic su **Azioni > Blocca**.



Altre risorse per amministratori e sviluppatori di Code Stream

Sono disponibili ulteriori informazioni su Code Stream per amministratori e sviluppatori di Code Stream.

Tabella 9-1. Ulteriori risorse per gli amministratori

Per informazioni su...	Vedere queste risorse...
<p>Altri modi in cui gli amministratori possono utilizzare Code Stream:</p> <ul style="list-style-type: none"> ■ Configurare le pipeline per automatizzare il test e il rilascio delle applicazioni cloud native. ■ Automatizzare e testare il codice sorgente dello sviluppatore attraverso il testing, fino alla produzione. ■ Configurare le pipeline per permettere agli sviluppatori di testare le modifiche prima di confermarle nel ramo primario. ■ Tenere traccia delle metriche delle pipeline principali. 	<p>Code Stream</p> <ul style="list-style-type: none"> ■ Documentazione di vRealize Automation ■ Sito Web del prodotto vRealize Automation <p>Hands on di VMware</p> <ul style="list-style-type: none"> ■ Utilizzare la community di vRealize Automation. ■ Utilizzare la Learning zone di VMware. ■ Eseguire ricerche nei blog di VMware. ■ Provare gli Hands-on Lab VMware.

Tabella 9-2. Altre risorse per gli sviluppatori

Per informazioni su...	Vedere queste risorse...
<p>Altri modi in cui gli sviluppatori possono utilizzare Code Stream:</p> <ul style="list-style-type: none"> ■ Utilizzare le immagini del registro pubbliche e private per creare ambienti per applicazioni o servizi nuovi. ■ Configurare gli ambienti di sviluppo in modo che sia possibile creare rami dalla build stabile più recente. ■ Aggiornare gli ambienti di sviluppo con gli artefatti e le modifiche al codice più recenti. ■ Testare le modifiche al codice non confermate rispetto alle build stabili più recenti di altri servizi dipendenti. ■ Ricevere una notifica quando una modifica confermata in una pipeline CI/CD primaria interrompe altri servizi. 	<p>Code Stream</p> <ul style="list-style-type: none"> ■ Documentazione di vRealize Automation ■ Sito Web del prodotto vRealize Automation <p>Hands on di VMware</p> <ul style="list-style-type: none"> ■ Utilizzare la community di vRealize Automation. ■ Utilizzare la Learning zone di VMware. ■ Eseguire ricerche nei blog di VMware. ■ Provare gli Hands-on Lab VMware.