

Gebruik en beheer van vRealize Automation Code Stream

14 december 2022

vRealize Automation 8.7

U vindt de recentste technische documentatie op de website van VMware:

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

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

VMware Nederland B.V.
Key Office Papendorp
3e verdieping
Orteliuslaan 850
Utrecht
Nederland
Tel: +31 (0) 30-2849500
Fax: +31 (0) 30- 2849501
www.vmware.com/nl

Copyright © 2022 VMware, Inc. Alle rechten voorbehouden. [Informatie over copyright en handelsmerken.](#)

Inhoud

1	Wat is Code Stream en hoe werkt het?	5
2	Instellen om mijn releaseproces te modelleren	10
	Hoe voeg ik een project toe?	15
	Hoe beheer ik gebruikerstoegang en goedkeuringen?	16
	Wat zijn gebruikersbewerkingen en goedkeuringen?	25
3	Pijplijnen maken en gebruiken	27
	Hoe kan ik een pijplijn uitvoeren en de resultaten bekijken?	30
	Welke taaktypen zijn beschikbaar?	35
	Hoe gebruik ik variabele bindingen in pijplijnen?	41
	Hoe gebruik ik variabelebindingen in een voorwaardetaak om een pijplijn uit te voeren of te stoppen?	51
	Welke variabelen en expressies kan ik gebruiken bij het binden van pijplijntaken?	54
	Hoe verzend ik meldingen over mijn pijplijn?	72
	Hoe maak ik een JIRA-ticket wanneer een pijplijntaak mislukt?	75
	Hoe kan ik mijn implementatie terugdraaien?	77
4	Plannen om uw systeemeigen code te ontwikkelen, integreren en leveren	84
	De pijplijnwerkplek configureren	84
	Een CICD-systeemeigen build plannen voordat u de slimme pijplijnsjabloon gebruikt	88
	Een CI-systeemeigen build plannen voordat u de slimme pijplijnsjabloon gebruikt	96
	Een CD-systeemeigen build plannen voordat u de slimme pijplijnsjabloon gebruikt	97
	Een CICD-systeemeigen build plannen voordat u handmatig taken toevoegt	99
	Plannen voor terugdraaien	105
5	Tutorials	108
	Hoe kan ik code vanuit mijn GitHub-of GitLab-opslagplaats continu integreren in mijn pijplijn?	109
	Hoe kan ik de release van een applicatie die ik implementeer vanuit een YAML-cloudsjabloon automatiseren?	113
	Hoe kan ik de release van een applicatie op een Kubernetes-cluster automatiseren?	121
	Hoe implementeer ik mijn applicatie naar mijn Blue-Green-implementatie?	129
	Hoe integreer ik mijn eigen bouw-, test- en implementatietools?	134
	Hoe gebruik ik de resource-eigenschappen van een cloudsjabloontaak in mijn volgende taak	146
	Hoe gebruik ik een REST API om te integreren met andere applicaties?	150
	Hoe gebruik ik een pijplijn als code?	155

6 Verbinden met eindpunten 161

- Wat zijn eindpunten? 161
- Hoe integreer ik met Jenkins? 164
- Hoe integreer ik met Git? 171
- Hoe integreer ik met Gerrit? 173
- Hoe integreer ik met vRealize Orchestrator? 177

7 Pijplijnen activeren 183

- Hoe kan ik de Docker-trigger gebruiken om een continue leveringspijplijn uit te voeren? 183
- Hoe gebruik ik de Git-trigger om een pijplijn uit te voeren? 192
- Hoe gebruik ik de Gerrit-trigger om een pijplijn uit te voeren? 200

8 Pijplijnen bewaken 209

- Wat toont het pijplijndashboard mij? 209
- Hoe gebruik ik aangepaste dashboards om KPI's te volgen? 213

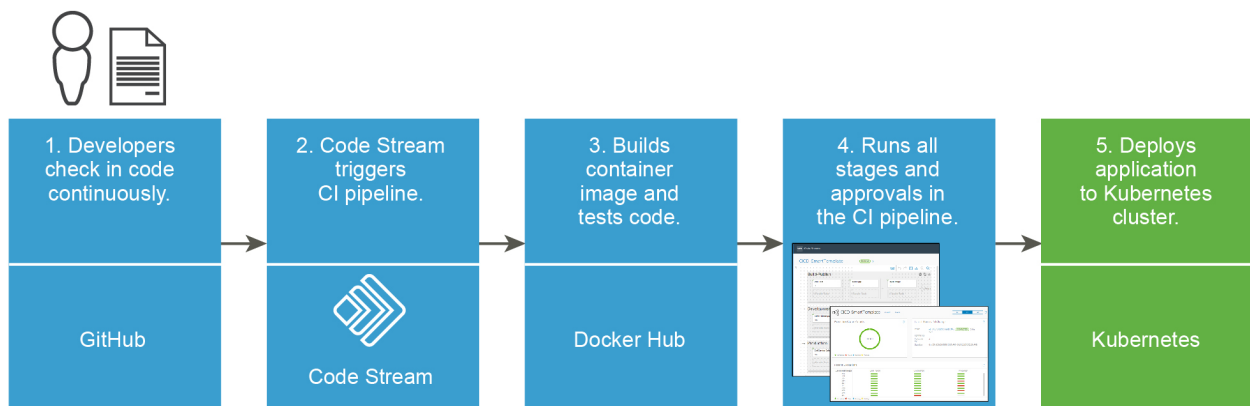
9 Meer informatie 215

- Wat is zoeken? 215
- Meer resources voor beheerders en ontwikkelaars 221

Wat is Code Stream en hoe werkt het?

1

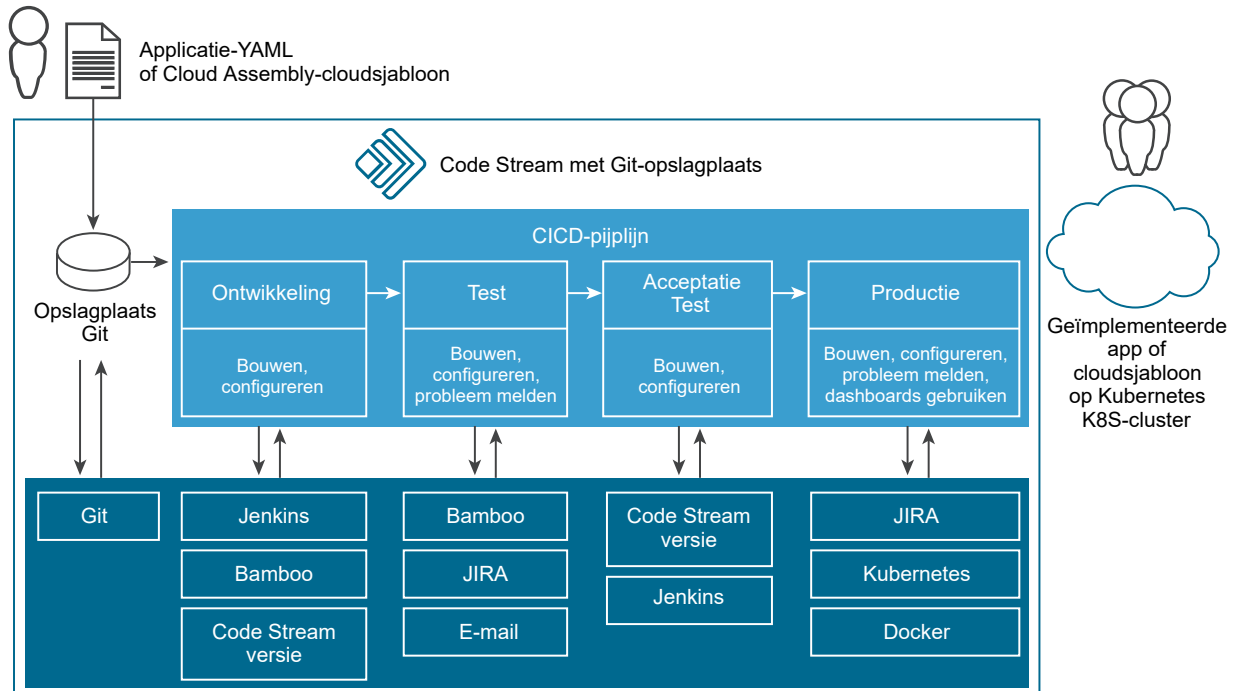
vRealize Automation Code Stream™ is een tool voor continue integratie en continue levering (CICL). Door pijplijnen te maken die het softwarereleaseproces in uw DevOps-levenscyclus modelleren, bouwt u de code-infrastructuur die uw software snel en continu levert.



Wanneer u Code Stream gebruikt om uw software te leveren, integreert u twee van de belangrijkste onderdelen van uw DevOps-levenscyclus: uw releaseproces en uw ontwikkelaarstools. Na de eerste instelling, die Code Stream met uw bestaande ontwikkelingstools integreert, automatiseren de pijplijnen uw volledige DevOps-levenscyclus.

Vanaf vRealize Automation 8.2 worden blueprints VMware Cloud Templates genoemd.

U maakt een pijplijn die uw software bouwt, test en vrijgeeft. Code Stream gebruikt die pijplijn om uw software uit te werken van de opslagplaats van de broncode tot de productie via testen.



U vindt meer informatie over het plannen van uw continue integratie en pijplijnen voor continue levering bij [Hoofdstuk 4 Plannen om op een systeemeigen manier uw code te bouwen, integreren en leveren in Code Stream](#).

Hoe Code Stream-beheerders Code Stream gebruiken

Als beheerder maakt u eindpunten en zorgt u ervoor dat er werkende instanties beschikbaar zijn voor ontwikkelaars. U kunt pijplijnen maken, activeren en beheren, en meer. U heeft de rol **Administrator**, zoals beschreven in [Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream?](#).

Tabel 1-1. Hoe Code Stream-beheerders ontwikkelaars ondersteunen

Om ontwikkelaars te ondersteunen...	Wat u kunt doen...
Omgevingen bieden en beheren.	<p>Maak omgevingen voor ontwikkelaars om hun code te testen en te implementeren.</p> <ul style="list-style-type: none"> ■ Volg de status en verzend e-mailmeldingen. ■ Houd uw ontwikkelaars productief door ervoor te zorgen dat hun omgevingen continu werken. <p>Zie Meer resources voor Code Stream-beheerders en -ontwikkelaars voor meer informatie.</p> <p>Zie ook Hoofdstuk 5 Tutorials voor het gebruik van Code Stream.</p>
Geef eindpunten op.	Zorg ervoor dat ontwikkelaars beschikken over werkende instanties van eindpunten die verbinding kunnen maken met hun pijplijnen.
Verstrek integraties met andere services.	<p>Zorg ervoor dat integraties met andere services werken.</p> <p>Raadpleeg VMware Cloud Services-documentatie voor meer informatie.</p>

Tabel 1-1. Hoe Code Stream-beheerders ontwikkelaars ondersteunen (vervolg)

Om ontwikkelaars te ondersteunen...	Wat u kunt doen...
Pijplijnen maken.	<p>Maak pijplijnen die releaseprocessen modelleren.</p> <p>Zie Hoofdstuk 3 Pijplijnen maken en gebruiken in Code Stream voor meer informatie.</p>
Activeer pijplijnen.	<p>Zorg ervoor dat pijplijnen worden uitgevoerd wanneer er gebeurtenissen plaatsvinden.</p> <ul style="list-style-type: none"> ■ Gebruik de Docker-trigger om een standalone pijplijn voor continue levering (CD) te activeren wanneer een build-artefact wordt gemaakt of bijgewerkt. ■ Gebruik de Git-trigger om een pijplijn te activeren wanneer een ontwikkelaar wijzigingen in de code doorvoert. ■ Gebruik de Gerrit-trigger om een pijplijn te activeren wanneer ontwikkelaars code beoordelen, samenvoegen en meer. ■ Gebruik de Docker-trigger om een standalone pijplijn voor continue levering (CD) te activeren wanneer een build-artefact wordt gemaakt of bijgewerkt. <p>Zie Hoofdstuk 7 Pijplijnen in Code Stream activeren voor meer informatie.</p>
Beheer pijplijnen en goedkeuringen.	<p>Blijf op de hoogte van pijplijnen.</p> <ul style="list-style-type: none"> ■ Bekijk de pijplijnstatus en bekijk wie de pijplijnen heeft uitgevoerd. ■ Bekijk goedkeuringen voor de uitvoering van pijplijnen en beheer goedkeuringen voor actieve en inactieve pijplijn-uitvoeringen. <p>Zie Wat zijn gebruikersbewerkingen en goedkeuringen in Code Stream? voor meer informatie.</p> <p>Zie ook Hoe kan ik aangepaste dashboards gebruiken om KPI's voor mijn pijplijn in Code Stream te volgen?.</p>
Bewaak ontwikkelaarsomgevingen.	<p>Maak custom dashboards die de pijplijnstatus, trends, statistieken en belangrijke indicatoren controleren. Gebruik de custom dashboards om pijplijnen te bewaken die slagen of mislukken in ontwikkelaarsomgevingen. Identificeer en rapporteer over resources die niet genoeg worden gebruikt en maak resources vrij.</p> <p>U kunt ook het volgende zien:</p> <ul style="list-style-type: none"> ■ Hoe lang een pijplijn is uitgevoerd voordat deze is gelukt. ■ Hoe lang een pijplijn heeft gewacht op goedkeuring, en u kunt de gebruiker die deze moet goedkeuren op de hoogte stellen. ■ Fasen en taken die het vaakst mislukken. ■ Fasen en taken die de meeste tijd nodig hebben om te worden uitgevoerd. ■ Releases waarmee de ontwikkelingsteams bezig zijn. ■ Applicaties die met succes zijn geïmplementeerd en vrijgegeven. <p>Zie Hoofdstuk 8 Pijplijnen bewaken in Code Stream voor meer informatie.</p>
Problemen oplossen.	<p>Problemen met de pijplijn oplossen in ontwikkelaarsomgevingen.</p> <ul style="list-style-type: none"> ■ Identificeer problemen en los ze op in omgevingen met continue integratie en continue levering (CICD). ■ Gebruik de pijplijndashboards en maak custom dashboards om meer te zien. Zie Hoofdstuk 8 Pijplijnen bewaken in Code Stream. <p>Zie ook Hoofdstuk 2 Code Stream instellen om mijn releaseproces te modelleren.</p>

Code Stream is onderdeel van VMware Cloud Services.

- Gebruik Cloud Assembly om cloudsjablonen te implementeren.
- Gebruik Service Broker om cloudsjablonen op te halen uit de catalogus.

Om meer te weten over andere dingen die u kunt doen, raadpleegt u [Documentatie voor VMware vRealize Automation](#).

Hoe ontwikkelaars Code Stream gebruiken

Als ontwikkelaar gebruikt u Code Stream om pijplijnen te maken en uit te voeren en pijplijnactiviteiten op de dashboards in de gaten te houden. U heeft de rol `User`, zoals beschreven in [Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream?](#).

Nadat u een pijplijn hebt uitgevoerd, moet u weten:


- Of uw code in alle fasen van de pijplijn is geslaagd. Bekijk de resultaten in de pijplijnnitvoeringen voor meer informatie.
- Wat moet ik doen als de pijplijn is mislukt en wat heeft de fout veroorzaakt. Bekijk de belangrijkste fouten in de pijplijndashboards voor meer informatie.

Tabel 1-2. Ontwikkelaars die Code Stream gebruiken

Om uw code te integreren en vrij te geven	Doet u het volgende
Bouw pijplijnen.	Test en implementeer uw code. Werk uw code bij als een pijplijn mislukt.
Verbind uw pijplijn met eindpunten.	Verbind de taken in uw pijplijn met eindpunten, zoals een GitHub-opslagplaats.
Voer pijplijnen uit.	Voeg een gebruikersbewerkingstaak voor goedkeuring toe zodat een andere gebruiker uw pijplijn op specifieke momenten kan goedkeuren.
Dashboards bekijken.	Bekijk de resultaten op het pijplijndashboard. U kunt de trends, geschiedenis, fouten en nog veel meer zien.

Zie [Aan de slag met VMware Code Stream](#) voor meer informatie over aan de slag gaan.

Meer documentatie kunt u vinden in het paneel Ondersteuning in het product

Als u de informatie die u nodig heeft hier niet vindt, kunt u meer hulp krijgen in het product. 

- Klik op de wegwijzers en de knopinfo in de gebruikersinterface om de contextafhankelijke informatie die u nodig heeft te krijgen waar en wanneer u deze nodig heeft.
- Open het paneel voor ondersteuning in het product en lees de onderwerpen die worden weergegeven op de pagina actieve gebruikersinterface. U kunt ook in het paneel zoeken om antwoorden op vragen te krijgen.

Meer over webhooks

U kunt meerdere webhooks voor verschillende takken maken door hetzelfde Git-eindpunt te gebruiken en verschillende waarden voor de naam van de tak op te geven op de pagina voor webhookconfiguratie. Als u een andere webhook voor een andere tak in dezelfde Git-opslagplaats wilt maken, hoeft u het Git-eindpunt niet meerdere keren te klonen voor meerdere takken. In plaats daarvan geeft u de naam van de tak op in de webhook, zodat u het Git-eindpunt opnieuw kunt gebruiken. Als de tak op de Git-webhook hetzelfde is als de tak op het eindpunt, hoeft u geen taknaam op de pagina van de Git-webhook op te geven.

Code Stream instellen om mijn releaseproces te modelleren

2

Als u uw releaseproces wilt modelleren, maakt u een pijplijn die de fasen, taken en goedkeuringen weergeeft die u normaal gebruikt voor het vrijgeven van uw software. Code Stream automatiseert vervolgens het proces dat de code bouwt, test, goedkeurt en implementeert.

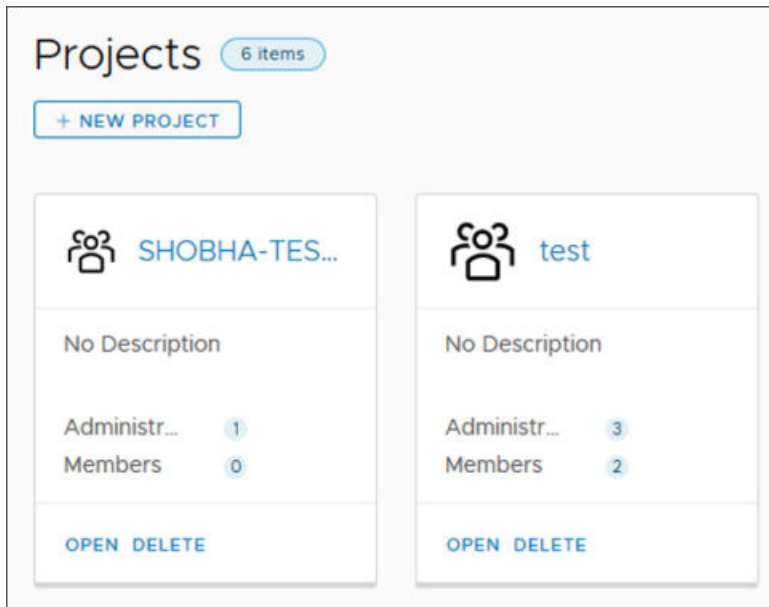
Nu u alles klaar hebt voor het modelleren van uw software-releaseproces, doet u dat in Code Stream als volgt.

Voorwaarden

- Controleer of er al eindpunten beschikbaar zijn. Klik in Code Stream op **Eindpunten**.
- Ontdek meer over systeemeigen manieren waarmee u uw code kunt bouwen en implementeren. Zie [Hoofdstuk 4 Plannen om op een systeemeigen manier uw code te bouwen, integreren en leveren in Code Stream](#).
- Bepaal of bepaalde resources die u in uw pijplijn wilt gebruiken, als beperkt moeten worden gemarkeerd. Zie [Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream?](#).
- Als u de rol van gebruiker of kijker hebt in plaats van de rol van beheerder, bepaalt u wie de beheerder is voor uw Code Stream-instantie.

Procedure

- 1 Bekijk de beschikbare projecten in Code Stream en selecteer er een dat geschikt is voor u.
 - Als er geen projecten worden weergegeven, vraagt u het aan een Code Stream-beheerder die een project kan maken en u lid kan maken van het project. Zie [Hoe voeg ik een project toe in Code Stream?](#).
 - Als u geen lid bent van een van de projecten in de lijst, vraagt u het aan een Code Stream-beheerder die u als lid van een project kan toevoegen.



- 2 Voeg alle nieuwe eindpunten toe die u nodig heeft voor uw pijplijn.

Mogelijk heeft u bijvoorbeeld Git, Jenkins, Code Stream Build, Kubernetes en JIRA nodig.

- 3 Maak variabelen zodat u de waarden in uw pijplijntaken kunt hergebruiken.

Om de resources te beperken die in uw pijplijnen worden gebruikt, gebruikt u beperkte variabelen, zoals een hostmachine. U kunt voorkomen dat de pijplijn wordt uitgevoerd totdat een andere gebruiker deze expliciet goedkeurt.

Beheerders kunnen geheime variabelen en beperkte variabelen maken. Gebruikers kunnen geheime variabelen maken.

U kunt een variabele zo vaak opnieuw gebruiken als u wilt in meerdere pijplijnen. Een variabele die een hostmachine definieert, kan bijvoorbeeld `HostIPAddress` zijn. Als u de variabele wilt gebruiken in een pijplijntaak, voert u `${var.HostIPAddress}` in.

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

- 4 Als u een beheerder bent, markeert u alle eindpunten en variabelen die cruciaal zijn voor uw bedrijf als beperkte resources.

Wanneer een gebruiker die geen beheerder is een pijplijn probeert uit te voeren die een beperkte resource bevat, stopt de pijplijn bij de taak die de beperkte resource gebruikt. Vervolgens moet een beheerder de pijplijn hervatten.

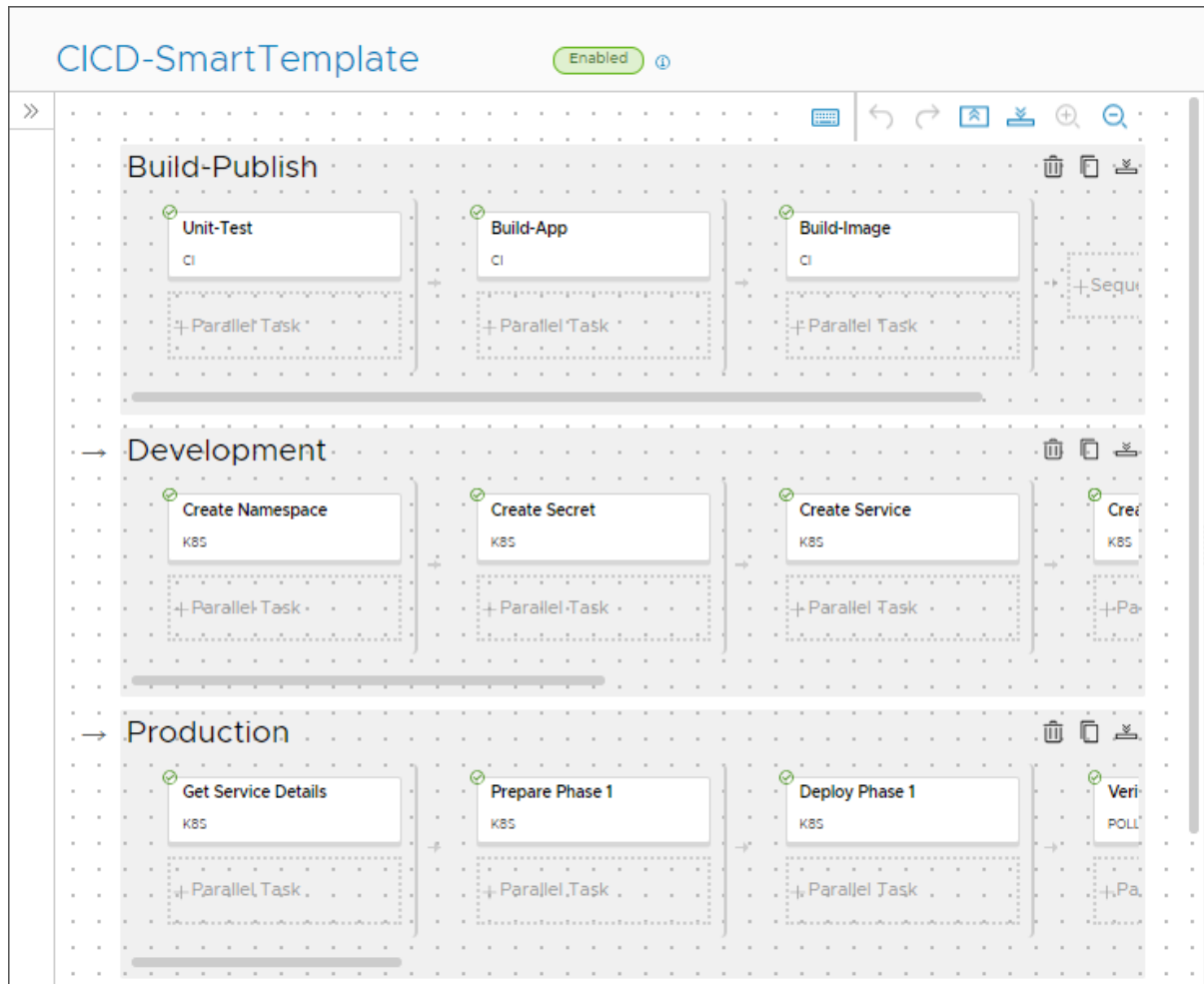
- 5 Plan de bouwstrategie voor uw systeemeigen CICD-, CI- of CD-pijplijn.

Voordat u een pijplijn maakt die uw code continu integreert (CI) en continu implementeert (CD), moet u uw bouwstrategie plannen. Met het bouwplan kunt u bepalen wat Code Stream nodig heeft, zodat het systeem uw code op een systeemeigen manier kan bouwen, integreren, testen en implementeren.

Een systeemeigen Code Stream-build maken	Resulteert in deze bouwstrategie
Gebruik een van de slimme pijplijnsjablonen.	<ul style="list-style-type: none"> ■ Bouwt alle fasen en taken voor u. ■ Kloont de opslagplaats van de bron. ■ Bouw en test uw code. ■ Plaatst uw code in een container voor implementatie. ■ Vult de taakstappen voor de pijplijn in op basis van uw selecties.
Voeg fasen en taken handmatig toe.	U voegt fasen en taken toe en voert de informatie in om deze in te vullen.

- 6 Maak uw pijplijn met behulp van een slimme pijplijnsjabloon of door handmatig fasen en taken toe te voegen aan de pijplijn.

Vervolgens markeert u resources als beperkt. Voeg indien nodig goedkeuringen toe. Pas normale, beperkte of geheime variabelen toe. Voeg bindingen tussen taken toe.











- 7 Valideer uw pijplijn, schakel deze in en voer deze uit.

8 Bekijk de uitvoeringen van de pijplijn.

Executions 280 items

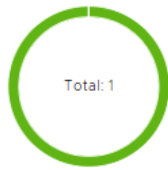
[NEW EXECUTION](#) Search Filter Refresh

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

9 Om de status en KPI's te volgen, gebruikt u de pijplijndashboards en maakt u eventueel aangepaste dashboards.

CICD-SmartTemplate [CLONE](#) [BACK](#) 1D 7D 14D Refresh


Execution Status Counts Refresh



Total: 1

● Completed ● Failed ● Running ● Waiting

Latest Successful Change Refresh































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 Refresh

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

● Completed ● Failed ● Running ● Waiting

Resultaten

U heeft een pijplijn gemaakt die u in het geselecteerde project kunt gebruiken.

U kunt ook de pijplijn-YAML exporteren en deze vervolgens importeren en opnieuw gebruiken in andere projecten.

Wat nu te doen

Meer informatie over toepassingsvoorbeelden die u mogelijk wilt toepassen in uw omgeving. Zie [Hoofdstuk 5 Tutorials voor het gebruik van Code Stream](#).

Hoe voeg ik een project toe in Code Stream?

U maakt een project en voegt er beheerders en leden aan toe. Projectleden kunnen functies gebruiken zoals het maken van een pijplijn en het toevoegen van een eindpunt. Om een project voor een ontwikkelingsteam te maken, te verwijderen of bij te werken, moet u een Code Stream-beheerder zijn.

Er moet een project zijn voordat u een pijplijn kunt maken. Wanneer u een pijplijn maakt, selecteert u een project dat al uw pijplijngegevens groepeerd. Definities voor eindpunten en variabelen zijn ook afhankelijk van een bestaand project.

Voorwaarden

- Controleer of u de rol van Code Stream-beheerder heeft. Zie [Wat zijn rollen in Code Stream?](#).
Als u niet de rol van Code Stream-beheerder heeft, maar u een beheerder bent in Cloud Assembly, kunt u de Cloud Assembly-UI gebruiken om projecten te maken, bij te werken of te verwijderen. Zie [Hoe voeg ik een project toe voor mijn Cloud Assembly-ontwikkelingsteam](#).
- Als u Active Directory-groepen toevoegt aan projecten, controleert u of u Active Directory-groepen voor uw organisatie hebt geconfigureerd. Zie [Hoe activeer ik Active Directory-groepen in vRealize Automation voor projecten?](#) Als de groepen niet zijn gesynchroniseerd, zijn ze niet beschikbaar wanneer u ze probeert toe te voegen aan een project.

Procedure

- 1 Selecteer **Projecten** en klik op **Nieuw project**.
- 2 Voer de projectnaam in.
- 3 Klik op **Maken**.
- 4 Selecteer de kaart voor het nieuw gemaakte project en klik op **Openen**.
- 5 Klik op het tabblad **Gebruikers**, voeg gebruikers toe en wijs rollen toe.
 - De projectbeheerder kan leden toevoegen.
 - Het projectlid met een servicerol kan services gebruiken.
 - De projectkijker kan projecten zien, maar kan deze niet maken, bijwerken of verwijderen.

Zie [Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream?](#) voor meer informatie over projectrollen.

6 Klik op **Opslaan**.

Wat nu te doen

Voeg eindpunten en pijplijnen toe die het project gebruiken. Zie [Hoofdstuk 6 Code Stream verbinden met eindpunten](#) en [Hoofdstuk 3 Pijplijnen maken en gebruiken in Code Stream](#).

Nadat u een pijplijn hebt gemaakt, wordt de naam van het project dat al uw pijplijngegevens groepeerst, weergegeven op pijplijnkaarten en kaarten voor pijplijnuitvoering.

Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream?

Code Stream biedt verschillende manieren om ervoor te zorgen dat gebruikers over de juiste autorisatie en toestemming beschikken om met pijplijnen te werken die uw softwareapplicaties vrijgeven.

Elk teamlid heeft een toegewezen rol, waarmee specifieke rechten voor-pijplijnen, eindpunten en dashboards worden geboden, en de mogelijkheid om resources als beperkt te markeren.

Met gebruikersbewerkingen en goedkeuringen kunt u bepalen wanneer een pijplijn wordt uitgevoerd en wanneer deze moet stoppen voor een goedkeuring. Uw rol bepaalt of u een pijplijn kunt hervatten en of u pijplijnen kunt uitvoeren die beperkte eindpunten of variabelen bevatten.

Gebruik geheime variabelen om gevoelige informatie te verbergen en te versleutelen. Gebruik een beperkte variabele voor tekenreeksen, wachtwoorden en URL's die moeten worden verborgen en versleuteld, en om het gebruik in uitvoeringen te beperken. Gebruik bijvoorbeeld een geheime variabele voor een wachtwoord of URL. U kunt geheime en beperkte variabelen gebruiken in elk type taak in uw pijplijn.

Wat zijn rollen in Code Stream?

Afhankelijk van uw rol in Code Stream kunt u bepaalde acties uitvoeren en toegang krijgen tot bepaalde gebieden. Uw rol kan u bijvoorbeeld toestaan om pijplijnen te maken, bij te werken en uit te voeren. Of u bent mogelijk alleen gerechtigd om pijplijnen te bekijken.

`Alle acties behalve beperkte` betekent dat deze rol toestemming heeft om de acties maken, lezen, bijwerken en verwijderen uit te voeren op entiteiten, behalve voor beperkte variabelen en eindpunten.

Tabel 2-1. Toegangsrechten op service- en projectniveau in Code Stream

Code Stream-rollen					
Toegangsniveau's	Code Stream-beheerder	Code Stream-ontwikkelaar	Code Stream-uitvoerder	Code Stream-lezer	Code Stream-gebruiker
Code Stream-toegang op serviceniveau	Alle acties	Alle acties behalve beperkte	Uitvoeringsacties	Alleen-lezen	Geen
Toegang op projectniveau: Projectbeheerder	Alle acties	Alle acties	Alle acties	Alle acties	Alle acties
Toegang op projectniveau: Lid project	Alle acties	Alle acties behalve beperkte	Alle acties behalve beperkte	Alle acties behalve beperkte	Alle acties behalve beperkte
Toegang op projectniveau: Kijker project	Alle acties	Alle acties behalve beperkte	Uitvoeringsacties	Alleen-lezen	Alleen-lezen

Gebruikers met de rol Projectbeheerder kunnen alle acties uitvoeren op projecten waarvoor zij projectbeheerder zijn.

Een projectbeheerder kan pijplijnen, variabelen, eindpunten, dashboards en triggers maken, lezen, bijwerken en verwijderen, en een pijplijn starten die beperkte eindpunten of variabelen bevat, als deze resources zich in het project bevinden waarin de gebruiker een projectbeheerder is.

Gebruikers met de rol van Servicekijker kunnen alle informatie zien die beschikbaar is voor de beheerder. Ze kunnen geen acties uitvoeren tenzij een beheerder ze projectbeheerder of projectlid maakt. Als de gebruiker is gekoppeld aan een project, hebben ze de rechten voor de rol. De projectkijker breidt hun rechten niet uit zoals de rol van beheerder of lid dit doet. Deze rol is alleen-lezen in alle projecten.

Als u leesrechten in een project hebt, kunt u nog steeds beperkte resources zien.

- Om beperkte eindpunten te zien, waarvoor een vergrendelingspictogram op de eindpuntkaart wordt weergegeven, klikt u op **Configureren > Eindpunten**.
- Als u beperkte en geheime variabelen wilt zien, waarvoor BEPERKT of GEHEIM in de kolom **type** wordt weergegeven, klikt u op **Configureren > Variabelen**.

Tabel 2-2. Mogelijkheden van Code Stream-servicerol

UI-context	Capaciteiten	Code Stream-beheerdersrol	Code Stream-ontwikkelaarsrol	Code Stream-uitvoerdersrol	Code Stream-kijkersrol	Code Stream-gebruikersrol
Pijplijnen						
	Pijplijnen weergeven	Ja	Ja	Ja	Ja	

Tabel 2-2. Mogelijkheden van Code Stream-servicerol (vervolg)

UI-context	Capaciteiten	Code Stream-beheerdersrol	Code Stream-ontwikkelaarsrol	Code Stream-uitvoerdersrol	Code Stream-kijkersrol	Code Stream-gebruikersrol
	Pijplijnen maken	Ja	Ja			
	Pijplijnen uitvoeren	Ja	Ja	Ja		
	Pijplijnen uitvoeren die beperkte eindpunten of variabelen bevatten	Ja				
	Pijplijnen bijwerken	Ja	Ja			
	Pijplijnen verwijderen	Ja	Ja			
Pijplijnuitvoeringen						
	Pijplijnuitvoeringen weergeven	Ja	Ja	Ja	Ja	
	De pijplijnuitvoeringen hervatten, pauzeren en annuleren	Ja	Ja	Ja		
	Pijplijnen hervatten die stoppen voor goedkeuring bij beperkte resources	Ja				
Aangepaste integraties						
	Aangepaste integraties maken	Ja	Ja			
	Aangepaste integraties lezen	Ja	Ja	Ja	Ja	
	Aangepaste integraties bijwerken	Ja	Ja			
Eindpunten						

Tabel 2-2. Mogelijkheden van Code Stream-servicerol (vervolg)

UI-context	Capaciteiten	Code Stream-beheerdersrol	Code Stream-ontwikkelaarsrol	Code Stream-uitvoerdersrol	Code Stream-kijkersrol	Code Stream-gebruikersrol
	Uitvoeringen weergeven	Ja	Ja	Ja	Ja	
	Uitvoeringen maken	Ja	Ja			
	Uitvoeringen bijwerken	Ja	Ja			
	Uitvoeringen verwijderen	Ja	Ja			
Resources als beperkt markeren						
	Een eindpunt of variabele als beperkt markeren	Ja				
Dashboards						
	Dashboards weergeven	Ja	Ja	Ja	Ja	
	Dashboards maken	Ja	Ja			
	Dashboards bijwerken	Ja	Ja			
	Dashboards verwijderen	Ja	Ja			

Aangepaste rollen en rechten in Code Stream

U kunt aangepaste rollen in Cloud Assembly maken die rechten uitbreiden naar gebruikers die met pijplijnen werken. Wanneer u een aangepaste rol voor Code Stream-pijplijnen maakt, selecteert u een of meer rechten voor **Pijplijn**.

Selecteer het minimale aantal rechten voor **Pijplijn** die vereist zijn voor gebruikers aan wie deze aangepaste rol wordt toegewezen.

Wanneer een gebruiker aan een project is toegewezen en een rol in dat project heeft gekregen en aan die gebruiker een aangepaste rol is toegewezen met een of meer rechten voor **Pijplijn**, kan deze gebruiker alle acties uitvoeren die door de rechten worden toegestaan. De gebruiker kan bijvoorbeeld beperkte variabelen maken, beperkte pijplijnen beheren, aangepaste integraties maken en beheren, en nog veel meer.

Tabel 2-3. Pijplijnrechten die u kunt toewijzen aan aangepaste rollen

Pijplijnrecht	Code Stream-beheerder	Code Stream-ontwikkelaar	Code Stream-uitvoerder	Code Stream-lezer	Code Stream-gebruiker	Projectbeheerder	Projectleden	Projectkijker
Pijplijnen beheren	Ja	Ja				Ja	Ja	
Beperkte pijplijnen beheren	Ja					Ja		
Custom integraties beheren	Ja	Ja						
Pijplijnen uitvoeren	Ja	Ja	Ja			Ja	Ja	
Beperkte pijplijnen uitvoeren	Ja					Ja		
Uitvoeringen beheren	Ja					Ja		
Lezen. Dit recht is niet zichtbaar.	Ja	Ja	Ja	Ja		Ja	Ja	Ja

Tabel 2-4. Hoe u pijplijnrechten kunt gebruiken met aangepaste rollen

Recht	Wat u kunt doen
Pijplijnen beheren	<ul style="list-style-type: none"> ■ Pijplijnen maken, bijwerken, verwijderen en klonen. ■ Pijplijnen vrijgeven aan VMware Service Broker en de vrijage ongedaan maken. ■ Eindpunten maken, bijwerken en verwijderen. ■ Regelmatige en geheime variabelen maken, bijwerken en verwijderen. ■ Een Gerrit-listener maken, klonen, bijwerken en verwijderen. ■ Verbinding maken met een Gerrit-listener en de verbinding verbreken. ■ Een Gerrit-trigger maken, klonen, bijwerken en verwijderen. ■ Een Git-webhook maken, bijwerken en verwijderen. ■ Een Docker-webhook maken, bijwerken en verwijderen. ■ Slimme pijplijnsjablonen gebruiken om pijplijnen te maken. ■ Pijplijnen uit YAML importeren en exporteren in YAML. ■ Aangepaste dashboards maken, bijwerken en verwijderen. ■ Alle aangepaste integraties lezen. ■ Alle beperkte eindpunten en variabelen lezen, maar waarden kunnen niet worden weergegeven.
Beperkte pijplijnen beheren	<ul style="list-style-type: none"> ■ Eindpunten maken, bijwerken en verwijderen. ■ Eindpunten markeren als beperkt, beperkte eindpunten bijwerken en deze verwijderen. ■ Regelmatige en geheime variabelen maken, bijwerken en verwijderen. ■ Beperkte variabelen maken, bijwerken en verwijderen. ■ Alle rechten die u kunt uitvoeren met behulp van Pijplijnen beheren.
Custom integraties beheren	<ul style="list-style-type: none"> ■ Aangepaste integraties maken en bijwerken. ■ Versies van aangepaste integraties beheren en deze vrijgeven. ■ Aangepaste integratieversies verwijderen en afschaffen. ■ Aangepaste integraties verwijderen.
Pijplijnen uitvoeren	<ul style="list-style-type: none"> ■ Voer pijplijnen uit. ■ Uitvoeringen van pijplijnen onderbreken, hervatten en annuleren. ■ Pijplijnuitvoeringen opnieuw uitvoeren. ■ Een Gerrit-triggergebeurtenis hervatten, opnieuw uitvoeren en handmatig activeren. ■ Een gebruikersbewerking goedkeuren en batchgoedkeuringen van gebruikersbewerkingen uitvoeren.

Tabel 2-4. Hoe u pijplijnrechten kunt gebruiken met aangepaste rollen (vervolg)

Recht	Wat u kunt doen
Beperkte pijplijnen uitvoeren	<ul style="list-style-type: none"> ■ Voer pijplijnen uit. ■ Uitvoeringen van pijplijnen onderbreken, hervatten, annuleren en verwijderen. ■ Pijplijnuitvoeringen opnieuw uitvoeren. ■ Een actieve pijplijnuitvoering synchroniseren. ■ Een actieve pijplijnuitvoering gedwongen verwijderen. ■ Een Gerrit-triggergebeurtenis hervatten, opnieuw uitvoeren, verwijderen en handmatig activeren. ■ Beperkte items verhelpen en doorgaan met de uitvoering van de pijplijn. ■ Gebruikerscontext uitschakelen en doorgaan met de pijplijnuitvoering na de goedkeuring van een gebruikersbewerkingstaak. ■ Alle rechten die u kunt uitvoeren met Pijplijnen uitvoeren.
Uitvoeringen beheren	<ul style="list-style-type: none"> ■ Voer pijplijnen uit. ■ Uitvoeringen van pijplijnen onderbreken, hervatten, annuleren en verwijderen. ■ Pijplijnuitvoeringen opnieuw uitvoeren. ■ Een Gerrit-triggergebeurtenis hervatten, opnieuw uitvoeren, verwijderen en handmatig activeren. ■ Alle rechten die u kunt uitvoeren met Pijplijnen uitvoeren.

Aangepaste rollen kunnen combinaties van rechten bevatten. Deze rechten zijn onderverdeeld in groepen van mogelijkheden waarmee gebruikers pijplijnen kunnen beheren of uitvoeren, met en zonder beperkte resources. Deze machtigingen staan voor alle mogelijkheden die elke rol kan uitvoeren in Code Stream.

Als u bijvoorbeeld een aangepaste rol maakt en het recht **Beperkte pijplijnen beheren** opneemt, kunnen gebruikers met de ontwikkelaarsrol in Code Stream:

- Eindpunten maken, bijwerken en verwijderen.
- Eindpunten markeren als beperkt, beperkte eindpunten bijwerken en deze verwijderen.
- Regelmatige en geheime variabelen maken, bijwerken en verwijderen.
- Beperkte variabelen maken, bijwerken en verwijderen.

Tabel 2-5. Voorbeelden van combinaties van pijplijnrechten in aangepaste rollen

Aantal rechten toegewezen aan aangepaste rol	Voorbeelden van gecombineerde rechten	Deze combinatie gebruiken
Eén recht	Pijplijnen uitvoeren	
Twee rechten	Pijplijnen beheren en Pijplijnen uitvoeren	
Drie rechten	Pijplijnen beheren en Pijplijnen uitvoeren en Beperkte pijplijnen uitvoeren	
	Pijplijnen beheren en Aangepaste integraties beheren en Beperkte pijplijnen uitvoeren	Deze combinatie kan van toepassing zijn op een Code Stream-ontwikkelaarsrol, maar is beperkt tot de projecten waarvan de gebruiker lid is.

Tabel 2-5. Voorbeelden van combinaties van pijplijnrechten in aangepaste rollen (vervolg)

Aantal rechten toegewezen aan aangepaste rol	Voorbeelden van gecombineerde rechten	Deze combinatie gebruiken
	Pijplijnen beheren en Aangepaste integraties beheren en Uitvoeringen beheren	Deze combinatie kan van toepassing zijn op een Code Stream-beheerder, maar is beperkt tot de projecten waarvan de gebruiker lid is.
	Pijplijnen beheren, Beperkte pijplijnen beheren en Aangepaste integraties beheren	Met deze combinatie heeft een gebruiker volledige rechten en kan deze alles in Code Stream maken en verwijderen.

Als u de rol van beheerder heeft

Als beheerder kunt u aangepaste integraties, eindpunten, variabelen, triggers, pijplijnen en dashboards maken.

Met projecten kunnen pijplijnen toegang krijgen tot infrastructuurresources. Beheerders maken projecten zodat gebruikers pijplijnen, eindpunten en dashboards samen kunnen groeperen. Gebruikers selecteren vervolgens het project in hun pijplijnen. Elk project omvat een beheerder en gebruikers met toegewezen rollen.

Als u de beheerdersrol hebt, kunt u eindpunten en variabelen als beperkte resources markeren en kunt u pijplijnen uitvoeren die gebruikmaken van beperkte resources. Als een gebruiker die geen beheerder is, de pijplijn uitvoert die een beperkt eindpunt of beperkte variabele bevat, stopt de pijplijn bij de taak waar de beperkte variabele wordt gebruikt en moet een beheerder de pijplijn hervatten.

Als beheerder kunt u ook vragen dat pijplijnen worden gepubliceerd in vRealize Automation Service Broker.

Als u de rol van ontwikkelaar heeft

U kunt werken met pijplijnen zoals een beheerder dat kan, behalve dat u niet kunt werken met beperkte eindpunten of variabelen.

Als u een pijplijn uitvoert die beperkte eindpunten of variabelen gebruikt, wordt de pijplijn alleen uitgevoerd tot aan de taak die de beperkte resource gebruikt. Vervolgens wordt het systeem gestopt en moet een Code Stream-beheerder of projectbeheerder de pijplijn hervatten.

Als u de rol van Gebruiker heeft

U heeft toegang tot Code Stream, maar heeft niet dezelfde rechten als de andere rollen.

Als u de rol van Kijker heeft

U kunt dezelfde resources zien die een beheerder ziet, zoals pijplijnen, eindpunten, uitvoeringen van pijplijnen, dashboards, aangepaste integraties en triggers, maar u kunt ze niet maken, bijwerken of verwijderen. Om acties uit te voeren, moet aan de lezersrol ook de rol van projectbeheerder of projectlid worden toegewezen.

Gebruikers met de lezersrol kunnen projecten zien. Ze kunnen ook beperkte eindpunten en beperkte variabelen zien, maar kunnen geen gedetailleerde informatie zien.

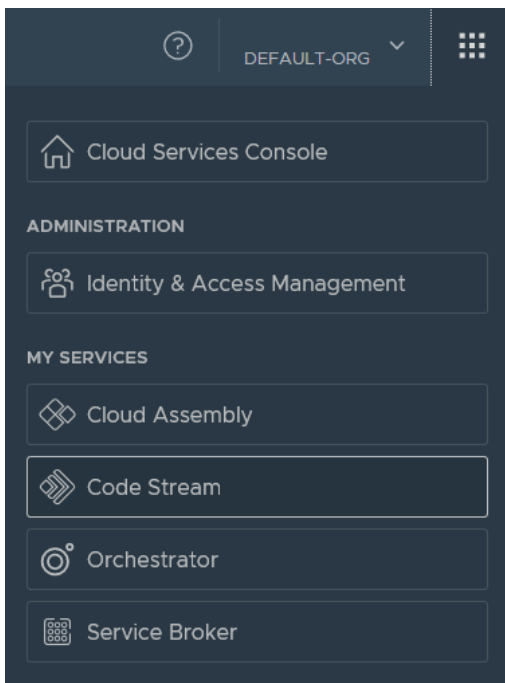
Als u de rol van Uitvoerder heeft

U kunt pijplijnen uitvoeren en actie ondernemen bij taken voor gebruikersbewerkingen. U kunt ook pijplijnuitvoeringen hervatten, pauzeren en annuleren. Maar u kunt geen pijplijnen wijzigen.

Hoe kan ik rollen toewijzen en bijwerken?

Als u rollen wilt toewijzen aan en wilt bijwerken voor andere gebruikers, moet u een beheerder zijn.

- 1 Als u de actieve gebruikers en hun rollen wilt zien, klikt u in vRealize Automation op de negen puntjes rechtsboven.
- 2 Klik op **Identiteits- en toegangsbeheer**.



- 3 Als u gebruikersnamen en -rollen wilt weergeven, klikt u op **Actieve gebruikers**.

Identity & Access Management				
Active Users		Enterprise Groups		
EDIT ROLES		REMOVE ACCESS		Q
<input type="checkbox"/>	Name	Username	Organization Roles	Service Roles
<input type="checkbox"/>	Local Admin	admin		

- 4 Om rollen voor een gebruiker toe te voegen of om zijn rollen te wijzigen, klikt u op het selectievakje naast de gebruikersnaam en klikt u op **Rollen bewerken**.

- 5 Wanneer u gebruikersrollen toevoegt of wijzigt, kunt u ook toegang tot services toevoegen.
- 6 Om uw wijzigingen op te slaan, klikt u op **Opslaan**.

Wat zijn gebruikersbewerkingen en goedkeuringen in Code Stream?

In het gebied Gebruikersbewerkingen worden pijplijnen weergegeven waarvoor goedkeuring nodig is. De vereiste goedkeurder kan de pijplijnuitvoering goedkeuren of afwijzen.

Wanneer u een pijplijn maakt, moet u mogelijk een goedkeuring toevoegen aan een pijplijn als:

- een teamlid uw code moet controleren.
- een andere gebruiker een build-artefact moet bevestigen.
- u ervoor moet zorgen dat alle tests zijn voltooid.
- Een taak gebruikt een resource die door een beheerder als beperkt is gemarkeerd en de taak moet worden goedgekeurd.
- De pijplijn zal software vrijgeven voor productie.

Om te bepalen of een pijplijntaak moet worden goedgekeurd, moet de vereiste goedkeurder rechten en expertise hebben.

Wanneer u een taak voor gebruikersbewerking toevoegt, kunt u de time-out voor verlooptijd instellen in dagen, uren of minuten. Stel bijvoorbeeld dat u wilt dat de vereiste gebruiker de pijplijn binnen 30 minuten goedkeurt. Als de gebruiker deze niet binnen 30 minuten goedkeurt, mislukt de pijplijn zoals verwacht.

Als u het verzenden van e-mailmeldingen inschakelt, verzendt de taak Gebruikersbewerking alleen meldingen naar goedkeurders die volledige e-mailadressen hebben en niet naar goedkeurdersnamen die geen e-mailindeling hebben.

Nadat de vereiste gebruiker de taak heeft goedgekeurd:

- De uitvoering van de pijplijn die in behandeling is, kan doorgaan.
- Wanneer de pijplijn doorgaat, worden alle eerder in behandeling zijnde aanvragen voor goedkeuring van dezelfde gebruikersbewerkingstaak geannuleerd.

User Operations GUIDED SETUP

Active Items Inactive Items

✓ APPROVE ✗ REJECT

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

In het gebied voor gebruikersbewerkingen worden items die moeten worden goedgekeurd of afgewezen, weergegeven als actieve of inactieve items. Elk item is toegewezen aan een gebruikersbewerkingstaak in een pijplijn.

- **Actieve items** wachten op de goedkeurder die de taak moet controleren en goedkeuren of afwijzen. Als u een gebruiker bent die op de lijst met goedkeurders staat, kunt u de gebruikersbewerkingsrij uitvouwen en op **Accepteren** of **Afwijzen** klikken.
- **Inactieve items** zijn goedgekeurd of afgewezen. Als een gebruiker de gebruikersbewerking heeft afgewezen of als er een time-out optreedt voor de goedkeuring van de taak, kan deze niet meer worden goedgekeurd.

Het indexnummer is een unieke alfanumerieke tekenreeks van zes tekens die u als filter kunt gebruiken om te zoeken naar een specifieke goedkeuring.

Pijplijngoedkeuringen worden ook weergegeven in het gebied **Uitvoeringen**.

- Pijplijnen die wachten op goedkeuring, geven hun status aan als wachtend.
- Andere statussen zijn in de wachtrij geplaatst, voltooid en mislukt.
- Als uw pijplijn een wachtstatus heeft, moet een andere gebruiker uw pijplijntaak goedkeuren.

Pijplijnen maken en gebruiken in Code Stream

3

U kunt vRealize Automation Code Stream gebruiken om uw bouw-, test- en implementatieproces te modelleren. Met vRealize Automation Code Stream stelt u de infrastructuur in die uw releasecyclus ondersteunt en maakt u pijplijnen die uw softwarereleaseactiviteiten modelleren. vRealize Automation Code Stream ontwikkelt uw software van ontwikkelingscode via testen naar de implementatie ervan in uw productie-instanties.

Elke pijplijn bevat fasen en taken. Fasen vertegenwoordigen uw ontwikkelingsfasen en taken voeren de vereiste acties uit die uw softwareapplicatie tijdens de fasen leveren.

Wat zijn pijplijnen in vRealize Automation Code Stream?

Een pijplijn is een model voor continue integratie en continue levering van uw softwarereleaseproces. Deze ontwikkelt uw software van broncode, via testen, naar productie. Deze bevat een reeks fasen met taken die de activiteiten in uw softwarereleasecyclus vertegenwoordigen. Uw softwareapplicatie doorloopt de pijplijn van de ene fase naar de volgende.

U voegt eindpunten toe zodat de taken in uw pijplijn verbinding kunnen maken met gegevensbronnen, opslagplaatsen of meldingssystemen.

Pijplijnen maken

U kunt een pijplijn maken door te beginnen met een blanco canvas, een slimme pijplijnsjabloon te gebruiken of door de YAML-code te importeren.

- Gebruik het blanco canvas. Zie [Een CI/CD-systeemeigen build plannen in Code Stream voordat u handmatig taken toevoegt](#) voor een voorbeeld.
- Gebruik een slimme pijplijnsjabloon. Zie [Hoofdstuk 4 Plannen om op een systeemeigen manier uw code te bouwen, integreren en leveren in Code Stream](#) voor een voorbeeld.
- YAML-code importeren. Klik op **Pijplijnen > Importeren**. In het dialoogvenster **Importeren** selecteert u het YAML-bestand of voert u de YAML-code in en klikt u op **Importeren**.

Wanneer u het blanco canvas gebruikt om een pijplijn te maken, voegt u fasen, taken en goedkeuringen toe. De pijplijn automatiseert het proces dat uw applicatie bouwt, test, implementeert en uitbrengt. De taken in elke fase voeren acties uit voor de bouw, het testen en de release van uw code via elke fase.

Tabel 3-1. Voorbeelden van pijplijnfasen en -toepassingen

Voorbeeldfase	Voorbeelden van wat u kunt doen
Ontwikkeling	<p>In een ontwikkelingsfase kunt u een machine inrichten, een artefact ophalen of een bouwtaak toevoegen die een Docker-host maakt voor continue integratie van uw code, en meer.</p> <p>Bijvoorbeeld:</p> <ul style="list-style-type: none"> ■ Zie Een native build voor continue integratie plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt om een CI-build (continue integratie) te plannen en te maken die uw code oplevert met behulp van de systeemeigen build-mogelijkheid in vRealize Automation Code Stream.
Test	<p>In een testfase kunt u een Jenkins-taak toevoegen om uw softwareapplicatie te testen, testtools voor naverwerking zoals JUnit en JaCoCo invoegen en meer.</p> <p>Bijvoorbeeld:</p> <ul style="list-style-type: none"> ■ Integreer vRealize Automation Code Stream met Jenkins en voer een Jenkins-taak uit in uw pijplijn, die uw broncode bouwt en test. Zie Hoe integreer ik Code Stream met Jenkins?. ■ Maak aangepaste scripts die de mogelijkheid van vRealize Automation Code Stream uitbreiden om te integreren met uw eigen tools voor het bouwen, testen en implementeren. Zie Hoe kan ik mijn eigen tools voor bouwen, testen en implementeren integreren met Code Stream?. ■ Voor een pijplijn met continue integratie (CI) kunt u de naverwerkingstrends volgen. Zie Hoe kan ik aangepaste dashboards gebruiken om KPI's voor mijn pijplijn in Code Stream te volgen?.
Productie	<p>In een productiefase kunt u een cloudsjabloon integreren in Cloud Assembly dat uw infrastructuur inricht, uw software op een Kubernetes-cluster implementeert en nog veel meer.</p> <p>Bijvoorbeeld:</p> <ul style="list-style-type: none"> ■ Zie Hoe implementeer ik mijn applicatie in Code Stream naar mijn Blue-Green-implementatie? voor voorbeeldfasen voor ontwikkeling en productie, waarmee u uw softwareapplicatie in uw eigen Blue-Green-implementatiemodel kunt implementeren. ■ Zie Hoe kan ik de release van een applicatie die ik implementeer vanuit een YAML-cloudsjabloon in Code Stream automatiseren? om een cloudsjabloon te integreren in uw pijplijn. U kunt ook een implementatietaak toevoegen waarmee een script wordt uitgevoerd om de applicatie te implementeren. ■ Automatiseer de implementatie van uw softwareapplicaties naar een Kubernetes-cluster door middel van Hoe kan ik de release van een applicatie in Code Stream op een Kubernetes-cluster automatiseren?. ■ Zie Hoe kan ik code vanuit mijn GitHub- of GitLab-opslagplaats continu integreren in mijn pijplijn in Code Stream? voor het integreren van code in uw pijplijn en het implementeren van uw build image.

U kunt uw pijplijn exporteren als een YAML-bestand. Klik op **Pijplijnen**, klik op een pijplijnkaart en klik vervolgens op **Acties > Exporteren**.

Pijplijnen goedkeuren

U kunt een goedkeuring van een ander teamlid op specifieke punten in uw pijplijn krijgen.

- Zie [Hoe kan ik een pijplijn uitvoeren en de resultaten bekijken?](#) als u goedkeuring op een pijplijn wilt vereisen door een gebruikersbewerkingstaak in een pijplijn op te nemen. Deze taak stuurt een e-mailmelding naar de gebruiker die deze moet controleren. De controleur

moet de goedkeuring goedkeuren of afwijzen voordat de pijplijn kan doorgaan met uitvoeren. Als voor de taak Gebruikersbewerking de time-out voor de verlooptijd is ingesteld op dagen, uren of minuten, moet de vereiste gebruiker de pijplijn goedkeuren voordat de taak verloopt. Anders mislukt de pijplijn zoals verwacht.

- Als een taak of fase mislukt in een willekeurige fase van een pijplijn kunt u vRealize Automation Code Stream een Jira-ticket laten maken. Zie [Hoe maak ik een JIRA-ticket in Code Stream wanneer een pijplijntaak mislukt?](#).

Pijplijnen activeren

Pijplijnen kunnen activeren wanneer ontwikkelaars hun code inchecken in de opslagplaats, of code controleren, of wanneer deze een nieuw of bijgewerkt buildartefact identificeert.

- Gebruik de Git-trigger om vRealize Automation Code Stream te integreren met de Git-levenscyclus en om een pijplijn te activeren wanneer ontwikkelaars hun code bijwerken. Zie [Hoe gebruik ik de Git-trigger in Code Stream om een pijplijn uit te voeren?](#).
- Gebruik de Gerrit-trigger om vRealize Automation Code Stream te integreren met de levenscyclus voor het controleren van de Gerrit-code en om een pijplijn te activeren bij het controleren van de code. Zie [Hoe gebruik ik de Gerrit-trigger in Code Stream om een pijplijn uit te voeren?](#).
- Als u een pijplijn wilt activeren wanneer een Docker-versieartefact wordt gemaakt of bijgewerkt, gebruikt u de Docker-trigger. Zie [Hoe kan ik de Docker-trigger gebruiken in Code Stream om een continue leveringspijplijn uit te voeren?](#).

Zie [Hoofdstuk 7 Pijplijnen in Code Stream activeren](#) voor meer informatie over de triggers die vRealize Automation Code Stream ondersteunt.

Dit hoofdstuk omvat de volgende onderwerpen:

- [Hoe kan ik een pijplijn uitvoeren en de resultaten bekijken?](#)
- [Welke typen taken zijn beschikbaar in Code Stream](#)
- [Hoe gebruik ik variabele bindingen in Code Stream-pijplijnen?](#)
- [Hoe gebruik ik variabelebindingen in een voorwaardetaak om een pijplijn uit te voeren of te stoppen in Code Stream?](#)
- [Welke variabelen en expressies kan ik gebruiken bij het binden van pijplijntaken in Code Stream?](#)
- [Hoe verzend ik meldingen over mijn pijplijn in Code Stream?](#)
- [Hoe maak ik een JIRA-ticket in Code Stream wanneer een pijplijntaak mislukt?](#)
- [Hoe kan ik mijn implementatie terugdraaien in Code Stream?](#)

Hoe kan ik een pijplijn uitvoeren en de resultaten bekijken?

U kunt een pijplijn uitvoeren vanuit de pijplijnkaart, in de pijplijnbewerkingsmodus en bij de uitvoering van de pijplijn. U kunt ook de beschikbare triggers gebruiken om Code Stream een pijplijn uit te laten voeren wanneer bepaalde gebeurtenissen optreden.

Wanneer alle fasen en taken in uw pijplijn geldig zijn, is de pijplijn klaar voor release of om te worden uitgevoerd of te worden geactiveerd.

Als u uw pijplijn wilt uitvoeren of activeren met Code Stream, kunt u de pijplijn inschakelen en uitvoeren zowel vanaf de pijplijnkaart of terwijl u zich in de pijplijn bevindt. Vervolgens kunt u de pijplijnuitvoering bekijken om te bevestigen dat de pijplijn uw code heeft gebouwd, getest en geïmplementeerd.

Wanneer een pijplijn wordt uitgevoerd, kunt u de uitvoering verwijderen als u een beheerder bent of een gebruiker die geen beheerder is.

- Beheerder: als u een pijplijnuitvoering wilt verwijderen wanneer deze wordt uitgevoerd, klikt u op **Uitvoeringen**. Klik op de te verwijderen uitvoering op **Acties > Verwijderen**.
- Gebruiker die geen beheerder is: als u een lopende uitvoering van een pijplijn wilt verwijderen, klikt u op **Uitvoeringen** en klikt u op **Alt Shift d**.

Wanneer een pijplijn wordt uitgevoerd en vastgelopen lijkt te zijn, kan een beheerder de uitvoering vernieuwen vanaf de pagina Uitvoeringen of de pagina Uitvoeringsdetails.

- Uitvoeringspagina: klik op **Uitvoeringen**. Klik op de te vernieuwen uitvoering op **Acties > Synchronisatie**.
- Uitvoeringsdetails: klik op **Uitvoeringen**, klik op de link naar de uitvoeringsdetails en klik op **Acties > Synchronisatie**.

Gebruik de triggers om een pijplijn uit te voeren wanneer specifieke gebeurtenissen plaatsvinden.

- De Git-trigger kan een pijplijn uitvoeren wanneer ontwikkelaars code bijwerken.
- De Gerrit-trigger kan een pijplijn uitvoeren wanneer coderevisies plaatsvinden.
- De Docker-trigger kan een pijplijn uitvoeren wanneer een artefact wordt gemaakt in een Docker-register.
- Het commando `curl` of het commando `wget` kan Jenkins een pijplijn laten uitvoeren nadat een Jenkins-build is voltooid.

Zie [Hoofdstuk 7 Pijplijnen in Code Stream activeren](#) voor meer informatie over het gebruik van de triggers.

De volgende procedure laat zien hoe u een pijplijn vanuit de pijplijnkaart kunt uitvoeren, uitvoeringen kunt bekijken, uitvoeringsdetails kunt zien en de acties kunt gebruiken. Ook wordt uitgelegd hoe u een pijplijn vrijgeeft, zodat u deze kunt toevoegen aan vRealize Automation Service Broker.

Voorwaarden

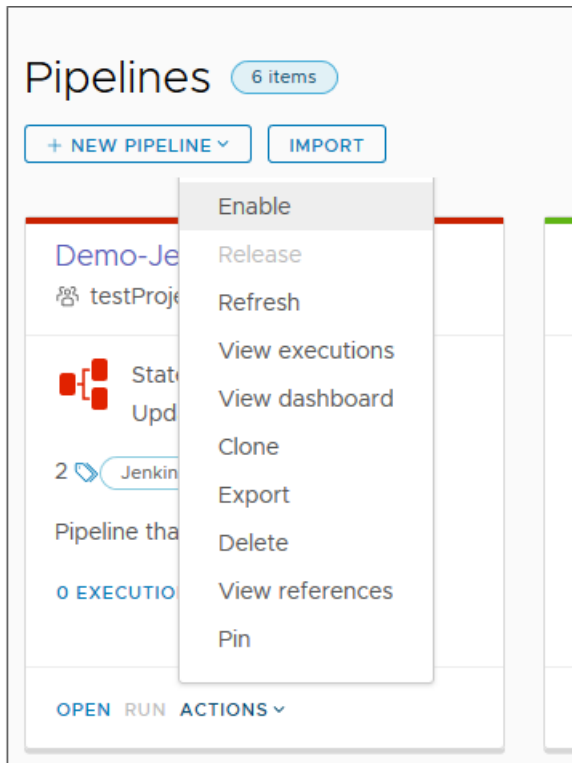
- Verifieer dat een of meerdere pijplijnen zijn gemaakt. Zie de voorbeelden in [Hoofdstuk 5 Tutorials voor het gebruik van Code Stream](#).

Procedure

1 Schakel uw pijplijn in.

Als u een pijplijn wilt uitvoeren of vrijgeven, moet u deze eerst inschakelen.

- Klik op **Pijplijnen**.
- Klik op uw pijplijnkaart op **Acties > Inschakelen**.



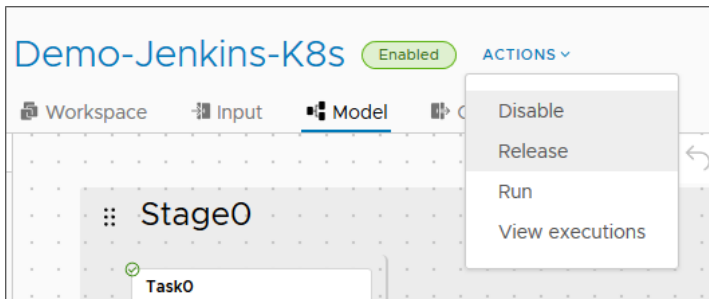
U kunt ook uw pijplijn inschakelen terwijl u in de pijplijn bent. Als uw pijplijn al is ingeschakeld, is **Uitvoeren** actief en geeft het menu **Acties Uitschakelen** weer.

2 (Optioneel) Geef uw pijplijn vrij.

Als u uw pijplijn beschikbaar wilt maken als catalogusitem in vRealize Automation Service Broker, moet u deze vrijgeven in Code Stream.

- a Klik op **Pijplijnen**.
- b Klik op uw pijplijnkaart op **Acties > Release**.

U kunt uw pijplijn ook vrijgeven wanneer u in de pijplijn bent.



Nadat u de pijplijn heeft vrijgegeven, opent u Service Broker om de pijplijn als catalogusitem toe te voegen en voert u deze uit. Zie [Code Stream-pijplijnen toevoegen aan de Service Broker-catalogus](#).

Opmerking Als het meer dan 120 minuten duurt om de pijplijn uit te voeren, geeft u een geschatte uitvoeringstijd op als een time-outwaarde voor de aanvraag. Als u de time-out van de aanvraag voor een project wilt instellen of bekijken, opent u Service Broker als beheerder en selecteert u **Infrastructuur > Projecten**. Klik op uw projectnaam en klik vervolgens op **Inrichting**.

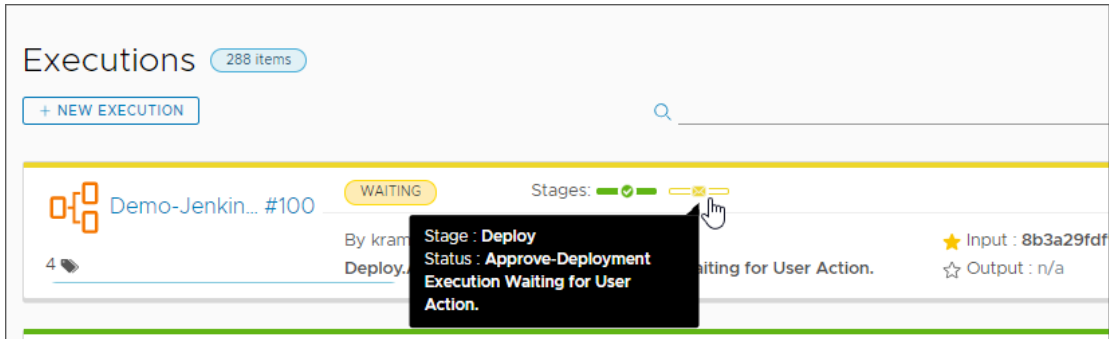
Als de time-outwaarde voor de aanvraag niet is ingesteld, wordt een uitvoering die meer dan 120 minuten vereist weergegeven als mislukt met een time-outfout bij de callbackaanvraag. De uitvoering van de pijplijn wordt echter niet beïnvloed.

- 3 Op de pijplijnkaart klikt u op **Uitvoeren**.
- 4 Als u de pijplijn wilt bekijken terwijl deze wordt uitgevoerd, klikt u op **Uitvoeringen**.

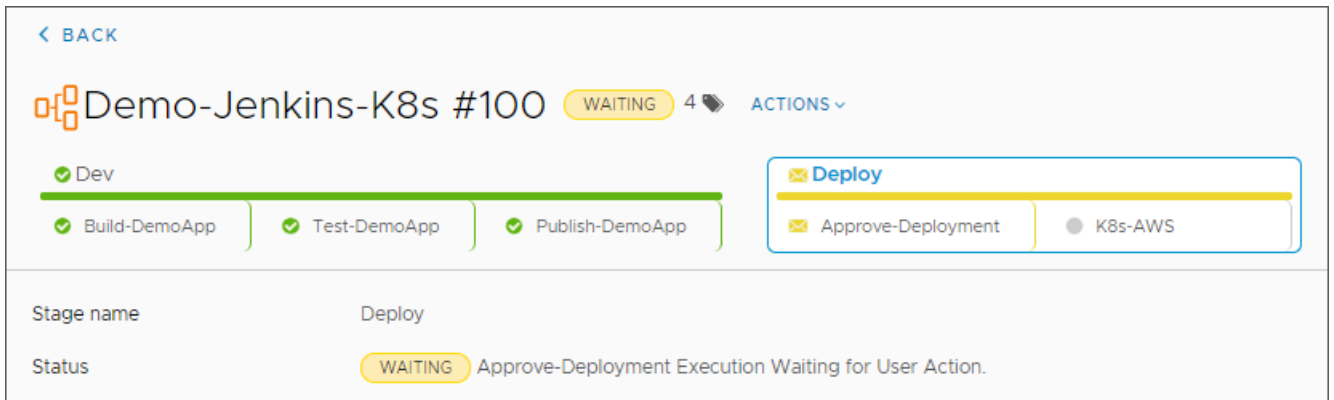
De pijplijn voert elke fase in de reeks uit en de pijplijnuitvoering geeft een statuspictogram weer voor elke fase. Als de pijplijn een gebruikersbewerkingstaak bevat, moet een gebruiker de taak goedkeuren zodat de pijplijn door kan gaan met de uitvoering. Wanneer een gebruikersbewerkingstaak wordt gebruikt, stopt de pijplijn en wacht die totdat de vereiste gebruiker de taak goedkeurt.

U kunt bijvoorbeeld de gebruikersbewerkingstaak gebruiken om de implementatie van code goed te keuren voor een productieomgeving.

Als voor de taak Gebruikersbewerking de time-out voor de verlooptijd is ingesteld op dagen, uren of minuten, moet de vereiste gebruiker de pijplijn goedkeuren voordat de taak verloopt. Anders mislukt de pijplijn zoals verwacht.



- 5 Als u de pijplijnfase wilt zien die op goedkeuring van de gebruiker wacht, klikt u op het statuspictogram voor de fase.

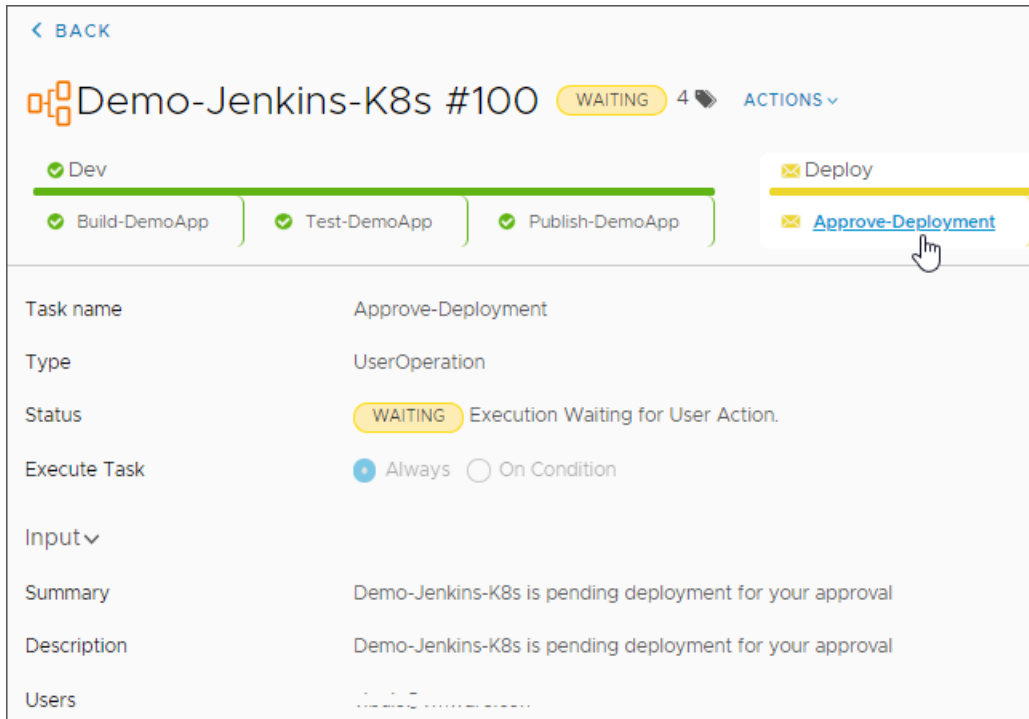


- 6 Als u de details van de taak wilt zien, klikt u op de taak.

Nadat de vereiste gebruiker de taak heeft goedgekeurd, moet een gebruiker met de relevante rol de pijplijn hervatten. Zie [Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream?](#) voor de vereiste rollen.

Als een uitvoering mislukt, moet u triage toepassen en de oorzaak van de fout oplossen. Ga vervolgens naar de uitvoering en klik op **Acties > Opnieuw uitvoeren**.

U kunt de uitvoeringen van de hoofdpijplijn en geneste uitvoeringen hervatten.



- 7 Bij de uitvoering van de pijplijn kunt u op **Acties** klikken om de pijplijn weer te geven en een actie selecteren zoals **Pauzeren**, **Annuleren** en meer. Wanneer een pijplijn wordt uitgevoerd, kunt u de pijplijnuitvoering verwijderen of synchroniseren als u een beheerder bent. Als u een gebruiker bent die geen beheerder is, kunt u een actieve pijplijn verwijderen.
- 8 Om eenvoudig te navigeren tussen uitvoeringen en de details van een taak weer te geven, klikt u op **Uitvoeringen** en klikt u op een pijplijnuitvoering. Klik vervolgens op het tabblad bovenaan en selecteer de pijplijnuitvoering.



Resultaten

Gefeliciteerd! U heeft een pijplijn uitgevoerd, de uitvoering van de pijplijn onderzocht en een gebruikersbewerkingstaak bekeken waarvoor goedkeuring vereist is zodat de pijplijn verder kan gaan met de uitvoering. U heeft ook het menu **Acties** in de pijplijnuitvoering gebruikt om terug te keren naar het pijplijnmodel, zodat u eventuele gewenste wijzigingen kunt aanbrengen.

Wat nu te doen

Zie [Hoofdstuk 5 Tutorials voor het gebruik van Code Stream](#) voor meer informatie over het gebruik van Code Stream om uw software-releasecyclus te automatiseren.

Welke typen taken zijn beschikbaar in Code Stream

Wanneer u uw pijplijn configureert, voegt u specifieke typen taken toe die de pijplijn uitvoert voor de acties die u nodig hebt. Elk taaktype integreert met een andere applicatie en schakelt uw pijplijn in terwijl deze uw applicaties bouwt, test en levert.

Om uw pijplijn uit te voeren, ongeacht of u artefacten uit een opslagplaats moet halen voor implementatie, een extern script moet uitvoeren of goedkeuring moet vereisen voor een gebruikersbewerking van een teamlid, heeft Code Stream het type taak voor u.

Code Stream ondersteunt het annuleren van een pijplijnuitvoering voor verschillende taaktypen. Wanneer u op **Annuleren** klikt bij de uitvoering van een pijplijn, krijgt de taak, fase of volledige pijplijn de annuleringsstatus en wordt de pijplijnuitvoering geannuleerd.

Met Code Stream kunt u de pijplijnuitvoering voor een taak, fase of de volledige pijplijn annuleren wanneer u deze taken gebruikt:

- Jenkins
- SSH
- PowerShell
- Gebruikersbewerking
- Pijplijn
- Cloudsjabloon
- vRO
- POLL

Code Stream geeft het annuleringsgedrag niet door aan systemen van derden voor deze taken: continue integratie, aangepaste integratie of Kubernetes. Code Stream markeert de taak als geannuleerd en stopt onmiddellijk met het ophalen van de status zonder te wachten totdat de taak is voltooid. De taak kan worden voltooid of mislukken op het systeem van derden, maar de uitvoering wordt onmiddellijk gestopt in Code Stream wanneer u op **Annuleren** klikt.

Voordat u een taak in uw pijplijn gebruikt, moet u controleren of het bijbehorende eindpunt beschikbaar is.

Tabel 3-2. Een goedkeuring verkrijgen of een beslissingspunt instellen

Type taak	Wat het doet	Voorbeelden en details
Gebruikersbewerking	Een taak Gebruikersbewerking schakelt een vereiste goedkeuring in die bepaalt wanneer een pijplijn wordt uitgevoerd en moet stoppen voor een goedkeuring.	Zie Hoe kan ik een pijplijn uitvoeren en de resultaten bekijken? en Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream? .
Voorwaarde	Voegt een beslissingspunt toe, dat bepaalt of de pijplijn nog steeds wordt uitgevoerd of stopt, op basis van voorwaarde-expressies. Wanneer de voorwaarde waar is, voert de pijplijn opeenvolgende taken uit. Indien deze onwaar is, stopt de pijplijn.	Zie Hoe gebruik ik variabelebindingen in een voorwaardetaak om een pijplijn uit te voeren of te stoppen in Code Stream? .

Tabel 3-3. Continue integratie en implementatie automatiseren

Type taak	Wat het doet	Voorbeelden en details
Cloudsjabloon	<p>Implementeert een automatiseringscloudsjabloon van GitHub, richt een applicatie in en automatiseert de continue integratie en continue levering (CICD) van die cloudsjabloon voor uw implementatie.</p>	<p>Zie Hoe kan ik de release van een applicatie die ik implementeer vanuit een YAML-cloudsjabloon in Code Stream automatiseren?.</p> <p>De parameters van de cloudsjabloon worden weergegeven nadat u eerst Maken of Bijwerken hebt geselecteerd en vervolgens Cloudsjabloon en Versie hebt geselecteerd. U kunt deze elementen, die variabelebindingen bevatten, toevoegen aan de invoertekstgebieden in de cloudsjabloontaak:</p> <ul style="list-style-type: none"> ■ Geheel getal ■ Opsommingstekenreeks ■ Booleaans ■ Array-variabele <p>Wanneer u een variabelebinding gebruikt in de invoer, moet u rekening houden met deze uitzonderingen. Voor opsommingen moet u een opsommingswaarde uit een vaste reeks selecteren. Voor Boole-waarden moet u de waarde invoeren in het tekstgebied voor invoer.</p> <p>De cloudsjabloonparameter wordt weergegeven in de cloudsjabloontaak wanneer een cloudsjabloon in Cloud Assembly invoervariabelen bevat. Als een cloudsjabloon bijvoorbeeld een invoertype van <code>Integer</code> heeft, kunt u het gehele getal rechtstreeks of als een variabele invoeren met behulp van een variabelebinding.</p>
CI	<p>De CI-taak zorgt voor een voortdurende integratie van uw code in uw pijplijn door een Docker-buildimage uit een registreindpunt te halen en deze te implementeren in een Kubernetes-cluster.</p> <p>De CI-taak geeft 100 regels van het logboek weer als uitvoer en geeft 500 regels weer wanneer u de logboeken downloadt.</p> <p>Voor de CI-taken zijn kortstondige poorten 32768 tot 61000 vereist.</p>	<p>Zie Een systeemeigen CICD-build plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt.</p>
Custom	<p>De aangepaste taak integreert Code Stream met uw eigen tools voor het bouwen, testen en implementeren.</p>	<p>Zie Hoe kan ik mijn eigen tools voor bouwen, testen en implementeren integreren met Code Stream?.</p>

Tabel 3-3. Continue integratie en implementatie automatiseren (vervolg)

Type taak	Wat het doet	Voorbeelden en details
Kubernetes	Automatiseer de implementatie van uw softwareapplicaties naar Kubernetes-clusters op AWS.	Zie Hoe kan ik de release van een applicatie in Code Stream op een Kubernetes-cluster automatiseren? .
Pijplijn	<p>Nest een pijplijn in een hoofdpijplijn. Wanneer een pijplijn is genest, gedraagt deze zich als een taak in de hoofdpijplijn.</p> <p>Op het tabblad Taak van de hoofdpijplijn kunt u eenvoudig naar de geneste pijplijn gaan door op de link ernaar te klikken. De geneste pijplijn wordt geopend in een nieuw tabblad van de browser.</p>	Als u geneste pijplijnen wilt vinden in Uitvoeringen , voert u genest in het zoekveld in.

Tabel 3-4. Ontwikkelings-, test- en implementatieapplicaties integreren

Taaktype...	Wat het doet...	Voorbeelden en details...
Bamboo	Communiqueert met een Bamboo-server voor continue integratie (CI), waarmee voortdurend software wordt gebouwd, getest en geïntegreerd in de voorbereiding op implementatie, en codebuilds worden geactiveerd wanneer ontwikkelaars wijzigingen doorvoeren. Het geeft de artefactlocaties weer die de Bamboo-build produceert, zodat de taak de parameters als output kan geven die andere taken moeten gebruiken voor de bouw en implementatie.	Maak verbinding met een Bamboo-server-eindpunt en start een Bamboo-bouwplan vanuit uw pijplijn.
Jenkins	Activeert Jenkins-taken die uw broncode bouwen en testen, testcases uitvoeren en custom scripts kunnen gebruiken.	Zie Hoe integreer ik Code Stream met Jenkins? .
TFS	Stelt u in staat om uw pijplijn te verbinden met de Team Foundation Server om bouwprojecten te beheren en aan te roepen, inclusief geconfigureerde taken voor het bouwen en testen van uw code.	Zie Wat zijn eindpunten in Code Stream? voor versies van Team Foundation Server die worden ondersteund door Code Stream.
vRO	<p>Breidt de mogelijkheid van Code Stream uit door vooraf gedefinieerde of custom werkstromen in vRealize Orchestrator uit te voeren.</p> <p>Code Stream ondersteunt basisverificatie en verificatie op basis van tokens voor vRealize Orchestrator. Code Stream gebruikt het API-token om het vRealize Orchestrator-cluster te verifiëren en te valideren. Met verificatie op basis van tokens ondersteunt Code Stream vRealize Orchestrator-eindpunten die een clouduitbreidbaarheidsproxy gebruiken. Als gevolg hiervan kunt u in Code Stream werkstromen activeren met een vRealize Orchestrator-eindpunt dat de clouduitbreidbaarheidsproxy gebruikt.</p>	Zie Hoe integreer ik Code Stream met vRealize Orchestrator? .

Tabel 3-5. Andere applicaties integreren met behulp van een API

Taaktype...	Wat het doet...	Voorbeelden en details...
REST	Integreert Code Stream met andere applicaties die gebruikmaken van een REST API, zodat u voortdurend softwareapplicaties kunt ontwikkelen en leveren die met elkaar communiceren.	Zie Hoe gebruik ik een REST API om Code Stream te integreren met andere applicaties? .
Poll	<p>Roept een REST API aan en pollt deze totdat de pijplijntaak voldoet aan de afsluitcriteria en wordt voltooid.</p> <p>Een Code Stream-beheerder kan het aantal polls instellen op een maximum van 10.000. Het pollinginterval moet 60 seconden of meer zijn.</p> <p>Wanneer u het selectievakje Doorgaan bij mislukt inschakelt, gaat de uitvoering van de polling door bij een overschrijding van dit aantal of interval.</p> <p>POLL Iteration Count: wordt weergegeven in de pijplijnuitvoering en geeft weer hoeveel keer de POLL-taak een antwoord van de URL heeft aangevraagd. Als de POLL-invoer bijvoorbeeld 65 is en 4 het aantal keer is dat de POLL-aanvraag is uitgevoerd, wordt 4 (van de 65) weergegeven voor het aantal iteraties in de uitvoer van de pijplijnuitvoering.</p>	Zie Hoe gebruik ik een REST API om Code Stream te integreren met andere applicaties? .

Tabel 3-6. Externe en door de gebruiker gedefinieerde scripts uitvoeren

Type taak	Wat het doet	Voorbeelden en details
PowerShell	<p>Met de PowerShell-taak kan Code Stream scriptopdrachten uitvoeren op een externe host. Een script kan bijvoorbeeld testtaken automatiseren en beheertypen van commando's uitvoeren.</p> <p>Het script kan extern zijn of door de gebruiker worden gedefinieerd. Het kan verbinding maken via HTTP of HTTPS en kan TLS gebruiken.</p> <p>Voor de Windows-host moet de service winrm zijn geconfigureerd en voor winrm moeten MaxShellsPerUser en MaxMemoryPerShellMB zijn geconfigureerd.</p> <p>Om een PowerShell-taak uit te voeren, moet u een actieve sessie voor de externe Windows-host hebben.</p> <p>Lengte van PowerShell-opdrachtregel</p> <p>Als u een base64 PowerShell-opdracht invoert, moet u er rekening mee houden dat u de totale opdrachtlength moet berekenen.</p> <p>De Code Stream-pijplijn codeert en verpakt een base64 PowerShell-opdracht in een andere opdracht, waardoor de algemene lengte van de opdracht toeneemt.</p> <p>De maximale lengte die voor de PowerShell-opdracht winrm is toegestaan, is 8192 bytes. De opdrachtlengthlimiet is lager voor de PowerShell-taak wanneer deze is gecodeerd en verpakt. Als gevolg hiervan moet u de opdrachtlength berekenen voordat u de PowerShell-opdracht invoert.</p> <p>De maximale opdrachtlength voor de Code Stream PowerShell-taak is afhankelijk van de base64-gecodeerde lengte van de oorspronkelijke opdracht. De opdrachtlength wordt als volgt berekend.</p> <pre>3 * (length of original command / 4)) - (numberOfPaddingCharacters) + 77 (Length of Write-output command)</pre> <p>De opdrachtlength voor Code Stream moet korter zijn dan de maximale limiet van 8192.</p>	<p>Wanneer u MaxShellsPerUser en MaxMemoryPerShellMB configureert:</p> <ul style="list-style-type: none"> De acceptabele waarde voor MaxShellsPerUser is 500 voor 50 gelijktijdige pijplijnen, met 5 PowerShell-taken voor elke pijplijn. Als u de waarde wilt instellen, voert u: winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}' uit. De acceptabele geheugenwaarde voor MaxMemoryPerShellMB is 2048. Als u de waarde wilt instellen, voert u: winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}' uit. <p>Het script schrijft de output naar een responsbestand dat door een andere pijplijn kan worden geconsumeerd.</p>
Secure Shell	<p>Met de SSH-taak kan de Bash-shellscriphtaak scriptopdrachten uitvoeren op een externe host. Een script kan bijvoorbeeld testtaken automatiseren en beheertypen van commando's uitvoeren.</p> <p>Het script kan extern zijn of door de gebruiker worden gedefinieerd. Het kan verbinding maken via HTTP of HTTPS en vereist een persoonlijke sleutel of wachtwoord.</p>	<p>Het script kan extern zijn of door de gebruiker worden gedefinieerd. Een script kan er bijvoorbeeld zo uitzien:</p> <pre>message="Hello World" echo \$message</pre> <p>Het script schrijft de output naar een responsbestand dat door een andere pijplijn kan worden geconsumeerd.</p>

Tabel 3-6. Externe en door de gebruiker gedefinieerde scripts uitvoeren (vervolg)

Type taak	Wat het doet	Voorbeelden en details
	De Secure Shell-service moet worden geconfigureerd op de Linux-host en de SSHD-configuratie van <code>MaxSessions</code> moet worden ingesteld op 50. Als u veel SSH-taken gelijktijdig uitvoert, verhoogt u <code>MaxSessions</code> en <code>MaxOpenSessions</code> op de SSH-host. Gebruik uw vRealize Automation-instantie niet als SSH-host als u de configuratie-instellingen voor <code>MaxSessions</code> en <code>MaxOpenSessions</code> moet wijzigen.	

Hoe gebruik ik variabele bindingen in Code Stream-pijplijnen?

Het binden van een pijplijntaak houdt in dat u een afhankelijkheid voor de taak maakt wanneer de pijplijn wordt uitgevoerd. U kunt op verschillende manieren een binding voor een pijplijntaak maken. U kunt een taak binden aan een andere taak, deze binden aan een variabele en expressie, of deze binden aan een voorwaarde.

Dollarbindingen toepassen op cloudsjabloonvariabelen in een cloudsjabloontaak

U kunt dollarbindingen toepassen op cloudsjabloonvariabelen in een cloudsjabloontaak in een Code Stream-pijplijn. De manier waarop u de variabelen in Code Stream wijzigt, is afhankelijk van de codering van de eigenschappen van de variabele in de cloudsjabloon.

Als u dollarbindingen in een cloudsjabloontaak moet gebruiken, maar de huidige versie van de cloudsjabloon die u gebruikt in de cloudsjabloontaak dit niet toestaat, kunt u de cloudsjabloon in Cloud Assembly wijzigen en een nieuwe versie implementeren. Vervolgens gebruikt u de nieuwe cloudsjabloonversie in uw cloudsjabloontaak en voegt u de dollarbindingen waar nodig toe.

Om dollarbindingen toe te passen op de typen eigenschappen die de Cloud Assembly-cloudsjabloon biedt, moet u over de juiste rechten beschikken.

- U moet dezelfde rol hebben als de persoon die de cloudsjabloonimplementatie heeft gemaakt in Cloud Assembly.
- De persoon die de pijplijn modelleert en de persoon die de pijplijn uitvoert, kunnen twee verschillende gebruikers zijn en kunnen verschillende rollen hebben.
- Als een ontwikkelaar de Code Stream-uitvoerderfunctie heeft en de pijplijn modelleert, moet de ontwikkelaar ook dezelfde Cloud Assembly-rol hebben als de persoon die de cloudsjabloon heeft geïmplementeerd. De vereiste rol kan bijvoorbeeld Cloud Assembly-beheerder zijn.
- Alleen de persoon die de pijplijn modelleert, kan de pijplijn maken en de implementatie maken omdat hij of zij rechten heeft.

Zo gebruikt u een API-token in de cloudsjabloontaak:

- De persoon die de pijplijn modelleert, kan een API-token geven aan een andere gebruiker die de rol van Code Stream-uitvoerder heeft. Wanneer de uitvoerder de pijplijn uitvoert, gebruikt deze het API-token en de verificatiegegevens die het API-token maakt.
- Wanneer een gebruiker het API-token in de cloudsjabloontaak invoert, worden de verificatiegegevens gemaakt die nodig zijn voor de pijplijn.
- Als u de API-tokenwaarde wilt versleutelen, klikt u op **Variabele maken**.
- Als u geen variabele voor het API-token maakt en deze in de cloudsjabloontaak gebruikt, wordt de waarde van het API-token weergegeven als tekst zonder opmaak.

Volg de volgende stappen als u dollarbindingen wilt toepassen op cloudsjabloonvariabelen in een cloudsjabloontaak.

U begint met een cloudsjabloon waarop invoervariabele-eigenschappen zijn gedefinieerd, zoals `integerVar`, `stringVar`, `flavorVar`, `BooleanVar`, `objectVar`, en `arrayVar`. U kunt de image-eigenschappen vinden die in de sectie `resources` zijn gedefinieerd. De eigenschappen in de cloudsjablooncode kunnen er als volgt uitzien:

```
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}'
```

U kunt dollartekenvariabelen (\$) gebruiken voor `image` en `flavor`. Bijvoorbeeld:

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

Als u een cloudsjabloon wilt gebruiken in een Code Stream-pijplijn en hieraan dollarbindingen wilt toevoegen, volgt u deze stappen.

- 1 Klik in Code Stream op **Pijplijnen > Blanco canvas**.
- 2 Voeg een taak **Cloudsjabloon** toe aan de pijplijn.
- 3 In de taak Cloudsjabloon selecteert u **Cloud Assembly Cloud Templates** voor **Bron van cloudsjabloon**, voert u de naam van de cloudsjabloon in en selecteert u de versie van de cloudsjabloon.
- 4 Merk op dat u een API-token kunt invoeren, dat verificatiegegevens voor de pijplijn biedt. Als u een variabele wilt maken die het API-token in de taak van de cloudsjabloon versleutelt, klikt u op **Variabele maken**.
- 5 In de tabel **Parameter en waarde** die wordt weergegeven, ziet u de parameterwaarden. De standaardwaarde voor `soort` is `klein` en de standaardwaarde voor `image` is `ubuntu`.
- 6 Stel dat u de cloudsjabloon in Cloud Assembly moet wijzigen. U kunt bijvoorbeeld het volgende doen:
 - a Stel de `soort` in zodat deze een eigenschap van het type `array` gebruikt. Cloud Assembly staat door komma's gescheiden waarden toe voor `Soort` wanneer het type **array** is.
 - b Klik op **Implementeren**.
 - c Voer op de pagina Implementatietype een implementatienaam in en selecteer de versie van de cloudsjabloon.
 - d Op de pagina Implementatie-invoeren kunt u een of meerdere waarden definiëren voor `Soort`.
 - e U ziet dat de Implementatie-inputs alle variabelen bevatten die zijn gedefinieerd in uw cloudsjablooncode en die worden weergegeven zoals gedefinieerd in de cloudsjablooncode. Bijvoorbeeld: `Integer Var`, `String Var`, `Flavor Var`, `Boolean Var`, `Object Var` en `Array Var`. `String Var` en `Flavor Var` zijn tekenreekswaarden en `Boolean Var` is een selectievakje.
 - f Klik op **Implementeren**.

- 7 Selecteer in Code Stream de nieuwe versie van de cloudsjabloon en voer waarden in de tabel **Parameter en waarde** in. Cloudsjablonen ondersteunen de volgende typen parameters, die Code Stream-bindingen met behulp van dollartekenvariabelen inschakelen. Er zijn kleine verschillen tussen de gebruikersinterface van de Code Stream-cloudsjabloontaak en de gebruikersinterface van de Cloud Assembly-cloudsjabloon. Afhankelijk van de codering van een cloudsjabloon in Cloud Assembly is het mogelijk niet toegestaan om waarden in de cloudsjabloontaak in Code Stream in te voeren.
- Voor **flavorVar**: als de cloudsjabloon het type als tekenreeks of array heeft gedefinieerd, voert u een tekenreeks of een array met door komma's gescheiden waarden in. Een voorbeeld-array lijkt op **test, test**.
 - Voor **BooleanVar** selecteert u **true** of **false** in het vervolgkeuzemenu. Of als u een variabelebinding wilt gebruiken, voert u **\$** in en selecteert u een variabelebinding in de

Parameter	Value
stringVar	raj
integerVar	1
flavorVar	medium
BooleanVar	\$
objectVar	var
arrayVar	input
	comments
	requestBy
	executionIndex
	executionId
	executionUrl
Output Parameter	name
	description
	Stage0

lijst.

- Voor **objectVar** de waarde tussen accolades en aanhalingstekens in deze indeling in: **{"bkix": "bkix":}**.
 - De **objectVar** wordt doorgestuurd naar de cloudsjabloon en kan op verschillende manieren worden gebruikt, afhankelijk van de cloudsjabloon. Hiermee kunt u een tekenreeksindeling voor een JSON-object maken en u kunt sleutelwaardeparen als door komma's gescheiden waarden toevoegen in de tabel met sleutelwaarden. U kunt tekst zonder opmaak invoeren voor een JSON-object of een sleutelwaardepaar als een normale tekenreeksindeling voor JSON.
 - Voor **arrayVar** voert u de door komma's gescheiden invoerwaarde in als een array met de volgende indeling: **["1", "2"]**.
- 8 In de pijplijn kunt u een invoerparameter binden aan een array.
- Klik op het tabblad **Invoer**.
 - Voer een naam in voor de invoer. Bijvoorbeeld **arrayInput**.
 - In de tabel **Parameter en waarde** klikt u op **arrayVar** en voert u **\${input.arrayInput}** in.

- d Nadat u de pijplijn heeft opgeslagen en ingeschakeld, moet u een waarde invoeren voor de array-invoer wanneer de pijplijn wordt uitgevoerd. Voer bijvoorbeeld `["1", "2"]` in en klik op **Uitvoeren**.

Nu heeft u geleerd hoe u variabelebindingen met het dollarteken (\$) kunt gebruiken in een cloudsjabloon in een cloudsjabloontaak van een Code Stream-pijplijn.

Een parameter doorgeven aan een pijplijn wanneer deze wordt uitgevoerd

U kunt invoerparameters aan uw pijplijn toevoegen zodat Code Stream deze doorgeeft aan de pijplijn. Wanneer de pijplijn wordt uitgevoerd, moet een gebruiker de waarde invoeren voor de invoerparameter. Wanneer u uitvoerparameters aan uw pijplijn toevoegt, kunnen de pijplijntaken de uitvoerwaarde van een taak gebruiken. Code Stream ondersteunt het gebruik van parameters op veel manieren die uw eigen pijplijnbehoeften ondersteunen.

Als u bijvoorbeeld een gebruiker wilt vragen naar de URL naar zijn Git-server wanneer een pijplijn met een REST-taak wordt uitgevoerd, kunt u de REST-taak binden aan een Git-server-URL.

Als u de variabelebinding wilt maken, voegt u een URL-bindingsvariabele toe aan de REST-taak. Wanneer de pijplijn wordt uitgevoerd en de REST-taak is bereikt, moet een gebruiker de URL voor de Git-server invoeren. U kunt als volgt de binding maken:

- 1 Klik op het tabblad **Invoer** in uw pijplijn.
- 2 Om de parameter in te stellen voor **Parameters automatisch injecteren** klikt u op **Git**.
De lijst met Git-parameters wordt weergegeven en bevat **GIT_SERVER_URL**. Als u een standaardwaarde voor de URL van de Git-server moet gebruiken, kunt u deze parameter bewerken.
- 3 Klik op **Model** en klik op uw REST-taak.
- 4 Voer op het tabblad **Taak** in het gebied **URL \$** in en selecteer vervolgens **invoer** en **GIT_SERVER_URL**.

Task : **Task3** Notifications Rollback **VALIDATE TASK**

Task name ⓘ * Task3

Type * REST

Continue on failure ☐

Execute task ☒ Always ☐ On condition

REST Request

Action * GET

URL \$ * \${input.GIT_SERVER_URL}

Agent endpoint

Headers

Output Parameters

status responseHeaders responseBody responseJson responseCode

De vermelding ziet er ongeveer zo uit: `${input.GIT_SERVER_URL}`

- 5 Om de integriteit van de binding van variabelen voor de taak te controleren, klikt u op **Taak valideren**.

Code Stream geeft aan dat de taak met succes is gevalideerd.

- 6 Wanneer de pijplijn de REST-taak uitvoert, moet een gebruiker de URL voor de Git-server invoeren. Anders wordt de taak niet voltooid.

Twee pijplijntaken binden door invoer- en uitvoerparameters te maken

Als u twee taken aan elkaar wilt binden, voegt u een bindingsvariabele toe aan de invoerconfiguratie van de ontvangende taak. Wanneer de pijplijn wordt uitgevoerd, vervangt een gebruiker de bindingsvariabele door de vereiste input.

Als u pijplijntaken samen wilt binden, gebruikt u de dollartekenvariabele (\$) in de invoerparameters en uitvoerparameters. In dit voorbeeld ziet u hoe u.

Stel dat u wilt dat uw pijplijn een URL in een REST-taak oproept en een antwoord uitvoert. Om de URL aan te roepen en het antwoord uit te voeren, neemt u zowel de invoerparameters als de uitvoerparameters op in uw REST-taak. U moet ook een gebruiker hebben die de taak kan goedkeuren, en een taak Gebruikersbewerkingen opnemen voor een andere gebruiker die deze kan goedkeuren wanneer de pijplijn wordt uitgevoerd. Dit voorbeeld toont hoe u expressies kunt gebruiken in de invoer- en uitvoerparameters en de pijplijn kunt laten wachten op goedkeuring van de taak.

- 1 Klik op het tabblad **Invoer** in uw pijplijn.

rest-ix-1 Enabled ACTIONS ▾

Workspace **Input** Model Output

Input Parameters ⓘ

Auto inject parameters ☐ Gerrit ☐ Git ☐ Docker ☒ None

ADD ADD/REMOVE INJECTED PARAMETERS

Starred ⓘ	Name ▾	Value ▾	Description ▾
⋮ ☆	URL	{Stage0.Task3.input.http://www.docs.vmware.com}	Docs URL

- 2 Laat de **Automatisch parameters injecteren** op **Geen**.
- 3 Klik op **Toevoegen** en voer de naam, waarde en beschrijving van de parameter in en klik op **OK**. Bijvoorbeeld:
 - a Voer een URL-naam in.
 - b Voer de waarde in: {Stage0.Task3.input.http://www.docs.vmware.com}
 - c Voer een beschrijving in.
- 4 Klik op het tabblad **Uitvoer**, klik op **Toevoegen** en voer de naam van de uitvoerparameter en de toewijzing in.

Add Pipeline Output Parameter

Name *

Reference \$ *

responseHeaders
 responseBody
 responseJson
 responseCode

- a Voer een unieke naam voor de uitvoerparameter in.
- b Klik in het gebied **Referentie** en voer \$ in.
- c Voer de toewijzing van de taakuitvoer in door de opties te selecteren wanneer ze verschijnen. Selecteer de fase **Stage0**, selecteer de taak **Task3**, select **uitvoer** en selecteer **responseCode**. Klik vervolgens op **OK**.

rest-ix-1
Enabled
ACTIONS

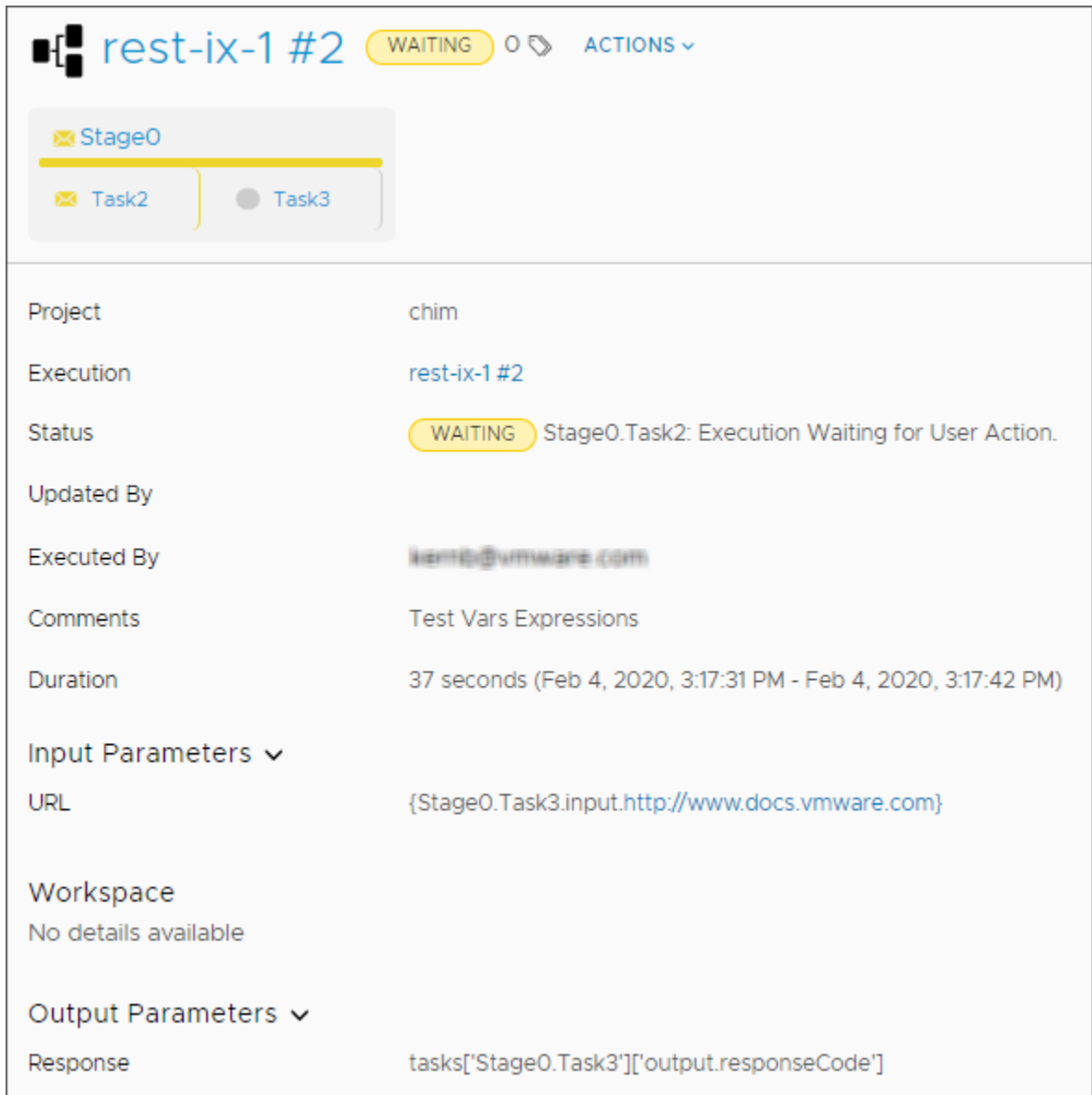
Workspace
Input
Model
Output

Output Parameters ⓘ

ADD

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

- 5 Sla uw pijplijn op.
- 6 Klik in het menu **Acties** op **Uitvoeren**.
- 7 Klik op **Acties > Uitvoeringen bekijken**.
- 8 Klik op de pijplijnuitvoering en bekijk de invoerparameters en uitvoerparameters die u hebt gedefinieerd.



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	kern@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 Als u de pijplijn wilt goedkeuren, klikt u op **Gebruikersbewerkingen** en bekijkt u de lijst met goedkeuringen op het tabblad **Actieve items**. Of blijft in Uitvoeringen, klikt u op de taak en klik vervolgens op **Goedkeuren**.
- 10 Als u de knoppen **Goedkeuren** en **Afwijzen** wilt inschakelen, klikt u op het selectievakje naast de uitvoering.
- 11 Vouw de vervolgkeuzepijl uit om de details weer te geven.
- 12 Als u de taak wilt goedkeuren, klikt u op **GOEDKEUREN**, voert u een reden in en klikt u op **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 Klik op **Uitvoeringen** en kijk hoe de pijplijn verder wordt uitgevoerd.

Executions 3,347 items GUIDED SETUP

+ NEW EXECUTION

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

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

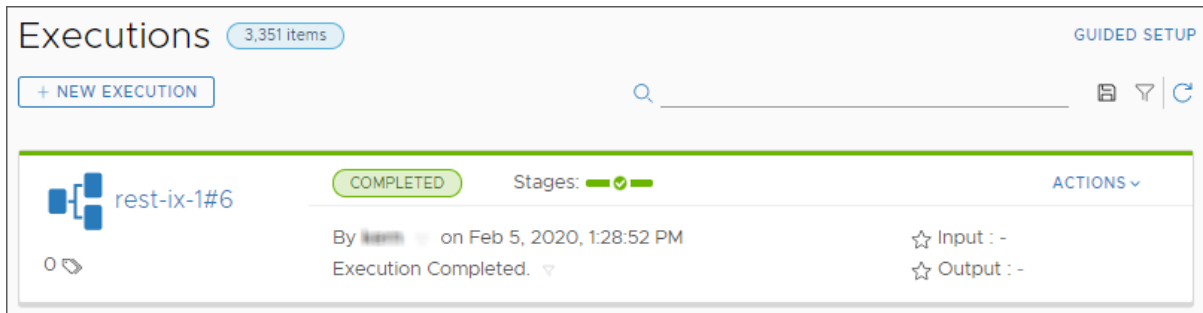
RUNNING

Comments: Testing

☆ Input : -

☆ Output : -

- 14 Als de pijplijn mislukt, corrigeert u eventuele fouten, slaat u de pijplijn op en voert u deze opnieuw uit.



Hoe kan ik meer informatie vinden over variabelen en expressies?

Zie [Welke variabelen en expressies kan ik gebruiken bij het binden van pijplijntaken in Code Stream?](#) voor meer informatie over het gebruik van variabelen en expressies wanneer u pijplijntaken bindt.

Zie [Hoe gebruik ik variabelebindingen in een voorwaardetaak om een pijplijn uit te voeren of te stoppen in Code Stream?](#) voor meer informatie over het gebruik van de pijplijntaakuitvoer met een binding van een voorwaardevariabele.

Hoe gebruik ik variabelebindingen in een voorwaardetaak om een pijplijn uit te voeren of te stoppen in Code Stream?

U kunt de output van een taak in uw pijplijn laten bepalen of de pijplijn wordt uitgevoerd of stopt op basis van een voorwaarde die u opgeeft. Om de pijplijn te laten slagen of mislukken op basis van de uitvoer van de taak, gebruikt u de voorwaardetaak.

U gebruikt de **voorwaardetaak** als een beslissingspunt in uw pijplijn. Met behulp van de Voorwaarde-taak met een voorwaarde-expressie die u opgeeft, kunt u alle properties in uw pijplijn, fasen en taken evalueren.

Het resultaat van de Voorwaarde-taak bepaalt of de volgende taak in de pijplijn wordt uitgevoerd.

- Bij een ware voorwaarde kan de pijplijnuitvoering verder gaan.
- Bij een onware voorwaarde wordt de pijplijn gestopt.

Zie [Hoe gebruik ik variabele bindingen in Code Stream-pijplijnen?](#) voor voorbeelden van hoe u de outputwaarde van de ene taak kunt gebruiken als de input voor de volgende taak door de taken te verbinden met een Voorwaarde-taak.

Tabel 3-7. De relatie tussen de Voorwaarde-taak en diens voorwaarde-expressie en de pijplijn

Voorwaarde-taak	Waarop dit invloed heeft	Wat het doet
Voorwaarde-taak	Pijplijn	De voorwaardetaak bepaalt of de pijplijn wordt uitgevoerd of stopt op dat moment, op basis van het feit of de uitvoer van de taak waar of onwaar is.
Voorwaarde-expressie	Output Voorwaarde-taak	<p>Wanneer de pijplijn wordt uitgevoerd, bepaalt de voorwaarde-expressie die u opneemt in de Voorwaarde-taak of de outputstatus waar of onwaar is. Een voorwaarde-expressie kan bijvoorbeeld vereisen dat de voorwaardetaak de uitvoerstatus <code>Voltooid</code> heeft of het buildnummer 74 gebruiken.</p> <p>De voorwaarde-expressie wordt weergegeven op het tabblad Taak in de voorwaardetaak.</p>

The screenshot shows the configuration interface for a 'Task : Task2'. The 'Type' is set to '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, a table 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)

De **voorwaardetaak** verschilt in functie en gedrag van de instelling **Op voorwaarde** in andere taaktypen.

In andere taaktypen bepaalt **Op voorwaarde** of de huidige taak wordt uitgevoerd, in plaats van opeenvolgende taken, op basis van of de voorwaarde-expressie waar of onwaar retourneert. De voorwaarde-expressie voor instelling **Op voorwaarde** produceert een uitvoerstatus waar of onwaar voor de huidige taak wanneer de pijplijn wordt uitgevoerd. De instelling **Op voorwaarde** wordt op het tabblad Taak weergegeven met een eigen voorwaarde-expressie.

In dit voorbeeld wordt de Voorwaarde-taak gebruikt.

Voorwaarden

- Verifieer dat er een pijplijn bestaat en dat deze fasen en taken bevat.

Procedure

- 1 Bepaal in uw pijplijn het beslissingspunt waar de Voorwaarde-taak moet verschijnen.
- 2 Voeg de voorwaardetaak toe vóór de taak die afhankelijk is van de status geslaagd of mislukt.
- 3 Voeg een voorwaarde-expressie toe aan de Voorwaarde-taak.

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

- 4 Valideer de taak.
- 5 Sla de pijplijn op. Vervolgens schakelt u de pijplijn in en voert u deze uit.

Resultaten

Bekijk de uitvoeringen van de pijplijn en kijk of de pijplijn nog steeds wordt uitgevoerd of stopt bij de voorwaardetaak.

Wat nu te doen

Als u een pijplijnimplementatie terugdraait, kunt u ook de voorwaardetaak gebruiken. Bij het terugdraaien van een pijplijn helpt de Voorwaarde-taak Code Stream bijvoorbeeld om een pijplijnfout te markeren op basis van de voorwaarde-expressie en kan één terugdraaistroom worden geactiveerd voor verschillende fouttypen.

Zie [Hoe kan ik mijn implementatie terugdraaien in Code Stream?](#) om een implementatie terug te draaien.

Welke variabelen en expressies kan ik gebruiken bij het binden van pijplijntaken in Code Stream?

Met variabelen en expressies kunt u invoerparameters en uitvoerparameters met uw pijplijntaken gebruiken. De parameters die u invoert, binden uw pijplijntaak aan een of meer variabelen, expressies of voorwaarden en bepalen het pijplijngedrag wanneer dit wordt uitgevoerd.

Pijplijnen kunnen eenvoudige of complexe softwareleveringsoplossingen uitvoeren

Wanneer u pijplijntaken aan elkaar bindt, kunt u standaard- en complexe expressies opnemen. Als gevolg hiervan kan uw pijplijn eenvoudige of complexe softwareleveringsoplossingen uitvoeren.

Als u de parameters in uw pijplijn wilt maken, klikt u op het tabblad **Invoer** of **Uitvoer** en voegt u een variabele toe door het dollarteken \$ en een expressie in te voeren. Deze parameter wordt bijvoorbeeld gebruikt als taakinvoer die een URL aanroept: `${Stage0.Task3.input.URL}`.

De indeling voor variabele bindingen maakt gebruik van syntaxonderdelen die scopes en sleutels worden genoemd. De `SCOPE` definieert de context als invoer of uitvoer en de `KEY` definieert de details. In het voorbeeld van de parameter `${Stage0.Task3.input.URL}` is de `input` de `SCOPE` en de `URL` de `KEY`.

Uitvoereigenschappen van elke taak kunnen worden omgezet in elk aantal geneste niveaus van een variabelebinding.

Zie [Hoe gebruik ik variabele bindingen in Code Stream-pijplijnen?](#) voor meer informatie over het gebruik van variabele bindingen in pijplijnen.

Dollarexpressies gebruiken met bereiken en sleutels om pijplijntaken te binden

U kunt pijplijntaken samenbinden door expressies in dollartekenvariabelen te gebruiken. U voert expressies in als `${SCOPE.KEY.<PATH>}`.

Om het gedrag van een pijplijntaak te bepalen, is in elke expressie **SCOPE** de context die Code Stream gebruikt. Het bereik zoekt naar een **KEY**, waarbij de details worden gedefinieerd voor de actie die de taak uitvoert. Wanneer de waarde voor **KEY** een genest object is, kunt u een optioneel **PATH** opgeven.

In deze voorbeelden worden **SCOPE** en **KEY** beschreven, en wordt uitgelegd hoe u ze in uw pijplijn kunt gebruiken.

Tabel 3-8. SCOPE en KEY gebruiken

SCOPE	Doel van expressie en voorbeeld	KEY	Hoe u SCOPE en KEY in uw pijplijn kunt gebruiken
invoer	Invoereigenschappen van een pijplijn: <code>\${input.input1}</code>	Naam van de invoereigenschap	<p>Als u wilt verwijzen naar de invoereigenschap van een pijplijn in een taak gebruikt u de volgende indeling:</p> <pre>tasks: mytask: type: REST input: url: \$ {input.url} action: get</pre> <pre>input: url: https:// www.vmware.com</pre>
uitvoer	Uitvoereigenschappen van een pijplijn: <code>\${output.output1}</code>	Naam van de uitvoereigenschap	<p>Als u wilt verwijzen naar een uitvoereigenschap voor het verzenden van een melding, gebruikt u deze indeling:</p> <pre>notifications: email: - endpoint: MyEmailEndpoint subject: "Deployment Successful" event: COMPLETED to: - user@example.org body: Pipeline deployed the service successfully. Refer \$ {output.serviceURL}</pre>

Tabel 3-8. SCOPE en KEY gebruiken (vervolg)

SCOPE	Doel van expressie en voorbeeld	KEY	Hoe u SCOPE en KEY in uw pijplijn kunt gebruiken
taakinvoer	Taakinvoer: \$ {MY_STAGE.MY_TASK.input. SOMETHING}	Geeft de invoer van een taak in een melding aan	<p>Wanneer een Jenkins-taak wordt gestart, kan deze verwijzen naar de naam van de taak die wordt geactiveerd op basis van de taakinvoer. In dit geval verzendt u een melding met de volgende indeling:</p> <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>
taakuitvoer	Taakuitvoer: \$ {MY_STAGE.MY_TASK.output .SOMETHING}	Geeft de uitvoer van een taak in een volgende taak aan	<p>Als u wilt verwijzen naar de uitvoer van pijplijntaak 1 in taak 2, gebruikt u de volgende indeling:</p> <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>

Tabel 3-8. SCOPE en KEY gebruiken (vervolg)

SCOPE	Doel van expressie en voorbeeld	KEY	Hoe u SCOPE en KEY in uw pijplijn kunt gebruiken
var	Variabele: <code>\${var.myVariable}</code>	Verwijst naar variabele in een eindpunt	<p>Als u wilt verwijzen naar een geheime variabele in een eindpunt voor een wachtwoord, gebruikt u de volgende indeling:</p> <pre> --- project: MyProject kind: ENDPOINT name: MyJenkinsServer type: jenkins properties: url: https:// jenkins.example.com username: jenkinsUser password: \$ {var.jenkinsPassword} </pre>
var	Variabele: <code>\${var.myVariable}</code>	Verwijzen naar variabele in een pijplijn	<p>Als u naar de variabele in een pijplijn-URL wilt verwijzen, gebruikt u de volgende indeling:</p> <pre> tasks: task1: type: REST input: action: get url: \$ {var.MY_SERVER_URL} </pre>
taakstatus	Status van een taak: <code>\$</code> <code>{MY_STAGE.MY_TASK.status}</code> <code>}</code> <code>\$</code> <code>{MY_STAGE.MY_TASK.status</code> <code>Message}</code>		
fasestatus	Status van een fase: <code>\${MY_STAGE.status}</code> <code>\$</code> <code>{MY_STAGE.statusMessage}</code>		

Standaardexpressies

U kunt variabelen met expressies gebruiken in uw pijplijn. Deze samenvatting bevat de standaardexpressies die u kunt gebruiken.

Expressie	Beschrijving
<code>\${comments}</code>	Opmerkingen die worden geleverd bij de aanvraag voor pijplijnuitvoering.
<code>\${duration}</code>	Duur van de uitvoering van de pijplijn.
<code>\${endTime}</code>	Eindtijd van de pijplijnuitvoering in UTC, indien voltooid.
<code>\${executedOn}</code>	Hetzelfde als de begintijd: begintijd van de pijplijnuitvoering in UTC.
<code>\${executionId}</code>	ID van de uitvoering van de pijplijn.
<code>\${executionUrl}</code>	URL die naar de pijplijn wordt uitgevoerd in de gebruikersinterface.
<code>\${name}</code>	Naam van de pijplijn.
<code>\${requestBy}</code>	Naam van de gebruiker die de uitvoering heeft aangevraagd.
<code>\${stageName}</code>	Naam van de huidige fase, wanneer deze wordt gebruikt in het bereik van een fase.
<code>\${startTime}</code>	Begintijd van de pijplijnuitvoering in UTC.
<code>\${status}</code>	Status van de uitvoering.
<code>\${statusMessage}</code>	Statusbericht van de uitvoering van de pijplijn.
<code>\${taskName}</code>	Naam van de huidige taak, wanneer deze wordt gebruikt bij een taakinvoer of -melding.

SCOPE en KEY gebruiken in pijplijntaken

U kunt expressies gebruiken met een van de ondersteunde pijplijntaken. In deze voorbeelden ziet u hoe u `SCOPE` en `KEY` kunt definiëren en de syntaxis kunt bevestigen. De codevoorbeelden gebruiken `MY_STAGE` en `MY_TASK` als de pijplijnfase en taaknamen.

Zie [Welke typen taken zijn beschikbaar in Code Stream](#) voor meer informatie over beschikbare taken.

Tabel 3-9. Gating-taken

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
Gebruikersbewerking			
	Input	<p>summary: samenvatting van de aanvraag voor de gebruikersbewerking</p> <p>description: beschrijving van de aanvraag voor de gebruikersbewerking</p> <p>approvers: lijst met e-mailadressen van goedkeurder, waarbij elke vermelding een variabele met een komma kan zijn of een puntkomma voor afzonderlijke e-mails kan gebruiken</p> <p>approverGroups: lijst met adressen van goedkeurersgroep voor het platform en de identiteit</p> <p>sendemail: stuurt desgewenst een e-mailmelding op aanvraag of antwoord wanneer dit is ingesteld op waar</p> <p>expirationInDays: aantal dagen dat de vervaltijd van de aanvraag vertegenwoordigt</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: hexadecimale tekenreeks van 6 cijfers die de aanvraag vertegenwoordigt</p> <p>respondedBy: accountnaam van de persoon die de Gebruikersbewerking heeft goedgekeurd/afgewezen</p> <p>respondedByEmail: e-mailadres van de persoon die heeft gereageerd</p> <p>comments: opmerkingen opgegeven tijdens reactie</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>
Voorwaarde			
	Input	<p>condition: voorwaarde om te evalueren. Wanneer de voorwaarde resulteert in waar, wordt de taak als voltooid gemarkeerd, terwijl andere antwoorden de taak laten mislukken</p>	<pre> \${MY_STAGE.MY_TASK.input.condition} </pre>
	Output	<p>result: resultaat bij evaluatie</p>	<pre> \${MY_STAGE.MY_TASK.output.response} </pre>

Tabel 3-10. Pijplijntaken

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
Pijplijn			
	Input	name: naam van de pijplijn die moet worden uitgevoerd inputProperties: invoereigenschappen die worden doorgegeven aan de uitvoering van de geneste pijplijn	<pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.inputProperties} # verwijzen naar alle eigenschappen \$ {MY_STAGE.MY_TASK.input.inputProperties.inpu t1} # verwijzen naar de waarde van input1 </pre>
	Output	executionStatus: status van de pijplijuitvoering executionIndex: index van de uitvoering van de pijplijn outputProperties: uitvoereigenschappen van de uitvoering van de pijplijn	<pre> \${MY_STAGE.MY_TASK.output.executionStatus} \${MY_STAGE.MY_TASK.output.executionIndex} \${MY_STAGE.MY_TASK.output.outputProperties} # verwijzen naar alle eigenschappen \$ {MY_STAGE.MY_TASK.output.outputProperties.ou tput1} # verwijzen naar de waarde van output1 </pre>

Tabel 3-11. Taken voor continue integratie automatiseren

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
CI			
	Input	steps: een reeks tekenreeksen, waarmee opdrachten worden weergegeven die moeten worden uitgevoerd export: omgevingsvariabelen die behouden moeten worden na het uitvoeren van de stappen artifacts: paden van artefacten die behouden moeten blijven in het gedeelde pad process: reeks configuratie-elementen voor JUnit-, JaCoCo-, Checkstyle-, FindBugs-verwerking	<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 } # verwijzen naar het pad van de eerste configuratie </pre>

Tabel 3-11. Taken voor continue integratie automatiseren (vervolg)

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
	Output	exports: sleutelwaardepaar, dat de geëxporteerde omgevingsvariabelen van de invoer export vertegenwoordigt artifacts: pad van goed bewaarde artefacten processResponse: reeks verwerkte resultaten voor de invoer process	<pre> \${MY_STAGE.MY_TASK.output.exports} # verwijzen naar alle export \$ {MY_STAGE.MY_TASK.output.exports.myvar} # verwijzen naar de waarde van myvar \${MY_STAGE.MY_TASK.output.artifacts} \$ {MY_STAGE.MY_TASK.output.processResponse} \$ {MY_STAGE.MY_TASK.output.processResponse[0].result} # resultaat van de eerste procesconfiguratie </pre>
Custom			
	Input	name: naam van de custom integratie version: een versie van de custom integratie, vrijgegeven of verouderd properties: eigenschappen die naar de custom integratie worden verzonden	<pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.version} \${MY_STAGE.MY_TASK.input.properties} # verwijzen naar alle eigenschappen \$ {MY_STAGE.MY_TASK.input.properties.property1} # verwijzen naar waarde property1 </pre>
	Output	properties: uitvoereigenschappen van de custom integratiereactie	<pre> \${MY_STAGE.MY_TASK.output.properties} # verwijzen naar alle eigenschappen \$ {MY_STAGE.MY_TASK.output.properties.property1} # verwijzen naar waarde property1 </pre>

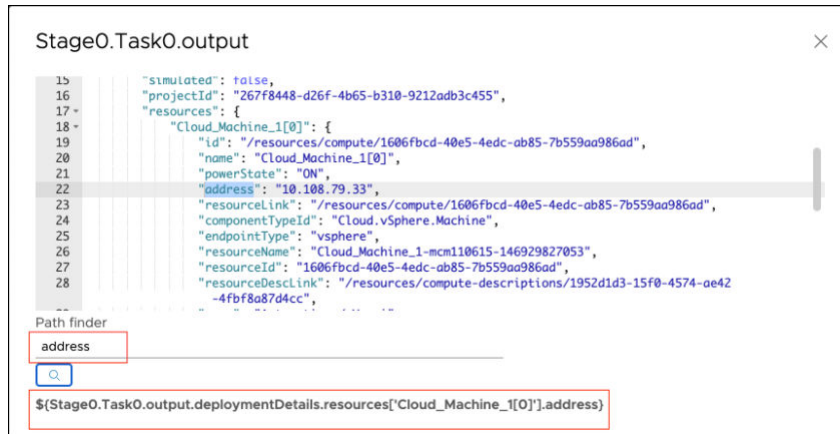
Tabel 3-12. Taken voor continue implementatie automatiseren: cloudsjabloon

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
Cloudsjabloon			
	Input	<p>action: een van createDeployment, updateDeployment, deleteDeployment, rollbackDeployment</p> <p>blueprintInputParams: gebruikt voor de acties Implementatie maken en Implementatie bijwerken</p> <p>allowDestroy: machines kunnen worden vernietigd tijdens het updateproces voor implementaties.</p> <p>CREATE_DEPLOYMENT</p> <ul style="list-style-type: none"> ■ <code>blueprintName</code>: naam van de cloudsjabloon ■ <code>blueprintVersion</code>: versie van de cloudsjabloon <p>OF</p> <ul style="list-style-type: none"> ■ <code>fileUrl</code>: URL van de externe cloudsjabloon-YAML, na het selecteren van een GIT-server. <p>UPDATE_DEPLOYMENT</p> <p>Een van deze combinaties:</p> <ul style="list-style-type: none"> ■ <code>blueprintName</code>: naam van de cloudsjabloon ■ <code>blueprintVersion</code>: versie van de cloudsjabloon <p>OF</p> <ul style="list-style-type: none"> ■ <code>fileUrl</code>: URL van de externe cloudsjabloon-YAML, na het selecteren van een GIT-server. <p>-----</p> <ul style="list-style-type: none"> ■ <code>deploymentId</code>: ID van de implementatie <p>OF</p> <ul style="list-style-type: none"> ■ <code>deploymentName</code>: naam van de implementatie <p>-----</p> <p>DELETE_DEPLOYMENT</p> <ul style="list-style-type: none"> ■ <code>deploymentId</code>: ID van de implementatie 	

Tabel 3-12. Taken voor continue implementatie automatiseren: cloudsjabloon (vervolg)

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
		<p>OF</p> <ul style="list-style-type: none"> ■ <code>deploymentName</code>: naam van de implementatie <p>ROLLBACK_DEPLOYMENT</p> <p>Een van deze combinaties:</p> <ul style="list-style-type: none"> ■ <code>deploymentId</code>: ID van de implementatie <p>OF</p> <ul style="list-style-type: none"> ■ <code>deploymentName</code>: naam van de implementatie <p>-----</p> <ul style="list-style-type: none"> ■ <code>blueprintName</code>: naam van de cloudsjabloon ■ <code>rollbackVersion</code>: versie waarnaar moet worden teruggedraaid 	
	Output		<p>Parameters die kunnen worden gebonden aan andere taken of aan de uitvoer van een pijplijn:</p> <ul style="list-style-type: none"> ■ Implementatienaam kan worden benaderd als <code>{Stage0.Task0.output.deploymentName}</code> ■ Implementatie-id kan worden benaderd als <code>{Stage0.Task0.output.deploymentId}</code> ■ Implementatiedetails is een complex object en interne details kunnen worden geopend met behulp van de JSON-resultaten. <p>Om toegang te krijgen tot een eigenschap, gebruikt u de puntoperator om de JSON-hiërarchie te volgen. Als u bijvoorbeeld toegang wilt tot het adres van resource <code>Cloud_Machine_1[0]</code>, is de <code>\$</code>-binding als volgt:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}</pre> <p>Op dezelfde manier is de <code>\$</code>-binding voor soort als volgt:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}</pre> <p>In de gebruikersinterface van Code Stream kunt u de <code>\$</code>-bindingen voor elke eigenschap verkrijgen.</p> <ol style="list-style-type: none"> 1 Klik in het eigenschapsgebied voor taakuitvoer op Uitvoer-JSON weergeven. 2 Als u de <code>\$</code>-binding wilt vinden, voert u een eigenschap in. 3 Klik op het zoekpictogram, waarin de bijbehorende <code>\$</code>-binding wordt weergegeven.

Voorbeeld van JSON-uitvoer:



Voorbeeld van het object Implementatiedetails:

```

{
  "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"
}
```

Tabel 3-13. Taken voor continue implementatie automatiseren: Kubernetes

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
Kubernetes			
	Input	<p>action: een van GET, CREATE, APPLY, DELETE, ROLLBACK</p> <ul style="list-style-type: none"> ■ timeout: algemene time-out voor acties ■ filterByLabel: extra label waarop moet worden gefilterd voor de actie GET met behulp van K8S labelSelector: <p>GET, CREATE, DELETE, APPLY</p> <ul style="list-style-type: none"> ■ yaml: inline YAML voor verwerking en verzending naar Kubernetes ■ parameters: paar SLEUTEL, WAARDE - vervang \$\$KEY door WAARDE in het inline YAML-invoergebied ■ filePath: het relatieve pad van het SCM Git-eindpunt, indien opgegeven, vanwaar de YAML moet worden opgehaald ■ scmConstants: paar SLEUTEL, WAARDE - vervang \$\$KEY door WAARDE in de YAML die is opgehaald via SCM. ■ continueOnConflict: wanneer dit is ingesteld op "waar", wordt de taak voortgezet als er al een resource is. <p>ROLLBACK</p> <ul style="list-style-type: none"> ■ resourceType: resourcetype om terug te draaien ■ resourceName: resourcenaam om terug te draaien ■ namespace: naamruimte waar de terugdraaiactie moet worden uitgevoerd ■ revision: herziening waarnaar moet worden teruggedraaid 	<p><code>\${MY_STAGE.MY_TASK.input.action}</code> # Bepaalt de actie die moet worden uitgevoerd.</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>\$</p> <p><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: hiermee wordt het volledige antwoord vastgelegd</p> <p><code>response.<RESOURCE></code>: resource komt overeen met configMaps, implementaties, eindpunten, ingresses, taken, naamruimten, pods, replicaSets, replicationControllers, geheimen, services, statefulSets, knooppunten, loadBalancers.</p> <p><code>response.<RESOURCE>.<KEY></code>: de sleutel komt overeen met een van apiVersion, soort, metagegevens, specificatie</p>	<p><code>\${MY_STAGE.MY_TASK.output.response}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.response.}</code></p>

Tabel 3-14. Ontwikkelings-, test- en implementatieapplicaties integreren

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
Bamboo			
	Input	plan: naam van het plan planKey: plansleutel variables: variabelen die moeten worden doorgegeven aan het plan parameters: parameters die moeten worden doorgegeven aan het plan	<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} # verwijzen naar alle parameters \${MY_STAGE.MY_TASK.input.parameters.param1} # verwijzen naar de waarde van param1 </pre>
	Output	resultUrl: URL van de resulterende build buildResultKey: sleutel van de resulterende build buildNumber: buildnummer buildTestSummary: samenvatting van de uitgevoerde tests successfulTestCount: testresultaat geslaagd failedTestCount: testresultaat mislukt skippedTestCount: testresultaat overgeslagen artifacts: artefacten van de build	<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} # verwijzen naar alle resultaten \${MY_STAGE.MY_TASK.output.successfulTestCount} # verwijzen naar de specifieke testelling \${MY_STAGE.MY_TASK.output.buildNumber} </pre>
Jenkins			
	Input	job: naam van de Jenkins-taak parameters: parameters die aan de taak moeten worden doorgegeven	<pre> \${MY_STAGE.MY_TASK.input.job} \${MY_STAGE.MY_TASK.input.parameters} # verwijzen naar alle parameters \${MY_STAGE.MY_TASK.input.parameters.param1} # verwijzen naar de waarde van een parameter </pre>
	Output	job: naam van de Jenkins-taak jobId: ID van de resulterende taak, zoals 1234 jobStatus: status in Jenkins jobResults: resultaten van de verzameling van test- en code dekking jobUrl: URL van de resulterende taakuitvoering	<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} # verwijzen naar alle resultaten \${MY_STAGE.MY_TASK.output.jobResults.junitResponse} # verwijzen naar JUnit-resultaten \${MY_STAGE.MY_TASK.output.jobResults.jacocoResponse} # verwijzen naar JaCoCo-resultaten \${MY_STAGE.MY_TASK.output.jobUrl} </pre>
TFS			

Tabel 3-14. Ontwikkelings-, test- en implementatieapplicaties integreren (vervolg)

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
	Input	projectCollection: projectverzameling vanuit TFS teamProject: geselecteerd project uit de beschikbare verzameling buildDefinitionId: definitie-id van de build om uit te voeren	\${MY_STAGE.MY_TASK.input.projectCollection} \${MY_STAGE.MY_TASK.input.teamProject} \${MY_STAGE.MY_TASK.input.buildDefinitionId}
	Output	buildId: resulterende build-ID buildUrl: URL om samenvatting van build te bezoeken logUrl: URL om te bezoeken voor logboeken dropLocation: locatie van eventuele artefacten laten vallen	\${MY_STAGE.MY_TASK.output.buildId} \${MY_STAGE.MY_TASK.output.buildUrl} \${MY_STAGE.MY_TASK.output.logUrl} \${MY_STAGE.MY_TASK.output.dropLocation}
vRO			
	Input	workflowId: id van de werkstroom die moet worden uitgevoerd parameters: parameters die moeten worden doorgegeven aan de werkstroom	\${MY_STAGE.MY_TASK.input.workflowId} \${MY_STAGE.MY_TASK.input.parameters}
	Output	workflowExecutionId: ID van de uitvoering van de werkstroom properties: uitvoereigenschappen van de uitvoering van de werkstroom	\${MY_STAGE.MY_TASK.output.workflowExecutionId} \${MY_STAGE.MY_TASK.output.properties}

Tabel 3-15. Andere applicaties integreren met behulp van een API

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
REST			
	Input	url: URL die moet worden uitgevoerd action: HTTP-methode die moet worden gebruikt headers: HTTP-kopteksten om door te geven payload: nettolading aanvragen fingerprint: vingerafdruk om te koppelen als de URL https is allowAllCerts: wanneer ingesteld op Waar, kan elk certificaat zijn met een URL van https	\${MY_STAGE.MY_TASK.input.url} \${MY_STAGE.MY_TASK.input.action} \${MY_STAGE.MY_TASK.input.headers} \${MY_STAGE.MY_TASK.input.payload} \${MY_STAGE.MY_TASK.input.fingerprint} \${MY_STAGE.MY_TASK.input.allowAllCerts}

Tabel 3-15. Andere applicaties integreren met behulp van een API (vervolg)

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
	Output	<p>responseCode: HTTP-reactiecode</p> <p>responseHeaders: HTTP-reactiekopecteksten</p> <p>responseBody: indeling tekenreeks van ontvangen reactie</p> <p>responseJson: traversable-reactie als het inhoudstype application/json is</p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseHeaders}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.output.responseHeaders.header1} # verwijzen naar de reactiekopectekst 'header1'</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson} # verwijzen naar reactie als JSON</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson.a.b.c} # verwijzen naar het geneste object na het a.b.c JSON-pad in de reactie</code></p>
Poll			
	Input	<p>url: URL die moet worden uitgevoerd</p> <p>headers: HTTP-kopecteksten om door te geven</p> <p>exitCriteria: criteria waaraan moet worden voldaan om de taak te laten slagen of mislukken Een sleutelwaardepaar van 'geslaagd' → Expressie, 'fout' → Expressie</p> <p>pollCount: aantal iteraties dat moet worden uitgevoerd. Een Code Stream-beheerder kan het aantal polls instellen op een maximum van 10.000.</p> <p>pollIntervalSeconds: aantal seconden dat moet worden gewacht tussen elke iteratie. Het pollinginterval moet 60 seconden of meer zijn.</p> <p>ignoreFailure: als dit is ingesteld op waar, worden tussenliggende reactiefouten genegeerd</p> <p>fingerprint: vingerafdruk om te koppelen als de URL https is</p> <p>allowAllCerts: wanneer ingesteld op Waar, kan elk certificaat zijn met een URL van https</p>	<p><code>\${MY_STAGE.MY_TASK.input.url}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.headers}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.exitCriteria}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollCount}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollIntervalSeconds}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.ignoreFailure}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.fingerprint}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code></p>
	Output	<p>responseCode: HTTP-reactiecode</p> <p>responseBody: indeling tekenreeks van ontvangen reactie</p> <p>responseJson: traversable-reactie als het inhoudstype application/json is</p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson} # Refer to response as JSON</code></p>

Tabel 3-16. Externe en door de gebruiker gedefinieerde scripts uitvoeren

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
PowerShell Om een PowerShell-taak uit te voeren, moet u het volgende doen: <ul style="list-style-type: none"> ■ Een actieve sessie met een externe Windows-host hebben. ■ Als u van plan bent een base64 PowerShell-opdracht in te voeren, berekent u eerst de algemene opdracht lengte. Zie Welke typen taken zijn beschikbaar in Code Stream voor meer informatie. 			
	Invoer	host: IP-adres of hostnaam van de machine username: gebruikersnaam die moet worden gebruikt voor de verbinding password: wachtwoord dat moet worden gebruikt om verbinding te maken useTLS: https-verbinding proberen trustCert: als dit is ingesteld op Waar, vertrouwt u zelfondertekende certificaten script: uit te voeren script workingDirectory: mappad waarnaar moet worden geschakeld voordat het script wordt uitgevoerd environmentVariables: een sleutelwaardepaar van de omgevingsvariabele die u wilt instellen arguments: argumenten om aan het script door te geven	<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>
	Uitvoer	response: inhoud van het bestand \$SCRIPT_RESPONSE_FILE responseFilePath: waarde van \$SCRIPT_RESPONSE_FILE exitCode: afsluitcode van het proces logFilePath: pad naar bestand met stdout errorFilePath: pad naar bestand met 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>
Secure Shell			

Tabel 3-16. Externe en door de gebruiker gedefinieerde scripts uitvoeren (vervolg)

Taak	Scope	Key	Hoe kan ik het SCOPE en de KEY in de taak gebruiken?
	Input	host: IP-adres of hostnaam van de machine username: gebruikersnaam die moet worden gebruikt voor de verbinding password: wachtwoord dat u wilt gebruiken om verbinding te maken (optioneel: privateKey kan ook worden gebruikt) privateKey: PrivateKey gebruiken om verbinding te maken passphrase: optionele wachtwoordzin om privateKey te ontgrendelen script: uit te voeren script workingDirectory: mappad waarnaar moet worden geschakeld voordat het script wordt uitgevoerd environmentVariables: sleutelwaardepaar van de omgevingsvariabele die moet worden ingesteld	<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.workingDirectory} } \$ {MY_STAGE.MY_TASK.input.environmentVariables} </pre>
	Output	response: inhoud van het bestand \$SCRIPT_RESPONSE_FILE responseFilePath: waarde van \$SCRIPT_RESPONSE_FILE exitCode: afsluitcode van het proces logFilePath: pad naar bestand met stdout errorFilePath: pad naar bestand met 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>

Een variabelebinding tussen taken gebruiken

In dit voorbeeld ziet u hoe u variabelebindingen in uw pijplijntaken kunt gebruiken.

Tabel 3-17. Voorbeelden van syntaxisindelingen

Voorbeeld	Syntaxis
Een taakuitvoerwaarde gebruiken voor pijplijnmeldingen en uitvoereigenschappen van de pijplijn	<code>\${<Stage Key>.<Task Key>.output.<Task output key>}</code>
Als u wilt verwijzen naar de vorige taakuitvoerwaarde als invoer voor de huidige taak	<code>\${<Previous/Current Stage key>.<Previous task key not in current Task group>.output.<task output key>}</code>

Meer informatie

Voor meer informatie over het binden van variabelen in taken zie:

- [Hoe gebruik ik variabele bindingen in Code Stream-pijplijnen?](#)
- [Hoe gebruik ik variabelebindingen in een voorwaardetaak om een pijplijn uit te voeren of te stoppen in Code Stream?](#)
- [Welke typen taken zijn beschikbaar in Code Stream](#)

Hoe verzend ik meldingen over mijn pijplijn in Code Stream?

Meldingen zijn manieren om met uw teams te communiceren en hen de status van uw pijplijnen in Code Stream te laten weten.

Om meldingen te verzenden wanneer een pijplijn wordt uitgevoerd, kunt u Code Stream-meldingen configureren op basis van de status van de volledige pijplijn, fase of taak.

- Een e-mailmelding stuurt een e-mail bij:
 - Voltooing, wachten, mislukking, annulering of starten van pijplijn.
 - Voltooing, mislukken of starten van fase.
 - Voltooing, wachten, mislukken of starten van taak.
- Een ticketmelding maakt een ticket en wijst dit toe aan een teamlid wanneer:
 - Pijplijn is mislukt of voltooid.
 - Fase mislukt.
 - Taak mislukt.
- Een webhook-melding stuurt een aanvraag naar een andere applicatie bij:
 - Mislukking, voltooing, wachten, annulering of starten van pijplijn.
 - Mislukking, voltooing of start van fase.
 - Mislukking, voltooing, wachten of starten van taak.

U kunt bijvoorbeeld een e-mailmelding voor een gebruikersbewerkingstaak configureren om goedkeuring te verkrijgen op een specifiek punt in uw pijplijn. Wanneer de pijplijn wordt uitgevoerd, verzendt deze taak een e-mail naar de persoon die de taak moet goedkeuren. Als voor de taak Gebruikersbewerking de time-out voor de verlooptijd is ingesteld op dagen, uren of minuten, moet de vereiste gebruiker de pijplijn goedkeuren voordat de taak verloopt. Anders mislukt de pijplijn zoals verwacht.

Als u een Jira-ticket wilt maken wanneer een pijplijntaak mislukt, kunt u een melding configureren. Of, als u een aanvraag naar een Slack-kanaal verzendt over de status van een pijplijn op basis van de pijplijngebeurtenis, kunt u een webhookmelding configureren.

U kunt variabelen in alle typen meldingen gebruiken. U kunt bijvoorbeeld `${var}` in de URL van een webhook-melding gebruiken.

Voorwaarden

- Verifieer dat een of meerdere pijplijnen zijn gemaakt. Zie de toepassingsvoorbeelden in [Hoofdstuk 5 Tutorials voor het gebruik van Code Stream](#).
- Controleer of u toegang hebt tot een werkende e-mailserver om e-mailmeldingen te verzenden. Neem contact op met uw beheerder voor hulp.
- Om tickets te maken, zoals een Jira-ticket, bevestigt u dat het eindpunt bestaat. Zie [Wat zijn eindpunten in Code Stream?](#)
- Als u een melding wilt verzenden op basis van een integratie, maakt u een webhook-melding. Vervolgens bevestigt u dat de webhook is toegevoegd en werkt. U kunt meldingen gebruiken met applicaties zoals Slack, GitHub of GitLab.

Procedure

- 1 Open een pijplijn.
- 2 Om een melding te maken voor de algehele pijplijnstatus of de status van een fase of taak:

Een melding maken voor:	Wat u doet:
Pijplijnstatus	Klik op een leeg gebied op het pijplijncanvas.
Status van een fase	Klik op een leeg gebied in een fase van de pijplijn.
Status van een taak	Klik op een taak in een fase van de pijplijn.

- 3 Klik op het tabblad **Meldingen**.
- 4 Klik op **Toevoegen**, selecteer het type melding en configureer de meldingsdetails.
- 5 Om een Slack-melding te maken wanneer een pijplijn slaagt, maakt u een webhook-melding.
 - a Selecteer **Webhook**.
 - b Voer de informatie in om de Slack-melding te configureren.

- c Klik op **Opslaan**.
- d Wanneer de pijplijn wordt uitgevoerd, ontvangt het Slack-kanaal de melding van de pijplijnstatus. Gebruikers kunnen bijvoorbeeld het volgende zien op het Slack-kanaal:

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

- 6 Als u een JIRA-ticket wilt maken, configureert u de ticketgegevens.
 - a Selecteer **Ticket**.
 - b Voer de informatie in om de Jira-melding te configureren.
 - c Klik op **Opslaan**.

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

Resultaten

Gefeliciteerd! U heeft geleerd dat u verschillende typen meldingen in verschillende gebieden van uw pijplijn in Code Stream kunt maken.

Wat nu te doen

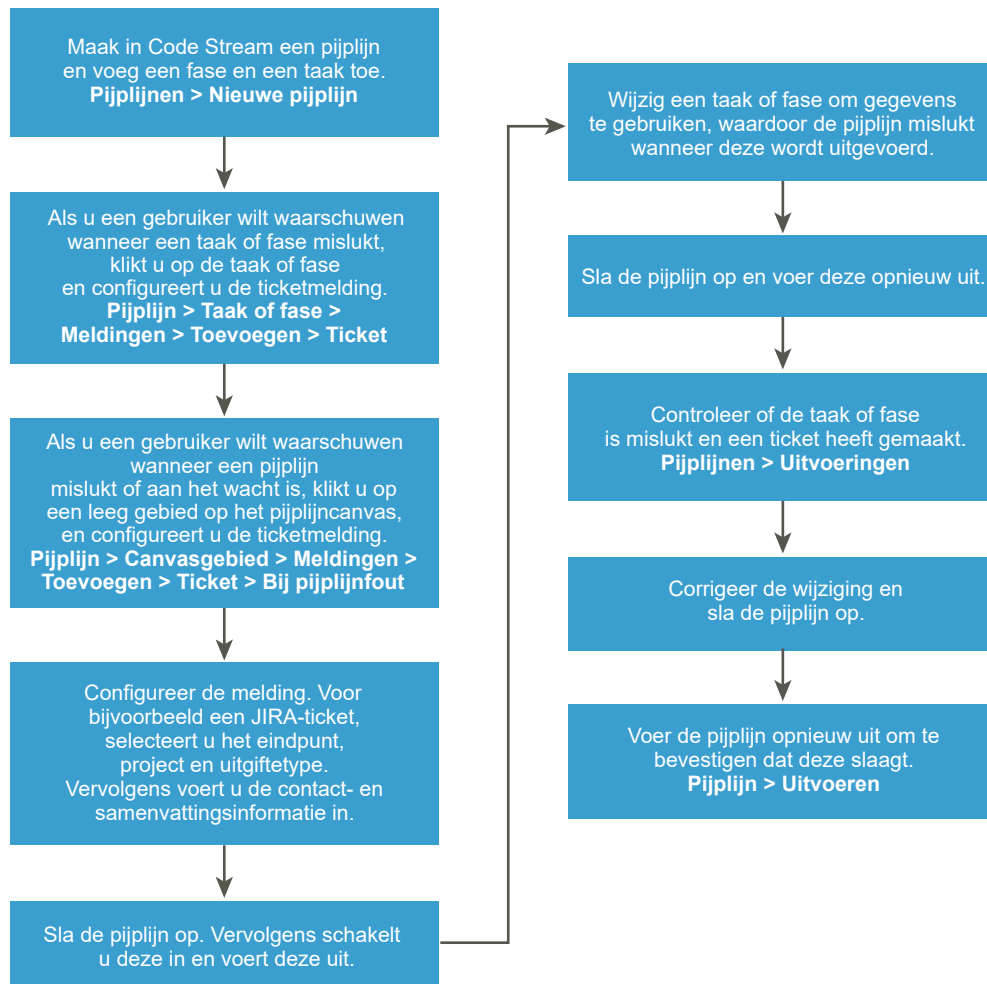
Zie [Hoe maak ik een JIRA-ticket in Code Stream wanneer een pijplijntaak mislukt?](#) voor een gedetailleerd voorbeeld van hoe u een melding kunt maken.

Hoe maak ik een JIRA-ticket in Code Stream wanneer een pijplijntaak mislukt?

Als een fase of taak in uw pijplijn mislukt, kunt u Code Stream een JIRA-ticket laten maken. U kunt de ticket toewijzen aan de persoon die het probleem moet oplossen. U kunt ook een ticket maken wanneer de pijplijn aan het wachten is of wanneer deze slaagt.

U kunt meldingen voor een taak, fase of pijplijn toevoegen en configureren. Code Stream maakt de ticket op basis van de status van de taak, fase of pijplijn waaraan u de melding toevoegt. Als een eindpunt bijvoorbeeld niet beschikbaar is, kunt u Code Stream een JIRA-ticket laten maken voor de taak die mislukt omdat deze geen verbinding kan maken met het eindpunt.

U kunt ook meldingen maken wanneer uw pijplijn slaagt. U kunt bijvoorbeeld uw QA-team informeren over gelukte pijplijnen zodat ze de build kunnen bevestigen en een andere testpijplijn kunnen uitvoeren. Of u kunt uw prestatieteam informeren zodat ze de prestaties van de pijplijn kunnen meten ter voorbereiding voor een update van de voorbereide inrichting of productie.



In dit voorbeeld wordt een JIRA-ticket gemaakt wanneer een pijplijntaak mislukt.

Voorwaarden

- Controleer of u een geldig JIRA-account heeft en of u zich kunt aanmelden bij uw JIRA-instantie.
- Controleer of er een JIRA-eindpunt bestaat en of het werkt.

Procedure

- 1 Klik in uw pijplijn op een taak.
- 2 Klik in het gebied voor taakconfiguratie op **Meldingen**.
- 3 Klik op **Toevoegen** en configureer de ticketgegevens.
 - a Klik op **Ticket**.
 - b Selecteer het JIRA-eindpunt.
 - c Voer het Jira-project en type probleem in.
 - d Voer het e-mailadres in van degene die de ticket ontvangt.
 - e Voer een samenvatting en beschrijving van de ticket in en klik vervolgens op **Opslaan**.

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 Sla de pijplijn op. Vervolgens schakelt u de pijplijn in en voert u deze uit.

5 Test de ticket.

- a Wijzig de taakgegevens om gegevens op te nemen die ervoor zorgen dat de taak mislukt.
- b Sla de pijplijn op en voer deze opnieuw uit.
- c Klik op **Uitvoeringen** en bevestig dat de pijplijn is mislukt.
- d Bevestig in de uitvoering of Code Stream de ticket heeft gemaakt en verzonden.
- e Wijzig de taakgegevens terug om deze te corrigeren, voer de pijplijn vervolgens opnieuw uit en controleer of deze slaagt.

Resultaten

Gefeliciteerd! U hebt Code Stream een JIRA-ticket laten maken toen de pijplijntaak mislukte en hebt het toegewezen aan de persoon die het probleem moest oplossen.

Wat nu te doen

Blijf meldingen toevoegen om uw team te waarschuwen over uw pijplijnen.

Hoe kan ik mijn implementatie terugdraaien in Code Stream?

U configureert het terugdraaien als een pijplijn met taken die uw implementatie retourneren naar een eerdere stabiele status na een fout in een implementatiepijplijn. Als u wilt terugdraaien nadat een fout is opgetreden, voegt u de terugdraai pijplijn toe aan taken of fasen.

Afhankelijk van uw rol kunnen uw redenen om terug te draaien verschillen.

- Als release-engineer wil ik dat Code Stream het succes kan controleren tijdens een release, zodat ik kan weten of de release moet worden voortgezet of teruggedraaid. Mogelijke fouten zijn onder meer het mislukken van een taak, een weigering in UserOps en het overschrijden van de drempelwaarde voor metriecken.
- Als eigenaar van een omgeving wil ik een vorige versie opnieuw implementeren zodat ik snel een omgeving kan terugdraaien naar een status waarvan ik weet dat die goed is.
- Als eigenaar van een omgeving wil ik het terugdraaien van een Blue-Green-implementatie ondersteunen zodat de uitvaltijd door mislukte releases kan worden geminimaliseerd.

Wanneer u een slimme pijplijnsjabloon gebruikt om een CD-pijplijn te maken waarbij de terugdraai-optie is aangeklikt, wordt het terugdraaien automatisch toegevoegd aan de taken in de pijplijn. In dit gebruiksscenario gebruikt u de slimme pijplijnsjabloon om het terugdraaien voor een applicatie-implementatie op een Kubernetes-cluster te definiëren met behulp van het doorlopende upgrade-implementatiemodel. De slimme pijplijnsjabloon maakt een implementatiepijplijn en een of meer terugdraai pijplijnen.

- In de implementatiepijplijn is terugdraaien vereist als de taken Implementatie bijwerken of Implementatie verifiëren mislukken.
- In de terugdraai pijplijn wordt de implementatie bijgewerkt met een oude image.

U kunt ook handmatig een terugdraai pijlijn maken met behulp van een lege sjabloon. Voordat u een terugdraai pijlijn maakt, moet u uw terugdraai stroom plannen. Zie [Plannen voor terugdraaien in Code Stream](#) voor meer achtergrondinformatie over terugdraaien.

Voorwaarden


- Controleer of u lid bent van een project in Code Stream. Als u geen beheerder bent, vraagt u de Code Stream-beheerder om u als lid toe te voegen aan een project. Zie [Hoe voeg ik een project toe in Code Stream?](#).
- Stel de Kubernetes-clusters in waarop uw pijlijn uw applicatie zal implementeren. Stel één ontwikkelingscluster en één productiecluster in.
- Controleer of u een Docker-register heeft ingesteld.
- Identificeer een project dat al uw werk groepeer, inclusief uw pijlijn, eindpunten en dashboards.
- Raak vertrouwd met de slimme CD-sjabloon zoals beschreven in het CD-gedeelte van [Een systeemeigen CICD-build plannen in Code Stream voordat u de slimme pijlijnsjabloon gebruikt](#), bijvoorbeeld:
 - Maak de Kubernetes-eindpunten voor ontwikkeling en productie die uw applicatie-image implementeren op de Kubernetes-clusters.
 - Bereid het Kubernetes YAML-bestand voor dat de naamruimte, service en implementatie maakt. Als u een image moet downloaden uit een privéopslagplaats, moet het YAML-bestand een sectie bevatten met het Docker-configuratiegeheim.


Procedure

- 1 Klik op **Pijlijnen > Nieuwe pijlijn > Slimme sjabloon > Continue levering**.
- 2 Voer de informatie in de slimme pijlijnsjabloon in.
 - a Selecteer een project.
 - b Voer een pijlijnnaam in, zoals **DoorlopendeUpgrade-Voorbeeld**.
 - c Selecteer de omgevingen voor uw applicatie. Om terugdraaien aan uw implementatie toe te voegen, moet u **Productie** selecteren.
 - d Klik op **Selecteren**, kies een Kubernetes-YAML-bestand en klik op **Verwerken**.
In de slimme pijlijnsjabloon worden de beschikbare services en implementatieomgevingen weergegeven.
 - e Selecteer de service die door de pijlijn wordt gebruikt voor de implementatie.
 - f Selecteer de clustereindpunten voor de ontwikkelingsomgeving en de productieomgeving.
 - g Selecteer als imagebron **Runtime-input voor pijlijn**.
 - h Selecteer als Implementatiemodel **Doorlopende upgrade**.


- i Klik op **Terugdraaien**.
- j Geef de **Gezondheidscontrole-URL** op.

Smart Template: Continuous Delivery

Endpoint prerequisites  Kubernetes Docker Registry


Project * 

Pipeline name *

Environment  * ☒ Development ☒ Production

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

Select service

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

1 services

Deployment

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


Image source * ☐ Docker trigger ☒ Pipeline runtime input

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

Rollback ☒

Health check URL *

CREATE CANCEL



- 3 Om de pijplijn met de naam TerugdraaiUpgrade-Voorbeeld te maken, klikt u op **Maken**.

De pijplijn met de naam TerugdraaiUpgrade-Voorbeeld wordt weergegeven en het terugdraaipictogram wordt weergegeven op taken die kunnen terugdraaien in de ontwikkelingsfase en de productiefase.

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 Notifications Rollback **VALIDATE TASK**

Task name: Create Secret

Type: Kubernetes

Continue on failure: ☐

Execute task: ☒ Always ☐ On condition

Kubernetes Task Properties

Kubernetes cluster: Dev-VKE-Cluster

Timeout (in Mins): 5

Action: ☐ Get ☒ Create ☐ Apply ☐ Delete ☐ Rollback

Continue on conflict: ☐

Payload source: ☐ Source control ☒ Local definition

Local YAML definition:

```
1 apiVersion: v1
2 data:
3   .dockercfg: eyJ2ZW50aG9ueS10YV5nb311ZXh0M15qZnJvZy5pb311c
4   2VybWVZ5161nHbmdvLW10dGEy11w1c0Fzc30vcnQ101Jm8tcmV0LW
5   1Uq111e3-c11c11bWp1c11nHbmdvLW10dGEy11w1c0Fzc30vcnQ101Jm8tcmV0LW
6   1Uq111e3-c11c11bWp1c11nHbmdvLW10dGEy11w1c0Fzc30vcnQ101Jm8tcmV0LW
7   1Uq111e3-c11c11bWp1c11nHbmdvLW10dGEy11w1c0Fzc30vcnQ101Jm8tcmV0LW
8   1Uq111e3-c11c11bWp1c11nHbmdvLW10dGEy11w1c0Fzc30vcnQ101Jm8tcmV0LW
```

Parameters: **ADD** **DELETE**

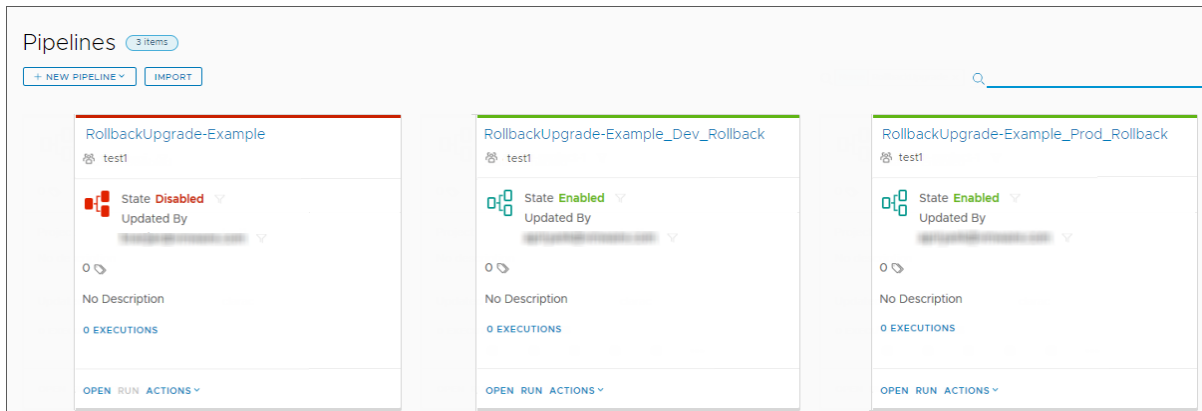
Output Parameters: **status** **response**

EDIT **RUN** **CLOSE** Last saved 9 minutes ago

4 Sluit de pijplijn.

Op de pagina Pijplijnen wordt de pijplijn weergegeven die u hebt gemaakt en wordt een nieuwe pijplijn weergegeven voor elke fase in uw pijplijn.

- DoorlopendeUpgrade-Voorbeeld. Code Stream deactiveert de pijplijn die u standaard heeft gemaakt. Dit zorgt ervoor dat u de pijplijn controleert voordat u deze uitvoert.
- DoorlopendeUpgrade-Voorbeeld_Ontw_Terugdraaien. Bij het mislukken van taken in de ontwikkelingsfase, zoals **Service maken**, **Geheim maken**, **Implementatie maken** en **Implementatie verifiëren**, wordt deze terugdraai-ontwikkelingspijplijn aangeroepen. Om het terugdraaien van ontwikkelingstaken te garanderen, schakelt Code Stream de pijplijn voor het terugdraaien van ontwikkelingsstappen standaard in.
- DoorlopendeUpgrade-Voorbeeld_Prod_Terugdraaien. Bij het mislukken van taken in de productiefase, zoals **Fase 1 implementeren**, **Fase 1 verifiëren**, **Fase Terugdraai implementeren**, **Fase Terugdraai voltooien** en **Fase Terugdraai verifiëren**, deze productiepijplijn voor terugdraaien aanroepen. Om het terugdraaien van productietaken te garanderen, schakelt Code Stream de productiepijplijn voor terugdraaien standaard in.



- 5 Schakel de pijplijn in die u heeft gemaakt in en voer deze uit.

Wanneer u de uitvoering start, wordt u in Code Stream gevraagd om invoerparameters. U geeft de image en tag op voor het eindpunt in de Docker-opslagplaats die u gebruikt.

- 6 Selecteer **Acties > Uitvoering bekijken** op de pagina Uitvoeringen en bekijk de pijplijnuitvoering.

De pijplijn start met **UITVOEREN...** en gaat door de taken in de ontwikkelingsfase. Als het uitvoeren van een taak door de pijplijn tijdens de ontwikkelingsfase mislukt, wordt de pijplijn met de naam DoorlopendeUpgrade-Voorbeeld_Ontw_Terugdraaien geactiveerd en wordt de implementatie teruggedraaid, en wordt de status van de pijplijn gewijzigd in **TERUGDRAAIEN**.

[< BACK](#)

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

[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 ▾
 image demo-image-cs
 tag latest
Workspace
 Details not available
Output Parameters ▾
 The Execution did not output any properties

Na het terugdraaien vermeldt de pagina Uitvoeringen twee uitvoeringen van de DoorlopendeUpgrade-Voorbeeld-pijplijnen.

- De pijplijn die u hebt gemaakt, wordt teruggedraaid en geeft **TERUGDRAAIEN_VOLTOOID** weer.
- De ontwikkelingspijplijn voor terugdraaien die het terugdraaien heeft geactiveerd en uitgevoerd, geeft **VOLTOOID** weer.

Executions 604 items

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

RollbackUpgrade-Example_Dev... #1 COMPLETED Stages: 1/2

1 [Rollback for RollbackUpgrade-Example#1](#)
 By started on 01/11/2019 1:36 PM
 Execution Completed.
Comments: Triggered to rollback Development. Create Deployment of RollbackUpgrade-Example#1

RollbackUpgrade-Example#1 ROLLBACK_COMPLETED Stages: 1/2

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

Resultaten

Gefeliciteerd! U heeft een pijplijn met terugdraaifunctie gedefinieerd en heeft gezien hoe Code Stream de pijplijn heeft teruggedraaid op het punt van mislukken.

Plannen om op een systeemeigen manier uw code te bouwen, integreren en leveren in Code Stream

Plan uw systeemeigen build voordat u Code Stream uw code laat bouwen, integreren en leveren met behulp van de systeemeigen mogelijkheid om een CICD-, CI- of CD-pijplijn voor u te maken. Vervolgens kunt u een pijplijn maken met behulp van een van de slimme pijplijnsjablonen of door fases en taken handmatig toe te voegen.

Om te plannen voor uw build met continue integratie en continue levering, hebben we verschillende voorbeelden toegevoegd die u laten zien hoe. In deze plannen worden de vereisten en overzichten beschreven die u kunnen helpen bij het voorbereiden en gebruiken van de native buildmogelijkheid wanneer u uw pijplijnen bouwt.

Dit hoofdstuk omvat de volgende onderwerpen:

- De pijplijnwerkplek configureren
- Een systeemeigen CICD-build plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt
- Een native build voor continue integratie plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt
- Een native build voor continue levering plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt
- Een CICD-systeemeigen build plannen in Code Stream voordat u handmatig taken toevoegt
- Plannen voor terugdraaien in Code Stream

De pijplijnwerkplek configureren

Om taken voor continue integratie en aangepaste taken uit te voeren, moet u een werkplek voor uw Code Stream-pijplijn configureren.

Selecteer in de pijplijnwerkplek Docker of Kubernetes als **Type** en geef het betreffende eindpunt op. De Docker- en Kubernetes-platforms beheren de gehele levenscyclus van de container die Code Stream implementeert voor het uitvoeren van de taak voor continue integratie (CI) of de aangepaste taak.

- Voor de Docker-werkplek zijn het Docker-hosteindpunt, de image-URL van de opbouwfunctie, het imageregister, de werkdirectory, de cache, omgevingsvariabelen, de CPU-limiet en de geheugenlimiet vereist. U kunt ook een kloon van de Git-opslagplaats maken.
- Voor de Kubernetes-werkplek zijn het Kubernetes API-eindpunt, de image-URL van de opbouwfunctie, het imageregister, de naamruimte, de NodePort, de permanente volumeclaim (PVC), de werkdirectory, de omgevingsvariabelen, de CPU-limiet en de geheugenlimiet vereist. U kunt ook een kloon van de Git-opslagplaats maken.

De configuratie van de pijplijnwerkplek heeft veel algemene parameters en andere parameters die specifiek zijn voor het type werkplek, zoals in de volgende tabel wordt beschreven.

Tabel 4-1. Werkplekgebieden, gegevens en beschikbaarheid

Selectie	Beschrijving	Gegevens en beschikbaarheid
Type	Type werkplek	Beschikbaar met Docker of Kubernetes.
Hosteindpunt	Hosteindpunten waar de taken voor continue integratie (CI) en aangepaste taken worden uitgevoerd.	Beschikbaar in de Docker-werkplek wanneer u het Docker-hosteindpunt selecteert. Beschikbaar met de Kubernetes-werkplek wanneer u het Kubernetes API-eindpunt selecteert.
Image-URL van opbouwfunctie	Naam en locatie van de image van de opbouwfunctie. Er wordt een container gemaakt met behulp van deze image op de Docker-host en het Kubernetes-cluster. De taken voor continue integratie (CI) en aangepaste taken worden binnen deze container uitgevoerd.	Voorbeeld: fedora:latest De image van de opbouwfunctie moet curl of wget bevatten.
Imageregister	Als de image van de opbouwfunctie beschikbaar is in een register en als het register verificatiegegevens vereist, moet u een imageregistereindpunt maken, en het vervolgens hier selecteren zodat de image uit het register kan worden opgehaald.	Beschikbaar met de Docker- en Kubernetes-werkplekken.
Werkdirectory	De werkdirectory is de locatie in de container waar de stappen van de taak voor continue integratie (CI) worden uitgevoerd en tevens de locatie waar de code wordt gekloond wanneer een Git-webhook een pijplijnuitvoering activeert.	Beschikbaar met Docker of Kubernetes.
Naamruimte	Als u geen naamruimte invoert, maakt Code Stream een unieke naam in het Kubernetes-cluster dat u heeft opgegeven.	Specifiek voor de Kubernetes-werkplek.

Tabel 4-1. Werkplekgebieden, gegevens en beschikbaarheid (vervolg)

Selectie	Beschrijving	Gegevens en beschikbaarheid
Proxy	<p>Om te communiceren met de werkplekpod in het Kubernetes-cluster implementeert Code Stream één proxyinstantie in de naamruimte <code>codestream-proxy</code> voor elk Kubernetes-cluster. U kunt het type NodePort of LoadBalancer selecteren op basis van de clusterconfiguratie.</p> <p>Welke optie u selecteert, is afhankelijk van de aard van het geïmplementeerde Kubernetes-cluster.</p> <ul style="list-style-type: none"> ■ Selecteer NodePort als de URL van de Kubernetes API-server die wordt opgegeven op het eindpunt, via een van de primaire knooppunten beschikbaar wordt gesteld. ■ Als de URL van de Kubernetes API-server beschikbaar wordt gesteld door een load balancer, zoals met Amazon EKS (Elastic Kubernetes Service), selecteert u LoadBalancer. 	
NodePort	<p>Code Stream gebruikt NodePort om te communiceren met de container die binnen het Kubernetes-cluster wordt uitgevoerd.</p> <p>Als u geen poort selecteert, gebruikt Code Stream een kortstondige poort die Kubernetes toewijst. U moet ervoor zorgen dat de configuratie van de firewallregels inkomend verkeer naar het kortstondige poortbereik (30000-32767) toestaat.</p> <p>Als u een poort invoert, moet u ervoor zorgen dat een andere service in het cluster deze nog niet gebruikt en dat de poort is toegestaan in de firewallregels.</p>	Specifiek voor de Kubernetes-werkplek.
Permanente volumeclaim	<p>Biedt een manier voor de Kubernetes-werkplek om bestanden in pijplijnuitvoeringen te behouden. Wanneer u een naam voor een permanente volumeclaim opgeeft, kan deze de logboeken, artefacten en cache opslaan.</p> <p>Raadpleeg de Kubernetes-documentatie op https://kubernetes.io/docs/concepts/storage/persistent-volumes/ voor meer informatie over het maken van een permanente volumeclaim.</p>	Specifiek voor de Kubernetes-werkplek.

Tabel 4-1. Werkplekgebieden, gegevens en beschikbaarheid (vervolg)

Selectie	Beschrijving	Gegevens en beschikbaarheid
Omgevingsvariabelen	Sleutelwaardeparen die hier worden doorgegeven, zijn beschikbaar voor alle taken voor continue integratie (CI) en aangepaste taken in een pijplijn wanneer deze wordt uitgevoerd.	<p>Beschikbaar met Docker of Kubernetes.</p> <p>Verwijzingen naar variabelen kunnen hier worden doorgegeven.</p> <p>Omgevingsvariabelen die worden opgegeven in de werkplek, worden doorgegeven aan alle taken voor continue integratie (CI) en aangepaste taken in de pijplijn.</p> <p>Als omgevingsvariabelen hier niet worden doorgegeven, moeten deze variabelen expliciet worden doorgegeven aan elke taak voor continue integratie (CI) en elke aangepaste taak in de pijplijn.</p>
CPU-limieten	Limieten voor CPU-resources voor de container voor continue integratie (CI) of container voor een aangepaste taak.	De standaardwaarde is 1 .
Geheugenlimieten	Limieten voor geheugen voor de container voor continue integratie (CI) of container voor een aangepaste taak.	De eenheid is MB.

Tabel 4-1. Werkplekgebieden, gegevens en beschikbaarheid (vervolg)

Selectie	Beschrijving	Gegevens en beschikbaarheid
Git-kloon	Wanneer u Git-kloon selecteert en een Git-webhook de pijplijn aanroept, wordt de code naar de werkplek (container) gekloond.	Als u Git-kloon niet inschakelt, moet u een andere expliciete taak voor continue integratie (CI) in de pijplijn configureren om eerst de code te klonen en vervolgens andere stappen uitvoeren zoals bouwen en testen.
Cache	<p>Met de Code Stream-werkplek kunt u een set directory's of bestanden in de cache opslaan om de volgende pijplijnuitvoeringen te versnellen. Voorbeelden van deze directory's zijn <code>.m2</code> en <code>npm_modules</code>. Als u het in de cache opslaan van gegevens tussen pijplijnuitvoeringen niet vereist, is een persistente volumeclaim niet nodig.</p> <p>Artefacten zoals bestanden of directory's in de container worden in de cache opgeslagen, zodat ze opnieuw kunnen worden gebruikt tijdens pijplijnuitvoeringen. Bijvoorbeeld, <code>node_modules</code>- of <code>.m2</code>-mappen kunnen in de cache worden opgeslagen. Cache accepteert een lijst met paden. Bijvoorbeeld:</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>Specifiek voor het type werkplek.</p> <p>In de Docker-werkplek bereikt u de Cache door een gedeeld pad in de Docker-host te gebruiken om de in de cache opgeslagen gegevens, artefacten en logboeken te behouden.</p> <p>In de Kubernetes-werkplek moet u een permanente volumeclaim opgeven om het gebruik van Cache in te schakelen. Anders is Cache niet beschikbaar.</p>

Wanneer u een Kubernetes API-eindpunt in de pijplijnwerkplek gebruikt, maakt Code Stream de nodige Kubernetes-resources zoals ConfigMap, Geheim en Pod om de taak voor continue integratie (CI) of de aangepaste taak uit te voeren. Code Stream communiceert met de container via de NodePort.

Als u gegevens wilt delen tussen pijplijnuitvoeringen, moet u een permanente volumeclaim opgeven, en Code Stream koppelt de permanente volumeclaim aan de container om de gegevens op te slaan en deze te gebruiken voor volgende pijplijnuitvoeringen.

Een systeemeigen CICD-build plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt

Om een pijplijn met continue integratie en continue levering (CICD) in Code Stream te maken, kunt u de slimme CICD-pijplijnsjabloon gebruiken. Om uw native build voor CICD te plannen, verzamelt u de informatie die u nodig heeft om de slimme pijplijnsjabloon in te vullen voordat u de pijplijn in dit voorbeeldplan maakt.

Als u een CI/CD-pijplijn wilt maken, moet u zowel de fase voor continue integratie (CI) als de fase voor continue levering (CD) van uw pijplijn plannen.

Nadat u de informatie in de slimme pijplijnsjabloon hebt ingevoerd en de sjabloon opslaat, wordt een pijplijn gemaakt die fasen en taken bevat. Deze geeft ook de implementatiebestemming van uw image aan op basis van de typen omgeving die u selecteert, zoals ontwikkeling (Dev) en productie (Prod). De pijplijn publiceert uw containerimage en voltooit de acties die nodig zijn om deze uit te voeren. Nadat uw pijplijn is uitgevoerd, kunt u trends volgen over alle pijplijn-uitvoeringen.

Wanneer een pijplijn een image van Docker Hub bevat, moet u ervoor zorgen dat `cURL` of `wget` in de image is ingesloten voordat u de pijplijn uitvoert. Wanneer de pijplijn wordt uitgevoerd, downloadt Code Stream een binair bestand dat gebruikmaakt van `cURL` of `wget` om commando's uit te voeren.

Zie [De pijplijnwerkplek configureren](#) voor meer informatie over het configureren van de werkplek.

De fase voor continue integratie (CI) plannen

Om de CI-fase van uw pijplijn te plannen, stelt u de externe en interne vereisten in en bepaalt u de informatie die nodig is voor het CI-gedeelte van de slimme pijplijnsjabloon. Hier volgt een overzicht.

In dit voorbeeld wordt een Docker-werkplek gebruikt.

Eindpunten en opslagplaatsen die u nodig heeft:

- Een opslagplaats voor Git-broncode waar ontwikkelaars hun code inchecken. Code Stream haalt de meest recente code op in de pijplijn wanneer ontwikkelaars wijzigingen doorvoeren.
- Een Git-eindpunt voor de opslagplaats waar de broncode van de ontwikkelaars zich bevindt.
- Een Docker-eindpunt voor de Docker-buildhost die de bouwcommando's in een container uitvoert.
- Een Kubernetes-eindpunt zodat Code Stream uw image kan implementeren op een Kubernetes-cluster.
- Een builderimage die de container maakt waarop de continue integratietests worden uitgevoerd.
- Een imageregister-eindpunt zodat de Docker-buildhost daar de builderimage kan ophalen.

U heeft toegang tot een project nodig. Het project groepeerd al uw werk, inclusief uw pijplijn, eindpunten en dashboards. Controleer of u lid bent van een project in Code Stream. Als u geen beheerder bent, vraagt u de Code Stream-beheerder om u als lid toe te voegen aan een project. Zie [Hoe voeg ik een project toe in Code Stream?](#).

U zult een Git-webhook nodig hebben waarmee Code Stream de Git-trigger kan gebruiken om uw pijplijn te activeren wanneer ontwikkelaars codewijzigingen doorvoeren. Zie [Hoe gebruik ik de Git-trigger in Code Stream om een pijplijn uit te voeren?](#).

Uw build-toolsets:

- Uw buildtype, zoals Maven.
- Alle buildtools voor de naverwerking die u gebruikt, inclusief JUnit, JaCoCo, Checkstyle en FindBugs.

Uw publicatietool:

- Een tool, zoals Docker, die uw buildcontainer implementeert.
- Een imagetag, ofwel de commit-ID ofwel het buildnummer.

Uw werkruimte voor het bouwen:

- Een Docker-buildhost, oftewel het Docker-eindpunt.
- Een imageregister. Het CI-gedeelte van de pijplijn haalt de image op van het geselecteerde registreindpunt. De container voert de CI-taken uit en implementeert uw image. Als het register verificatiegegevens vereist, moet u een imageregistereindpunt maken en dit vervolgens hier selecteren, zodat de host de image uit het register kan ophalen.
- URL voor de builderimage die de container maakt waarop de taken voor continue integratie worden uitgevoerd.

De fase voor continue levering (CD) plannen

Om de CD-fase van uw pijplijn te plannen, stelt u de externe en interne vereisten in en bepaalt u de informatie die moet worden ingevoerd in het CD-gedeelte van de slimme pijplijnsjabloon.

Eindpunten die u nodig heeft:

- Een Kubernetes-eindpunt zodat Code Stream uw image kan implementeren op een Kubernetes-cluster.

Omgevingstypen en -bestanden:

- Alle omgevingstypen waar Code Stream uw applicatie implementeert, zoals ontwikkeling en productie. De slimme pijplijnsjabloon maakt de fasen en taken in uw pijplijn op basis van de door u geselecteerde omgevingstypen.

Tabel 4-2. Pijplijnfasen die door de slimme CICD-pijplijnsjabloon worden gemaakt

Pijplijn-inhoud	Wat het doet
Bouw-publicatiefase	Bouwt en test uw code, maakt de builderimage en publiceert de image naar uw Docker-host.
Ontwikkelingsfase	Gebruikt een Amazon Web Services (AWS)-cluster voor ontwikkeling om uw image te maken en te implementeren. In deze fase kunt u een naamruimte op het cluster maken en een geheime sleutel maken.
Productiefase	Gebruikt een productieversie van de VMware Tanzu Kubernetes Grid Integrated Edition (voorheen VMware Enterprise PKS) om uw image naar een Kubernetes-cluster in productie te implementeren.

- Een Kubernetes YAML-bestand dat u selecteert in de Cd-sectie van de slimme CICD-pijplijnsjabloon.

Het Kubernetes YAML-bestand bevat drie vereiste secties voor naamruimte, service en implementatie en een optionele sectie voor Geheim. Als u van plan bent een pijplijn te maken door een image te downloaden uit een privéopslagplaats, moet u een sectie met het Docker-configuratiegeheim opnemen. Als de pijplijn die u maakt, alleen openbaar beschikbare images gebruikt, is geen geheim vereist. Het volgende YAML-voorbeeldbestand bevat vier secties.

```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream
  namespace: codestream
---
apiVersion: v1
data:
  .dockerconfigjson:
eyJhdXRocyI6eyJodHRwciovL2luZ12345678901ci5pby92MS8iOmsidXNlcm5hbWUiOiJhdXRvbWV0aW9uYmV0YSIsInBhc3N3b3JkIjo1Vk13YXJlQDEyMyIsImVtYWlsIjo1YXV0b21hdGlvbmJldGF1c2VyQGdtYWlsLmNvbSIsImFldGgiOiJZWFYwYjIxaGRhbHZibUpsZEdFNlZrMTNZWEpsUURFeU13PT0ifX19
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

```

Opmerking Het Kubernetes YAML-bestand wordt ook gebruikt in de slimme CD-pijplijnsjabloon, zoals in de volgende voorbeelden van gebruiksscenario's:

- [Hoe implementeer ik mijn applicatie in Code Stream naar mijn Blue-Green-implementatie?](#)
 - [Hoe kan ik mijn implementatie terugdraaien in Code Stream?](#)
 - [Hoe kan ik de Docker-trigger gebruiken in Code Stream om een continue leveringspijplijn uit te voeren?](#)
-

Als u het bestand in de slimme sjabloon wilt toepassen, klikt u op **Selecteren** en selecteert u het Kubernetes YAML-bestand. Klik vervolgens op **Proces**. In de slimme pijplijnsjabloon worden de beschikbare services en implementatieomgevingen weergegeven. U selecteert een service, het clustereindpunt en de implementatiestrategie. Als u bijvoorbeeld het Canary-implementatiemodel wilt gebruiken, selecteert u **Canary** en voert u een percentage in voor de implementatiefase.

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 ☐

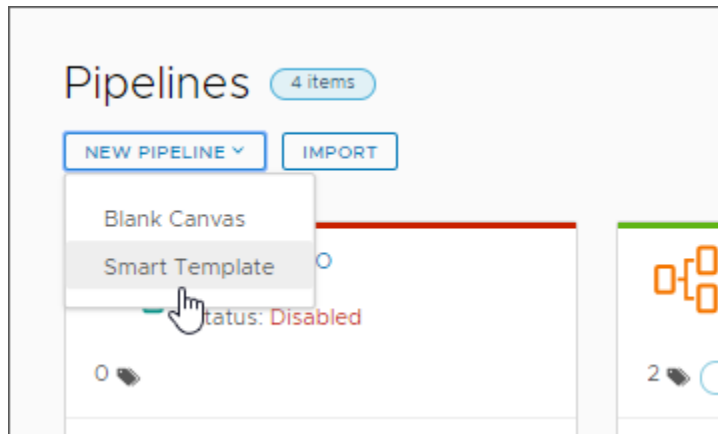
Health check URL ⓘ

Zie [Hoe implementeer ik mijn applicatie in Code Stream naar mijn Blue-Green-implementatie?](#) als u een voorbeeld wilt zien van het gebruik van de slimme pijplijnsjabloon om een pijplijn te maken voor een Blue-Green-implementatie.

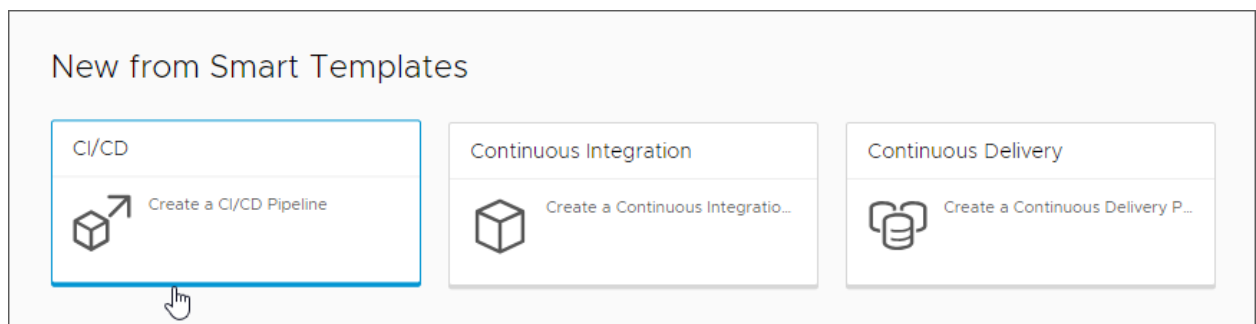
Hoe u de CICD-pijplijn maakt met behulp van de slimme pijplijnsjabloon

Nadat u alle informatie hebt verzameld en het benodigde hebt ingesteld, kunt u op de volgende manier een pijplijn maken op basis van de slimme CICD-pijplijnsjabloon.

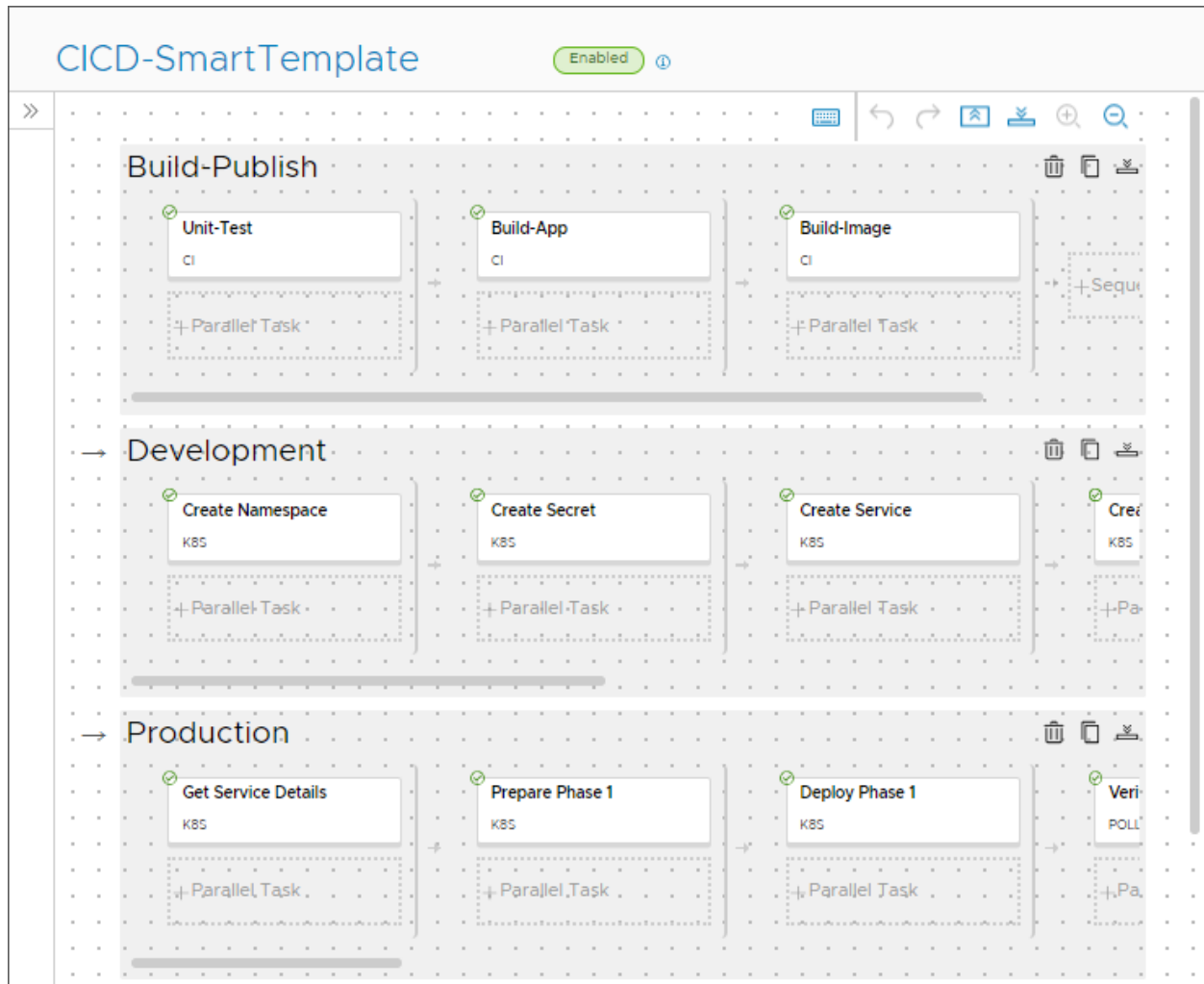
In pijplijnen selecteert u **Nieuwe pijplijn > Slimme sjablonen**.



U selecteert de slimme CICD-pijpijnsjabloon.



U vult de sjabloon in en slaat de pijplijn op met de fasen die deze maakt. Als u nog laatste wijzigingen moet aanbrengen, kunt u de pijplijn bewerken en opslaan.



Vervolgens schakelt u de pijplijn in en voert u deze uit. Nadat deze is uitgevoerd, kunt u de volgende zaken nog bekijken:

- Controleer of uw pijplijn is geslaagd. Klik op **Uitvoeringen** en zoek naar uw pijplijn. Als de pijplijn is mislukt, corrigeert u eventuele fouten en voert u deze opnieuw uit.
- Controleer of de Git-webhook goed werkt. Op het tabblad **Activiteit** in Git worden de gebeurtenissen weergegeven. Klik op **Triggers > Git > Activiteit**.
- Bekijk het pijplijndashboard en onderzoek de trends. Klik op **Dashboards** en zoek naar uw pijplijndashboard. U kunt ook een custom dashboard maken om te rapporteren over aanvullende KPI's.

Zie [Hoe kan ik code vanuit mijn GitHub- of GitLab-opslagplaats continu integreren in mijn pijplijn in Code Stream?](#) voor een gedetailleerd voorbeeld.

Een native build voor continue integratie plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt

Als u een pijplijn voor continue integratie (CI) in VMware Code Stream wilt maken, kunt u de slimme pijplijnsjabloon voor continue integratie gebruiken. Om uw native build voor continue integratie te plannen, verzamelt u de informatie die u nodig heeft om de slimme pijplijnsjabloon in te vullen voordat u de pijplijn in dit voorbeeldplan maakt.

Wanneer u de slimme pijplijnsjabloon invult, maakt deze een pijplijn voor continue integratie in uw opslagplaats en voert deze de acties uit zodat de pijplijn kan worden uitgevoerd. Nadat uw pijplijn is uitgevoerd, kunt u trends volgen over alle pijplijn-uitvoeringen.

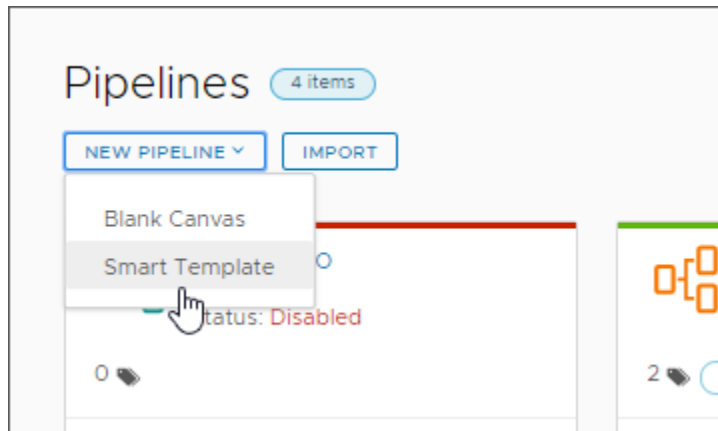
Uw build plannen voordat u de slimme pijplijnsjabloon voor continue integratie gebruikt:

- Identificeer een project dat al uw werk groepeerd, inclusief uw pijplijn, eindpunten en dashboards.
- Verzamel de informatie voor uw build zoals is beschreven in het gedeelte over continue levering van [Een systeemeigen CI/CD-build plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt](#).

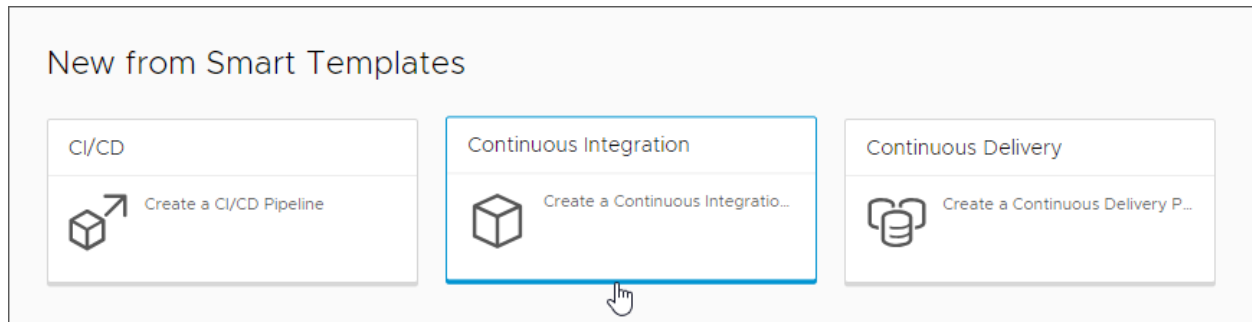
Voeg bijvoorbeeld een Kubernetes-eindpunt toe waar Code Stream de container zal implementeren.

Vervolgens maakt u een pijplijn met behulp van de slimme pijplijnsjabloon voor continue integratie.

In Pijplijnen selecteert u **Slimme sjablonen**.



U selecteert de slimme pijplijnsjabloon voor continue integratie.



Als u de pijplijn wilt opslaan met de fasen die deze maakt, vult u de sjabloon in en voert u een naam in voor de pijplijn. Als u de pijplijn wilt opslaan met de fasen die deze maakt, klikt u op **Maken**.

De Code Stream-pijplijnwerkplek ondersteunt Docker en Kubernetes voor taken voor continue integratie en aangepaste taken.

Zie [De pijplijnwerkplek configureren](#) voor meer informatie over het configureren van de werkplek.

Als u nog laatste wijzigingen wilt aanbrengen, kunt u de pijplijn bewerken. Vervolgens kunt u de pijplijn inschakelen en deze uitvoeren. Nadat de pijplijn is uitgevoerd:

- Controleer of uw pijplijn is geslaagd. Klik op **Uitvoeringen** en zoek naar uw pijplijn. Als de pijplijn is mislukt, corrigeert u eventuele fouten en voert u deze opnieuw uit.
- Controleer of de Git-webhook goed werkt. Op het tabblad **Activiteit** in Git worden de gebeurtenissen weergegeven. Klik op **Triggers > Git > Activiteit**.
- Bekijk het pijplijndashboard en onderzoek de trends. Klik op **Dashboards** en zoek naar uw pijplijndashboard. Als u over meer KPI's wilt rapporteren, kunt u een aangepast dashboard maken.

Zie [Hoe kan ik code vanuit mijn GitHub- of GitLab-opslagplaats continu integreren in mijn pijplijn in Code Stream?](#) voor een gedetailleerd voorbeeld.

Een native build voor continue levering plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt

Als u een pijplijn voor continue levering (CD) in Code Stream wilt maken, kunt u de slimme pijplijnsjabloon voor continue levering gebruiken. Om uw native build voor continue levering te plannen, verzamelt u de informatie die u nodig heeft om de slimme pijplijnsjabloon in te vullen voordat u de pijplijn in dit voorbeeldplan maakt.

Wanneer u de slimme pijplijnsjabloon invult, maakt deze een pijplijn voor continue levering in uw opslagplaats en voert deze de acties uit zodat de pijplijn kan worden uitgevoerd. Nadat uw pijplijn is uitgevoerd, kunt u trends volgen over alle pijplijn-uitvoeringen.

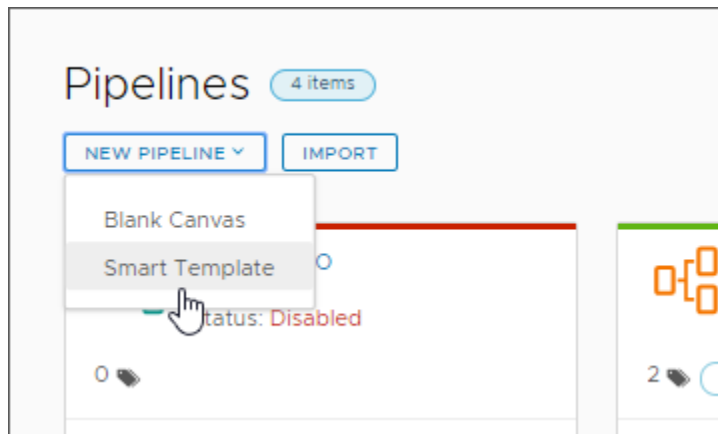
Uw build plannen voordat u de slimme pijplijnsjabloon voor continue levering gebruikt:

- Identificeer een project dat al uw werk groepeer, inclusief uw pijplijn, eindpunten en dashboards.

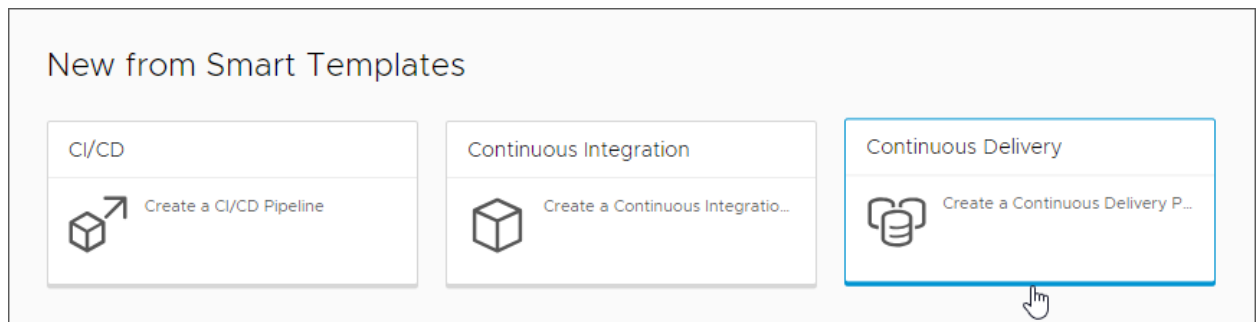
- Verzamel de informatie voor uw build zoals is beschreven in het gedeelte over continue levering van [Een systeemeigen CI/CD-build plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt](#). Bijvoorbeeld:
 - Voeg een Kubernetes-eindpunt toe waar Code Stream de container zal implementeren.
 - Bereid het Kubernetes YAML-bestand voor dat de naamruimte, service en implementatie maakt. Als u een image wilt downloaden uit een privéopslagplaats, moet het YAML-bestand een sectie bevatten met het Docker-configuratiegeheim.

Vervolgens maakt u een pijplijn met behulp van de slimme pijplijnsjabloon voor continue levering.

In Pijplijnen selecteert u **Slimme sjablonen**.



U selecteert de slimme pijplijnsjabloon voor continue levering.



U vult de sjabloon in en voert een naam in voor de pijplijn. Als u de pijplijn wilt opslaan met de fasen die deze maakt, klikt u op **Maken**.

De Code Stream-pijplijnwerkplek ondersteunt Docker en Kubernetes voor taken voor continue integratie en aangepaste taken.

Zie [De pijplijnwerkplek configureren](#) voor meer informatie over het configureren van de werkplek.

Als u nog laatste wijzigingen wilt aanbrengen, kunt u de pijplijn bewerken. Vervolgens kunt u de pijplijn inschakelen en deze uitvoeren. Nadat de pijplijn is uitgevoerd:

- Controleer of uw pijplijn is geslaagd. Klik op **Uitvoeringen** en zoek naar uw pijplijn. Als de pijplijn is mislukt, corrigeert u eventuele fouten en voert u deze opnieuw uit.

- Controleer of de Git-webhook goed werkt. Op het tabblad **Activiteit** in Git worden de gebeurtenissen weergegeven. Klik op **Triggers > Git > Activiteit**.
- Bekijk het pijplijndashboard en onderzoek de trends. Klik op **Dashboards** en zoek naar uw pijplijndashboard. Als u over meer KPI's wilt rapporteren, kunt u een aangepast dashboard maken.

Zie [Hoe kan ik code vanuit mijn GitHub- of GitLab-opslagplaats continu integreren in mijn pijplijn in Code Stream?](#) voor een gedetailleerd voorbeeld.

Een CI/CD-systeemeigen build plannen in Code Stream voordat u handmatig taken toevoegt

Om een pijplijn met continue integratie en continue levering (CI/CD) in Code Stream te maken, kunt u handmatig fasen en taken toevoegen. Om uw CI/CD-systeemeigen build te plannen, verzamelt u de informatie die u nodig heeft en maakt u vervolgens een pijplijn waaraan u handmatig fasen en taken toevoegt.

U moet zowel de fase voor continue integratie (CI) als de fase voor continue levering (CD) van uw pijplijn plannen. Nadat u de pijplijn heeft gemaakt en uitgevoerd, kunt u trends in de pijplijn-uitvoeringen volgen.

Wanneer een pijplijn een image van Docker Hub bevat, moet u ervoor zorgen dat `cURL` of `wget` in de image is ingesloten voordat u de pijplijn uitvoert. Wanneer de pijplijn wordt uitgevoerd, downloadt Code Stream een binair bestand dat gebruikmaakt van `cURL` of `wget` om commando's uit te voeren.

De Code Stream-pijplijnwerkplek ondersteunt Docker en Kubernetes voor taken voor continue integratie en aangepaste taken.

Zie [De pijplijnwerkplek configureren](#) voor meer informatie over het configureren van de werkplek.

De externe en interne vereisten plannen

Als u de CI- en CD-fasen van uw pijplijn wilt plannen, geven de volgende vereisten aan wat u moet doen voordat u uw pijplijn maakt.

In dit voorbeeld wordt een Docker-werkplek gebruikt.

Als u een pijplijn wilt maken op basis van dit voorbeeldplan, gebruikt u een Docker-host, een Git-opslagplaats, Maven en verschillende buildtools voor naverwerking.

Eindpunten en opslagplaatsen die u nodig heeft:

- Een opslagplaats voor Git-broncode waar ontwikkelaars hun code inchecken. Code Stream haalt de meest recente code op in de pijplijn wanneer ontwikkelaars wijzigingen doorvoeren.
- Een Docker-eindpunt voor de Docker-buildhost die de bouwcommando's in een container uitvoert.

- Een builderimage die de container maakt waarop de continue integratietests worden uitgevoerd.
- Een imageregister-eindpunt zodat de Docker-buildhost daar de builderimage kan ophalen.

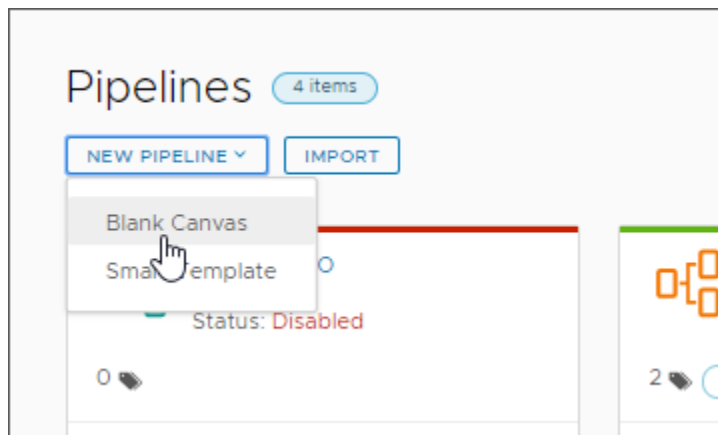
U heeft toegang tot een project nodig. Het project groepeerd al uw werk, inclusief uw pijplijn, eindpunten en dashboards. Controleer of u lid bent van een project in Code Stream. Als u geen beheerder bent, vraagt u de Code Stream-beheerder om u als lid toe te voegen aan een project. Zie [Hoe voeg ik een project toe in Code Stream?](#).

U zult een Git-webhook nodig hebben waarmee Code Stream de Git-trigger kan gebruiken om uw pijplijn te activeren wanneer ontwikkelaars codewijzigingen doorvoeren. Zie [Hoe gebruik ik de Git-trigger in Code Stream om een pijplijn uit te voeren?](#).

Hoe u de CICD-pijplijn maakt en de werkruimte configureert

U moet de pijplijn maken en vervolgens de werkruimte, de invoerparameters voor de pijplijn en taken configureren.

Om de pijplijn te maken, klikt u op **Pijplijnen > Nieuwe pijplijn > Blanco canvas**.



Voer de gegevens over continue integratie in op het tabblad Werkruimte:

- Neem uw Docker-buildhost op.
- Voer de URL voor uw builderimage.
- Selecteer het imageregister-eindpunt zodat de pijplijn de image daarvan kan ophalen. De container voert de CI-taken uit en implementeert uw image. Als het register verificatiegegevens vereist, moet u eerst het imageregister-eindpunt maken en dit vervolgens hier selecteren, zodat de host de image uit het register kan ophalen.
- Voeg de artefacten toe die in de cache moeten worden opgeslagen. Voor een geslaagde build worden artefacten, zoals directory's, gedownload als afhankelijkheden. De cache is de locatie waar deze artefacten zich bevinden. Afhankelijke artefacten kunnen bijvoorbeeld de `.m2`-map voor Maven en de `node_modules`-map voor Node.js bevatten. Deze directory's worden in de cache opgeslagen bij de uitvoering van pijplijnen om tijd te besparen tijdens de builds.

Configureer de inputparameters voor de pijplijn op het tabblad Input.

- Als uw pijplijn invoerparameters van een Git-, Gerrit- of Docker-triggergebeurtenis gebruikt, selecteert u het triggertype voor het Automatisch injecteren van parameters. Gebeurtenissen kunnen Onderwerp wijzigen voor Gerrit of Git of Naam van de eigenaar van de gebeurtenis voor Docker omvatten. Als uw pijplijn geen invoerparameters gebruikt die worden doorgegeven vanaf de gebeurtenis, laat u de selectie van Automatisch injecteren van parameters op **Geen** ingesteld staan.
- Als u een waarde en een beschrijving wilt toepassen op een inputparameter voor de pijplijn, klikt u op de drie verticale puntjes en klikt u op **Bewerken**. De waarde die u invoert, wordt gebruikt als input voor taken, fasen of meldingen.
- Om een inputparameter voor een pijplijn toe te voegen, klikt u op **Toevoegen**. U kunt bijvoorbeeld `approvers` toevoegen om een standaardwaarde voor elke uitvoering weer te geven, maar die kan worden omzeild om een andere goedkeurder bij runtime weer te geven.
- Als u een ingevoerde parameter wilt toevoegen of verwijderen, klikt u op **Geïnjecteerde parameter toevoegen/verwijderen**. U kunt bijvoorbeeld een ongebruikte parameter verwijderen om de pagina met zoekresultaten overzichtelijker te maken en alleen de invoerparameters weer te geven die worden gebruikt.

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

Configureer de pijplijn om uw code te testen:

- Voeg een CI-taak toe en configureer deze.
- Neem stappen op om `mvn test` uit te voeren op uw code.
- Om eventuele problemen na de taakuitvoeringen te identificeren, voert u buildtools voor naverwerking uit, zoals JUnit en JaCoCo, FindBugs en 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 §

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

Configureer de pijplijn om uw code te bouwen:

- Voeg een CI-taak toe en configureer deze.

- Neem stappen op om `mvn clean install` uit te voeren in uw code.
- Neem de locatie en de JAR-bestandsnaam op zodat uw artefact behouden blijft.

Task Build-App Notifications Rollback VALIDATE TASK

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

Type * CI

Precondition [SYNTAX GUIDE](#)

Continue on failure ☐

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

Steps *

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

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

Export
Enter comma separated values

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

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

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

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

Configureer de pijplijn om uw image op uw Docker-host te publiceren:

- Voeg een CI-taak toe en configureer deze.
- Voeg stappen toe die uw image doorvoeren, exporteren, bouwen en pushen.
- Voeg de exportsleutel van `IMAGE` toe voor de volgende taak die moet worden geconsumeerd.

The screenshot shows the 'Build-Image' task configuration window. At the top, there are tabs for 'Task', 'Build-Image', 'Notifications', and 'Rollback', along with a 'VALIDATE TASK' button. The 'Task name' is 'Build-Image'. The 'Type' is 'CI'. There is a 'Precondition' field with a 'SYNTAX GUIDE' button. The 'Continue on failure' checkbox is unchecked. The 'Continuous Integration' section has a note: 'A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.' 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), 'JUnit', 'JaCoCo', 'FindBugs', and 'Checkstyle', each with 'Label' and 'Path' input fields and a '+' button to add more.

Nadat u de werkruimte, de invoerparameters, de testtaken en de bouwtaken heeft geconfigureerd, slaat u uw pijplijn op.

Hoe u uw pijplijn kunt inschakelen en uitvoeren

Nadat u uw pijplijn met fasen en taken heeft geconfigureerd, kunt u de pijplijn opslaan en inschakelen.

Wacht vervolgens totdat de pijplijn is uitgevoerd en voltooid en controleer of deze is geslaagd. Als de pijplijn is mislukt, corrigeert u eventuele fouten en voert u deze opnieuw uit.

Nadat de pijplijn is geslaagd, zijn er een aantal dingen die u mogelijk wilt bevestigen:

- Controleer de uitvoering van de pijplijn en bekijk de resultaten van de taakstappen.
- Zoek in de werkruimte van de pijplijnuitvoering naar de details over uw container en de gekloonde Git-opslagplaats.
- Bekijk in de werkruimte de resultaten van uw naverwerkingstools en controleer op fouten, codedekking, bugs en stijlproblemen.
- Controleer of uw artefact behouden blijft. Controleer ook of de afbeelding is geëxporteerd met de naam en de waarde van de IMAGE.
- Ga naar uw Docker-opslagplaats en controleer of de pijplijn uw container heeft gepubliceerd.

Voor een gedetailleerd voorbeeld waarin wordt weergegeven hoe de code door Code Stream continu wordt geïntegreerd, raadpleegt u [Hoe kan ik code vanuit mijn GitHub- of GitLab-opslagplaats continu integreren in mijn pijplijn in Code Stream?](#).

Plannen voor terugdraaien in Code Stream

Als een pijplijnuitvoering mislukt, kunt u terugdraaien gebruiken om uw omgeving terug te brengen naar een vorige stabiele status. Als u terugdraaien wilt gebruiken, plant u een terugdraaistroom en leert u hoe deze moet worden geïmplementeerd.

Een terugdraaistroom schrijft de vereiste stappen voor om een fout in de implementatie terug te draaien. De stroom neemt de vorm aan van een terugdraaipijplijn die een of meer opeenvolgende taken bevat die afhankelijk zijn van het implementatietype dat is uitgevoerd en mislukt. De implementatie en het terugdraaien van een traditionele applicatie is bijvoorbeeld anders dan de implementatie en het terugdraaien van een container-applicatie.

Om terug te keren naar een goede implementatiestaat, bevat een terugdraaipijplijn doorgaans taken voor het volgende:

- Statussen of omgevingen opschonen.
- Een door de gebruiker gespecificeerd script uitvoeren om de wijzigingen terug te draaien.
- Een vorige revisie van een implementatie implementeren.

Om terugdraaifuncties toe te voegen aan een bestaande implementatiepijplijn, koppelt u de terugdraaipijplijn aan de taken of fasen in de implementatiepijplijn die u wilt terugdraaien voordat u uw implementatiepijplijn uitvoert.

Hoe configureer ik terugdraaien?

Als u terugdraaien in uw implementatie wilt configureren, moet u:

- Een implementatiepijplijn maken.
- Mogelijke foutpunten identificeren in de implementatiepijplijn die het terugdraaien activeren zodat u de terugdraaipijplijn kunt koppelen. U kunt bijvoorbeeld uw terugdraaipijplijn koppelen aan een voorwaarde of polling-taaktype in de implementatiepijplijn waarmee wordt gecontroleerd of een vorige taak met succes is voltooid. Zie [Hoe gebruik ik variabelebindingen in een voorwaardetaak om een pijplijn uit te voeren of te stoppen in Code Stream?](#) voor informatie over voorwaardetaken.
- Bepaal het bereik van de fout die de terugdraaipijplijn activeert, zoals een taak- of fasefout. U kunt terugdraaien ook koppelen aan een fase.
- Bepaal welke terugdraaitaak of -taken moeten worden uitgevoerd in geval van een fout. U maakt een terugdraaipijplijn met die taken.

U kunt handmatig een terugdraaipijplijn maken, of Code Stream kan er een voor u maken.

- Als u een leeg canvas gebruikt, kunt u handmatig een terugdraaipijplijn maken die parallel loopt met een bestaande implementatiepijplijn. Vervolgens koppelt u de terugdraaipijplijn aan een of meer taken in de implementatiepijplijn die terugdraaiacties activeren bij een fout.

- Met behulp van een slimme pijplijnsjabloon kunt u een implementatiepijplijn configureren met de terugdraaiactie. Vervolgens maakt Code Stream automatisch een of meer standaard terugdraai pijplijnen met vooraf gedefinieerde taken die de implementatie terugdraaien bij een fout.

Zie [Hoe kan ik mijn implementatie terugdraaien in Code Stream?](#) voor een gedetailleerd voorbeeld van hoe u een CD-pijplijn kunt configureren met terugdraaien met behulp van een slimme pijplijnsjabloon.

Wat gebeurt er als mijn implementatiepijplijn meerdere taken of fasen met terugdraaifunctie heeft

Als u meerdere taken of taken en fasen met terugdraaifunctie heeft toegevoegd, dan moet u er rekening mee houden dat de terugdraaisequentie varieert.

Tabel 4-3. Terugdraaivolgorde bepalen

Als u terugdraaien toevoegt aan...	Wanneer wordt er teruggedraaid...
Parallele taken	Als een van de parallelle taken mislukt, wordt deze taak teruggedraaid nadat alle parallelle taken zijn voltooid of zijn mislukt. Er wordt niet onmiddellijk teruggedraaid nadat de taak is mislukt.
Zowel de taak in een fase als de fase	Als een taak mislukt, wordt het terugdraaien van de taak uitgevoerd. Als de taak deel uitmaakt van een groep parallelle taken, wordt de taak teruggedraaid nadat alle parallelle taken zijn voltooid of zijn mislukt. Nadat het terugdraaien van de taak is voltooid of niet is voltooid, wordt het terugdraaien van de fase uitgevoerd.

Overweeg een pijpleiding met de volgende onderdelen:

- Een productiefase met terugdraaifunctie.
- Een groep parallelle taken, elke taak met een eigen terugdraaifunctie.

De taak met de naam **UPD Deploy US** heeft de terugdraai pijplijn **RB_Deploy_US**. Als **UPD Deploy US** mislukt, volgt het terugdraaien de stroom die is gedefinieerd in de **RB_Deploy_US**-pijplijn.

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

Als **UPD Deploy US** mislukt, wordt de **RB_Deploy_US**-pijplijn uitgevoerd nadat **UPD Deploy UK** en **UPD Deploy AU** ook zijn voltooid of mislukt. Er wordt niet onmiddellijk teruggedraaid nadat **UPD Deploy US** is mislukt. En omdat de productiefase ook een teruggedraaifunctie heeft, wordt de teruggedraai-pijplijn voor de fase uitgevoerd nadat de **RB_Deploy_US**-pijplijn is uitgevoerd.

Tutorials voor het gebruik van Code Stream

5

Code Stream modelleert en ondersteunt uw DevOps-releaselevenscyclus en test uw applicaties continu en geeft deze vrij aan ontwikkelingsomgevingen en productieomgevingen.

U heeft al alles wat u nodig heeft ingesteld zodat u Code Stream kunt gebruiken. Zie [Hoofdstuk 2 Code Stream instellen om mijn releaseproces te modelleren](#).

Nu kunt u pijplijnen maken die het bouwen en testen van de ontwikkelaarscode automatiseren voordat u deze vrijgeeft naar productie. U kunt Code Stream container-gebaseerde of traditionele applicaties laten implementeren.

Tabel 5-1. Code Stream gebruiken in uw DevOps-levenscyclus

Funcities	Voorbeelden van wat u kunt doen
Gebruik de systeemeigen bouwbaarheid in Code Stream.	<p>Maak pijplijnen voor continue integratie en levering (CICD), continue integratie (CI) en continue levering (CL) die uw code continu integreren, in containers plaatsen en leveren.</p> <ul style="list-style-type: none">■ Gebruik een slimme pijplijnsjabloon die een pijplijn voor u maakt.■ Voeg handmatig fasen en taken toe aan een pijplijn.
Breng uw applicaties uit en automatiseer releases.	<p>Integreer uw applicaties en geef ze vrij op verschillende manieren.</p> <ul style="list-style-type: none">■ Integreer continu uw code van een GitHub- of GitLab-opslagplaats in uw pijplijn.■ Integreer een Docker-host om taken voor continue integratie uit te voeren zoals is beschreven in dit blogartikel over het maken van een Docker-host voor vRealize Automation Code Stream.■ Automatiseer de implementatie van uw applicatie met behulp van een YAML-cloudsjabloon.■ Automatiseer de implementatie van uw applicatie naar een Kubernetes-cluster.■ Geef uw applicatie vrij naar een Blue-Green-implementatie.■ Integreer Code Stream met uw eigen tools voor het bouwen, testen en implementeren.■ Gebruik een REST API die Code Stream integreert met andere applicaties.
Volg trends, statistieken en KPI's.	<p>Maak custom dashboards en krijg inzicht in de prestaties van uw pijplijnen.</p>
Los problemen op.	<p>Als de uitvoering van een pijplijn mislukt, laat u Code Stream een Jira-ticket maken.</p>

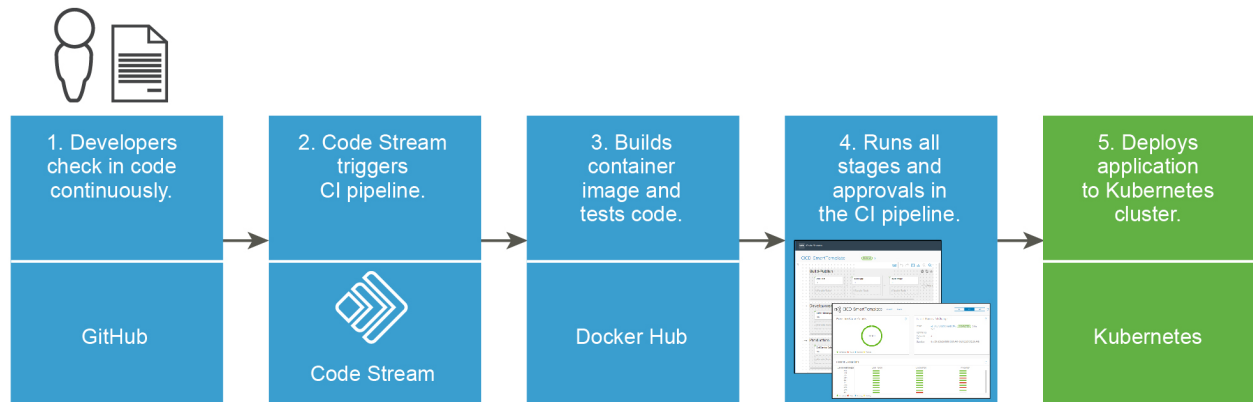
Dit hoofdstuk omvat de volgende onderwerpen:

- [Hoe kan ik code vanuit mijn GitHub- of GitLab-opslagplaats continu integreren in mijn pijplijn in Code Stream?](#)
- [Hoe kan ik de release van een applicatie die ik implementeer vanuit een YAML-cloudsjabloon in Code Stream automatiseren?](#)

- Hoe kan ik de release van een applicatie in Code Stream op een Kubernetes-cluster automatiseren?
- Hoe implementeer ik mijn applicatie in Code Stream naar mijn Blue-Green-implementatie?
- Hoe kan ik mijn eigen tools voor bouwen, testen en implementeren integreren met Code Stream?
- Hoe gebruik ik de resource-eigenschappen van een cloudsjabloontaak in mijn volgende taak
- Hoe gebruik ik een REST API om Code Stream te integreren met andere applicaties?
- Hoe kan ik een pijplijn gebruiken als code in Code Stream?

Hoe kan ik code vanuit mijn GitHub- of GitLab-opslagplaats continu integreren in mijn pijplijn in Code Stream?

Als ontwikkelaar wilt u uw code voortdurend integreren vanuit een GitHub- of GitLab Enterprise-opslagplaats. Wanneer uw ontwikkelaars hun code bijwerken en wijzigingen in de opslagplaats doorvoeren, kan Code Stream luisteren naar die wijzigingen en de pijplijn activeren.



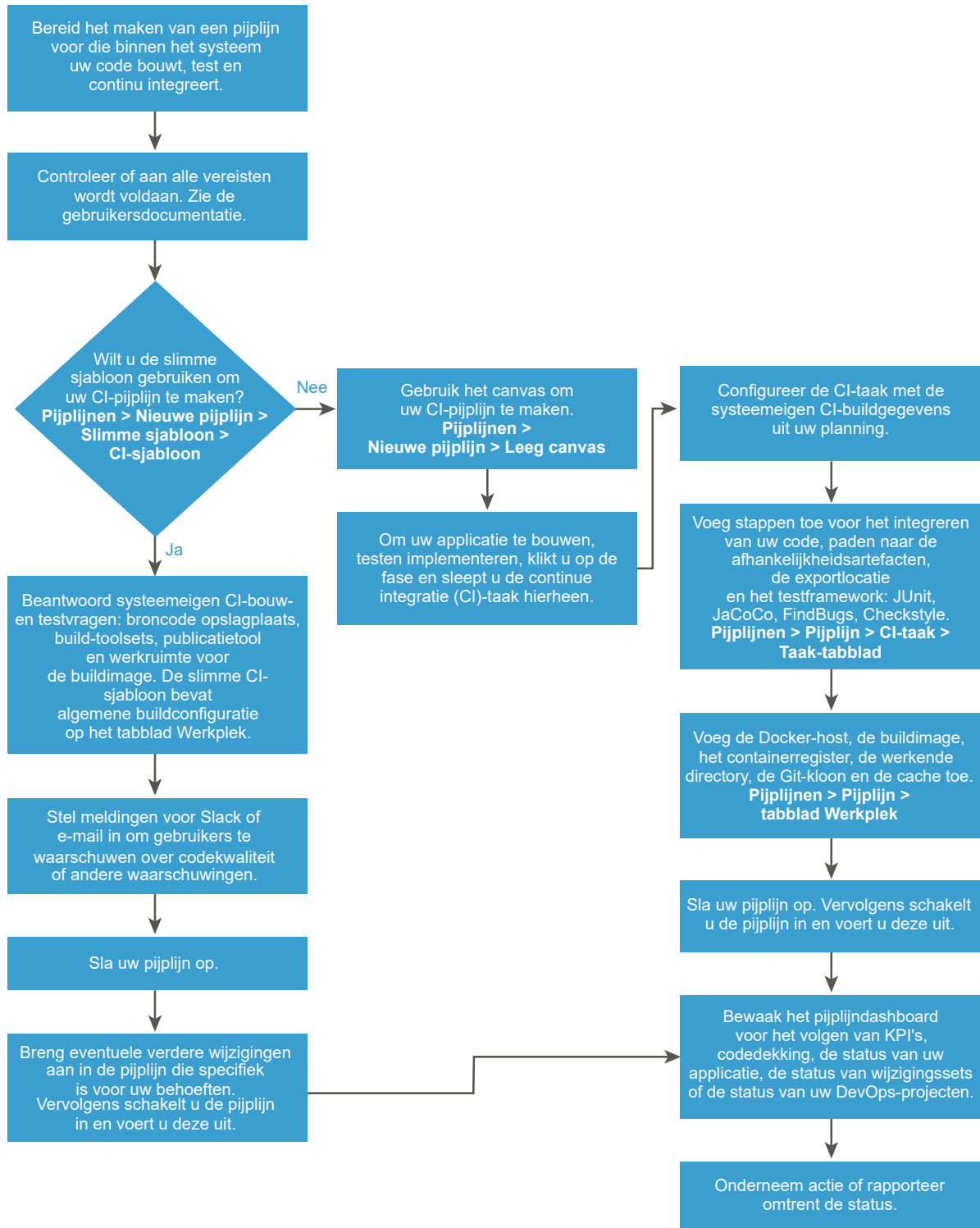
Als u wilt dat Code Stream uw pijplijn bij codewijzigingen activeert, gebruikt u de Git-trigger. Code Stream activeert vervolgens uw pijplijn elke keer dat u wijzigingen in uw code doorvoert.

De Code Stream-pijplijnwerkplek ondersteunt Docker en Kubernetes voor taken voor continue integratie en aangepaste taken.

Zie [De pijplijnwerkplek configureren](#) voor meer informatie over het configureren van de werkplek.

In het volgende stroomdiagram ziet u de werkstroom die u kunt kiezen als u een slimme pijplijnsjabloon gebruikt om uw pijplijn te maken of de pijplijn handmatig bouwt.

Figuur 5-1. Werkstroom die gebruikmaakt van een slimme pijplijnsjabloon of handmatig een pijplijn maakt



In het volgende voorbeeld wordt een Docker-werkplek gebruikt.

Om uw code te bouwen, gebruikt u een Docker-host. U gebruikt JUnit en JaCoCo als tools voor het testframework, die eenheidstests en codedekking uitvoeren, en u neemt deze in uw pijplijn op.

Vervolgens kunt u de slimme pijplijnsjabloon voor continue integratie gebruiken om een pijplijn voor continue integratie te maken die uw code bouwt, test en implementeert in het Kubernetes-cluster van uw projectteam op AWS. Als u de code-afhankelijkheidsartefacten wilt opslaan voor uw taak voor continue integratie, die tijd kan besparen bij het bouwen van code, kunt u een cache gebruiken.

In de pijplijntaak die uw code bouwt en test, kunt u verschillende stappen voor continue integratie opnemen. Deze stappen kunnen worden opgeslagen in dezelfde werkdirectory waar Code Stream de broncode kloont wanneer de pijplijn wordt geactiveerd.

Om uw code op het Kubernetes-cluster te implementeren, kunt u een Kubernetes-taak in uw pijplijn gebruiken. U moet vervolgens uw pijplijn inschakelen en uitvoeren. Vervolgens wijzigt u de code in de opslagplaats en bekijkt u de activering van de pijplijn. Om de trends in uw pijplijn te bewaken en te rapporteren nadat deze is uitgevoerd, gebruikt u de dashboards.

In het volgende voorbeeld gebruikt u de slimme pijplijnsjabloon voor continue integratie om een pijplijn voor continue integratie te maken die uw code continu integreert in uw pijplijn. In dit voorbeeld wordt een Docker-werkplek gebruikt.

U kunt ook handmatig de pijplijn maken en er fasen en taken aan toevoegen. Zie [Een CICD-systeemeigen build plannen in Code Stream voordat u handmatig taken toevoegt](#) voor meer informatie over het plannen van een continue integratie-build en het handmatig maken van de pijplijn.

Voorwaarden

- Plan voor uw build met continue integratie. Zie [Een native build voor continue integratie plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt](#).
- Verifieer dat er een opslagplaats voor GitLab-broncode bestaat. Neem contact op met uw Code Stream-beheerder voor hulp.
- Voeg een Git-eindpunt toe. Zie [Hoe gebruik ik de Git-trigger in Code Stream om een pijplijn uit te voeren?](#) voor een voorbeeld.
- Als u wilt dat Code Stream naar wijzigingen in uw GitHub-opslagplaats of GitLab-opslagplaats luistert en een pijplijn activeert wanneer er wijzigingen optreden, voegt u een webhook toe. Zie [Hoe gebruik ik de Git-trigger in Code Stream om een pijplijn uit te voeren?](#) voor een voorbeeld.
- Voeg een Docker-hosteindpunt toe, waardoor een container voor de taak voor continue integratie wordt gemaakt, die door meerdere taken voor continue integratie kan worden gebruikt. Zie [Wat zijn eindpunten in Code Stream?](#) voor meer informatie over eindpunten.
- Verkrijg de image-URL, de buildhost en de URL voor de buildimage. Neem contact op met uw Code Stream-beheerder voor hulp.

- Verifieer dat u JUnit en JaCoCo gebruikt voor de tools van uw testframework.
- Stel een externe instantie in voor uw CI-build: Jenkins, TFS of Bamboo. De Kubernetes-invoegtoepassing implementeert uw code. Neem contact op met uw Code Stream-beheerder voor hulp.

Procedure

- 1 Volg de vereisten.
- 2 Als u de pijplijn wilt maken met behulp van de slimme pijplijnsjabloon, opent u de slimme pijplijnsjabloon voor continue integratie en vult u het formulier in.
 - a Klik op **Pijplijnen > Nieuwe pijplijn > Slimme sjabloon > Continue integratie**.
 - b Beantwoord de vragen in de sjabloon over uw opslagplaats voor de broncode, build-toolsets, publicatietool en de werkruimte voor de buildimage.
 - c Voeg Slack- of e-mailmeldingen voor uw team toe.
 - d Als u wilt dat de slimme pijplijnsjabloon de pijplijn maakt, klikt u op **Maken**.
 - e Als u verdere wijzigingen wilt aanbrengen in de pijplijn, klikt u op **Bewerken**, brengt u de wijzigingen aan en klikt u op **Opslaan**.
 - f Schakel de pijplijn in en voer deze uit.
- 3 Als u de pijplijn handmatig wilt maken, voegt u fasen en taken toe aan het canvas en neemt u uw systeemeigen CI-buildinformatie op in de CI-taak.
 - a Klik op **Pijplijnen > Nieuwe pijplijn > Leeg canvas**.
 - b Klik op de fase en sleep vervolgens de verschillende CI-taken van het navigatiedeevenster naar de fase.
 - c Als u de CI-taak wilt configureren, klikt u erop en klikt u op het tabblad **Taak**.
 - d Voeg de stappen toe die uw code continu integreren.
 - e Voeg de paden toe aan de afhankelijkheidsartefacten.
 - f Voeg de exportlocatie toe.
 - g Voeg de tools voor het testframework toe die u gaat gebruiken.
 - h Voeg de Docker-host en buildimage toe.
 - i Voeg het containerregister, de werkdirectory en de cache toe.
 - j Sla de pijplijn op en schakel deze vervolgens in.
- 4 Wijzig de code in uw GitHub- of GitLab-opslagplaats.

De Git-trigger activeert uw pijplijn die met de uitvoering begint.
- 5 Als u wilt controleren of de codewijziging de pijplijn heeft geactiveerd, klikt u op **Triggers > Git > Activiteit**.

- 6 Als u de uitvoering van uw pijplijn wilt bekijken, klikt u op **Uitvoeringen** en controleert u of de stappen uw buildimage hebben gemaakt en geëxporteerd.

The screenshot displays the vRealize Automation Code Stream interface. On the left is a sidebar with navigation links: Dashboards, Executions, User Operations, Pipelines, Manage (with sub-links: Endpoints, Variables, Triggers, Gerrit, Git), and a search bar. The main area shows the execution details for 'CICD-SmartTemplate #51', which is in a 'COMPLETED' state. The 'Build-Publish' stage is active, showing three steps: 'Unit-Test', 'Build-App', and 'Build-Image' (the current step). The 'Build-Image' step is completed, showing a terminal log with Docker commands and warnings. The 'Exported' section shows the image name 'automation/cicd-smart-template:51'.

- 7 Als u het pijplijndashboard wilt bewaken zodat u KPI's en trends kunt volgen, klikt u op **Dashboards > Pijplijndashboards**.

Resultaten

Gefeliciteerd! U hebt een pijplijn gemaakt die uw code van een GitHub- of GitLab-opslagplaats continu integreert in uw pijplijn en uw buildimage implementeert.

Wat nu te doen

Zie [Meer resources voor Code Stream-beheerders en -ontwikkelaars](#) voor meer informatie.

Hoe kan ik de release van een applicatie die ik implementeer vanuit een YAML-cloudsjabloon in Code Stream automatiseren?

Als ontwikkelaar hebt u een pijplijn nodig die een Automation-cloudsjabloon ophaalt van een GitHub-instantie op locatie telkens wanneer u een wijziging doorvoert. U hebt de pijplijn nodig

om een WordPress-applicatie te implementeren op Amazon Web Services (AWS) EC2 of een datacentrum. Code Stream roept de cloudsjabloon aan vanuit de pijplijn en automatiseert de continue integratie en continue levering (CICD) van die cloudsjabloon voor de implementatie van uw applicatie.

Als u uw pijplijn wilt maken en activeren, hebt u een VMware Cloud-sjabloon nodig.

Voor **Bron van cloudsjabloon** in uw Code Streamcloudsjabloontaak kunt u een van de volgende selecteren:

- **Cloud Assembly Templates** als broncontrole. In dit geval hebt u geen GitLab- of GitHub-opslagplaats nodig.
- **Bronbeheer** als u GitLab of GitHub voor bronbeheer gebruikt. In dit geval moet u een Git-webhook hebben en de pijplijn activeren via de webhook.

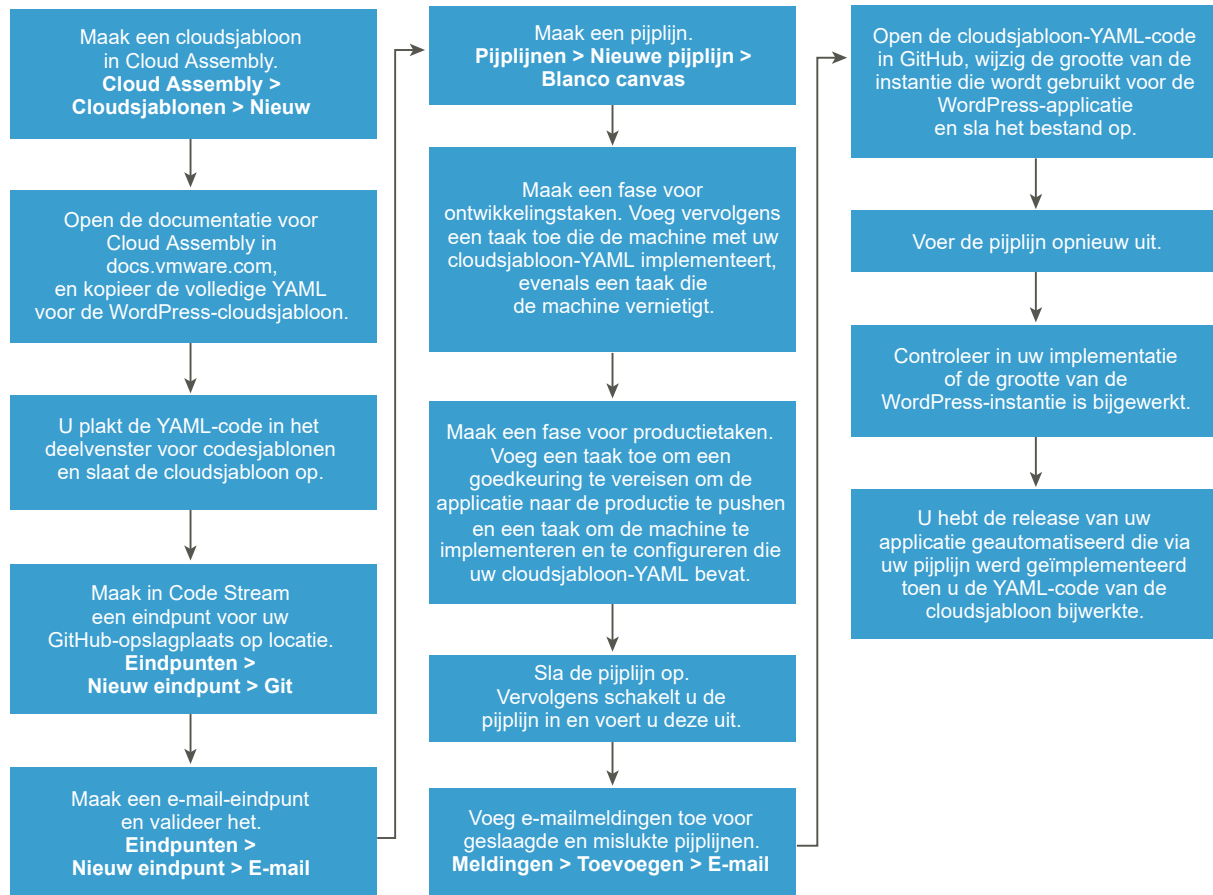
Als u een YAML-cloudsjabloon in uw GitHub-opslagplaats hebt en u deze cloudsjabloon in uw pijplijn wilt gebruiken, moet u het volgende doen.

- 1 Push de cloudsjabloon in Cloud Assembly naar uw GitHub-opslagplaats.
- 2 Maak in Code Stream een Git-eindpunt. Maak vervolgens een Git-webhook die uw Git-eindpunt en uw pijplijn gebruikt.
- 3 Als u uw pijplijn wilt activeren, moet u alle bestanden in uw GitHub-opslagplaats bijwerken en uw wijziging doorvoeren.

Als u geen YAML-cloudsjabloon in uw GitHub-opslagplaats hebt en u een cloudsjabloon van broncontrole wilt gebruiken, gebruikt u de volgende procedure om te leren hoe u dit kunt doen. U ziet hoe u een cloudsjabloon voor een WordPress-applicatie maakt en deze vanuit een GitHub-opslagplaats op locatie kunt activeren. Wanneer u een wijziging aanbrengt in de YAML-cloudsjabloon wordt de pijplijn geactiveerd en wordt de release van uw applicatie geautomatiseerd.

- In Cloud Assembly voegt u een cloudaccount toe, voegt u een cloudzone toe en maakt u de cloudsjabloon.
- In Code Stream voegt u een eindpunt toe voor de GitHub-opslagplaats op locatie waarop uw cloudsjabloon wordt gehost. Vervolgens voegt u de cloudsjabloon toe aan uw pijplijn.

In dit gebruiksscenario wordt uitgelegd hoe u een cloudsjabloon kunt gebruiken vanuit een GitHub-opslagplaats op locatie.

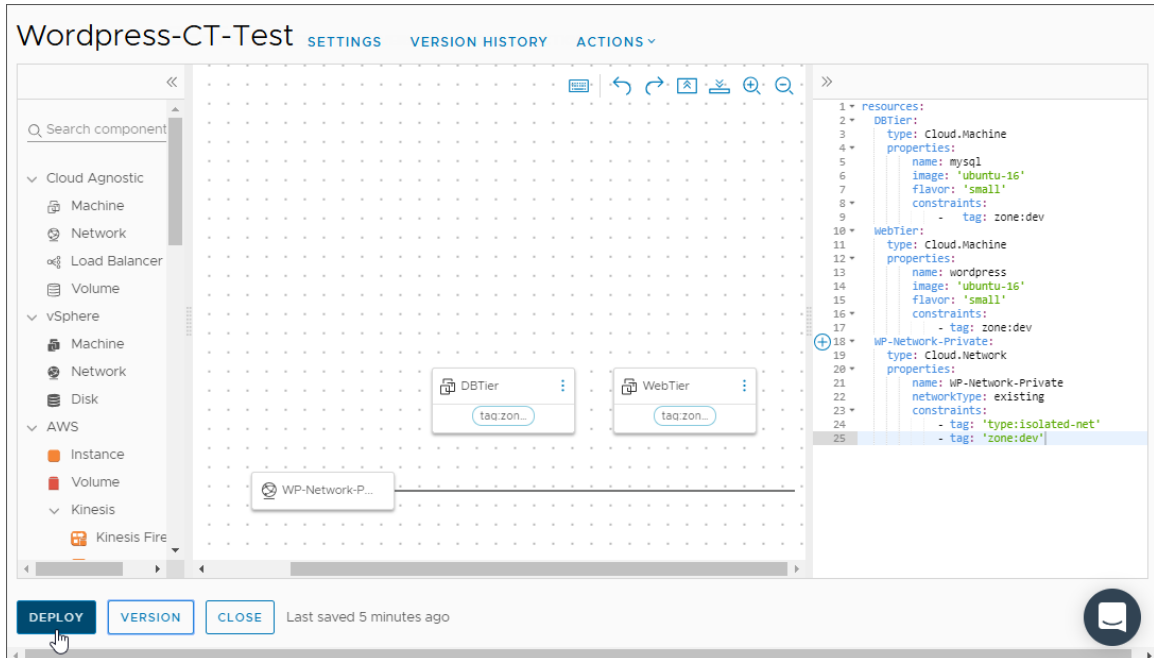


Voorwaarden

- Voeg een cloudaccount en een cloudzone toe aan uw vRealize Automation Cloud Assembly-infrastructuur. Zie de documentatie voor de vRealize Automation Cloud Assembly.
- Als u uw cloudsjabloon wilt maken in de volgende procedure, kopieert u de YAML-code van WordPress naar uw klembord. Zie de cloudsjabloon-YAML-code in het WordPress-gebruiksscenario in de documentatie voor vRealize Automation Cloud Assembly.
- Voeg de YAML-code voor de WordPress-applicatie toe aan uw GitHub-instantie.
- Voeg een webhook toe voor de Git-trigger zodat uw pijplijn uw YAML-code kan ophalen wanneer u deze bijwerkt en uw wijzigingen doorvoert. Klik in Code Stream op **Triggers > Git > Webhooks voor Git**.
- Als u met een cloudsjabloontaak wilt werken, moet u een van de Cloud Assembly-rollen hebben.

Procedure

- 1 In Cloud Assembly voert u deze stappen uit.
 - a Klik op **VMware Cloud Templates** en maak vervolgens een cloudsjabloon en een implementatie voor de WordPress-applicatie.
 - b Plak de WordPress YAML-code die u naar uw klembord hebt gekopieerd in uw cloudsjabloon en implementeer deze.



2 In Code Stream maakt u eindpunten.

- a Maak een Git-eindpunt voor uw GitHub-opslagplaats op locatie waar uw YAML-bestand zich bevindt.
- b Voeg een e-maileindpunt toe dat gebruikers op de hoogte kan stellen van de pijplijnstatus wanneer deze wordt uitgevoerd.

Add Endpoint

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

CREATE

VALIDATE

CANCEL

3 Maak een pijplijn en voeg meldingen toe voor geslaagde en mislukte pijplijnactiviteiten.

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 Voeg een fase voor ontwikkeling toe en voeg een cloudsjabloontaak toe.

- a Voeg een cloudsjabloontaak toe die de machine implementeert en configureer de taak om de cloudsjabloon-YAML voor de WordPress-applicatie te gebruiken.

```
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 Voeg een cloudsjabloontaak toe die de machine vernietigt om resources vrij te maken.

- 5 Voeg een fase voor productie toe en voeg goedkeurings- en implementatietaken toe.
 - a Voeg een gebruikersbewerkingstaak toe om goedkeuring te vereisen om de WordPress-applicatie naar productie te pushen.
 - b Voeg een cloudsjabloontaak toe om de machine te implementeren en configureer deze met de cloudsjabloon-YAML voor de WordPress-applicatie.

Wanneer u **Maken** kiest, moet de implementatienaam uniek zijn. Als u de naam leeg laat, zal Code Stream een unieke willekeurige naam toewijzen.

Dit is wat u moet weten als u **Terugdraaien** kiest in uw eigen gebruiksscenario: als u de actie **Terugdraaien** selecteert en een **Terugdraaierversie** invoert, moet de versie de indeling **n-x** hebben. Bijvoorbeeld **n-1**, **n-2**, **n-3**, enzovoort. Als u de implementatie op een andere locatie dan Code Stream maakt en bijwerkt, is terugdraaien niet toegestaan.

Wanneer u zich aanmeldt bij Code Stream krijgt deze een gebruikerstoken, dat 30 minuten geldig is. Voor langdurige pijplijnperiodes, wanneer de taak vóór de cloudsjabloontaak 30 minuten of langer duurt om uit te voeren, zal het gebruikerstoken vervallen. Hierdoor mislukt de cloudsjabloontaak.

Om ervoor te zorgen dat uw pijplijn langer dan 30 minuten kan worden uitgevoerd, kunt u een optioneel API-token invoeren. Wanneer Code Stream de cloudsjabloon aanroept, blijft het API-token behouden en blijft de cloudsjabloontaak het API-token gebruiken.

Wanneer u het API-token als variabele gebruikt, wordt het versleuteld. Anders wordt het gebruikt als tekst zonder opmaak.

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

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

6 Voer de pijplijn uit.

Om te controleren of elke taak is voltooid, klikt u op de taak die wordt uitgevoerd en onderzoekt u de status in de implementatiegegevens om gedetailleerde resourcegegevens te zien.

7 Wijzig in GitHub de soort van de WordPress-serverinstantie van `small` in `medium`.

Wanneer u wijzigingen doorvoert, wordt de pijplijn geactiveerd. De pijplijn haalt uw bijgewerkte code uit de GitHub-opslagplaats en bouwt uw applicatie.

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

8 Voer de pijplijn opnieuw uit, controleer of deze is gelukt en of de versie van de WordPress-instantie is gewijzigd van `small` in `medium`.**Resultaten**

Gefeliciteerd! U heeft de release van uw applicatie geautomatiseerd die u heeft geïmplementeerd vanuit een YAML-cloudsjabloon.

Wat nu te doen

Zie [Hoofdstuk 5 Tutorials voor het gebruik van Code Stream](#) voor meer informatie over hoe u Code Stream kunt gebruiken.

Zie [Meer resources voor Code Stream-beheerders en -ontwikkelaars](#) voor aanvullende referenties.

Hoe kan ik de release van een applicatie in Code Stream op een Kubernetes-cluster automatiseren?

Als Code Stream-beheerder of -ontwikkelaar kunt u Code Stream en VMware Tanzu Kubernetes Grid Integrated Edition (voorheen VMware Enterprise PKS genoemd) gebruiken om de implementatie van uw softwareapplicaties naar een Kubernetes-cluster te automatiseren. Dit toepassingsvoorbeeld vermeldt andere methoden die u kunt gebruiken om de release van uw applicatie te automatiseren.

In dit toepassingsvoorbeeld maakt u een pijplijn die twee fasen bevat en gebruikt u Jenkins om uw applicatie te bouwen en te implementeren.

- De eerste fase is voor ontwikkeling. Deze gebruikt Jenkins om uw code uit een tak in uw GitHub-opslagplaats te halen en deze vervolgens te bouwen, te testen en te publiceren.

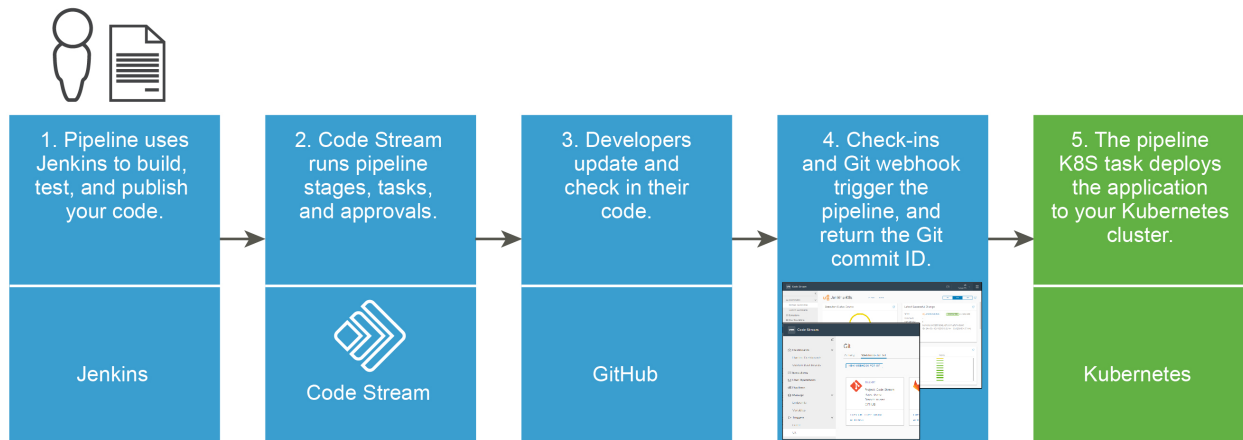
- De tweede fase is voor de implementatie. Er wordt een gebruikersbewerkingstaak uitgevoerd die moet worden goedgekeurd door belangrijke gebruikers voordat de pijplijn uw applicatie naar uw Kubernetes-cluster kan implementeren.

Wanneer u een Kubernetes API-eindpunt in de pijplijnwerkplek gebruikt, maakt Code Stream de nodige Kubernetes-resources zoals ConfigMap, Geheim en Pod om de taak voor continue integratie (CI) of de aangepaste taak uit te voeren. Code Stream communiceert met de container via de NodePort.

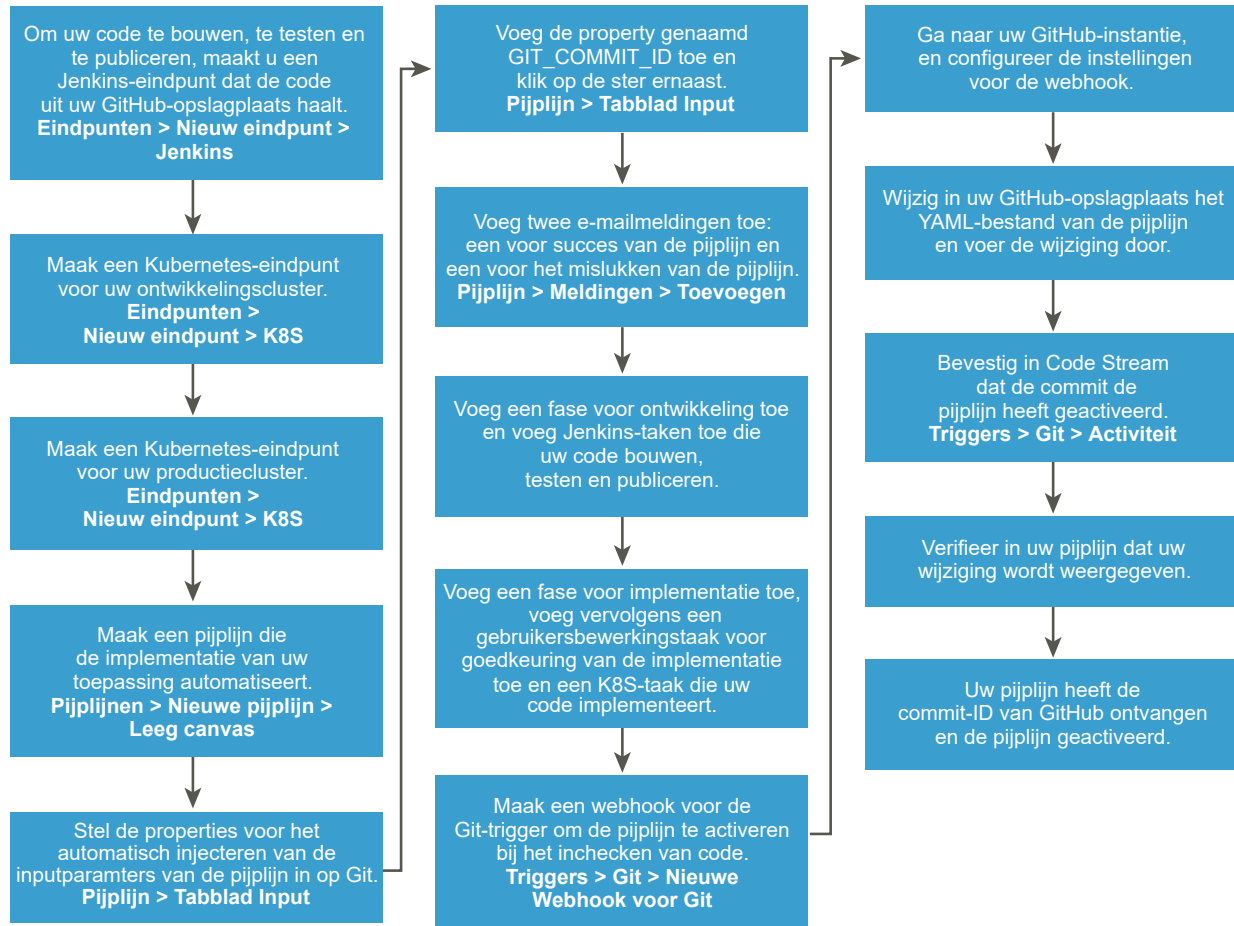
Als u gegevens wilt delen tussen pijplijnuitvoeringen, moet u een permanente volumeclaim opgeven, en Code Stream koppelt de permanente volumeclaim aan de container om de gegevens op te slaan en deze te gebruiken voor volgende pijplijnuitvoeringen.

De Code Stream-pijplijnwerkplek ondersteunt Docker en Kubernetes voor taken voor continue integratie en aangepaste taken.

Zie [De pijplijnwerkplek configureren](#) voor meer informatie over het configureren van de werkplek.



De ontwikkelingstools, implementatie-instanties en het YAML-bestand van de pijplijn moeten beschikbaar zijn zodat uw pijplijn uw applicatie kan bouwen, testen, publiceren en implementeren. De pijplijn implementeert uw applicatie naar ontwikkelings- en productie-instanties van Kubernetes-clusters op AWS.



Andere methoden die de release van uw applicatie automatiseren:

- In plaats van uw applicatie met Jenkins te bouwen kunt u de systeemeigen bouwmogelijkheid van Code Stream en een Docker-buildhost gebruiken.
- In plaats van uw applicatie in een Kubernetes-cluster te implementeren kunt u deze ook implementeren in een Amazon Web Services-cluster (AWS).

Zie voor meer informatie over het gebruik van de systeemeigen bouwmogelijkheid van Code Stream en een Docker-host:

- Een systeemeigen CI/CD-build plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt
- Een CI/CD-systeemeigen build plannen in Code Stream voordat u handmatig taken toevoegt

Voorwaarden

- Controleer of de applicatiecode die moet worden geïmplementeerd, zich in een werkende GitHub-opslagplaats bevindt.
- Controleer of u over een werkende instantie van Jenkins beschikt.
- Controleer of u over een werkende e-mailserver beschikt.

- Maak in Code Stream een e-mail-eindpunt dat verbinding maakt met uw e-mailserver.
- Stel twee Kubernetes-clusters in op Amazon Web Services (AWS) voor ontwikkeling en productie, waar uw-pijplijn uw applicatie zal implementeren.
- Controleer of de GitHub-opslagplaats de YAML-code voor uw pijplijn bevat of een YAML-bestand waarmee de metagegevens en specificaties voor uw omgeving worden gedefinieerd.

Procedure

- 1 Klik in Code Stream op **Eindpunten > Nieuw eindpunt** en maak een Jenkins-eindpunt dat u in uw pijplijn zult gebruiken om code uit uw GitHub-opslagplaats te halen.
- 2 Als u Kubernetes-eindpunten wilt maken, klikt u op **Nieuw eindpunt**.
 - a Maak een eindpunt voor uw Kubernetes-cluster voor ontwikkeling.
 - b Maak een eindpunt voor uw Kubernetes-cluster voor productie.

De URL voor uw Kubernetes-cluster kan een poortnummer bevatten of niet.

Bijvoorbeeld:

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

```
https://api.kubernetesserver.fa2c1d78-9f00-4e30-8268-4ab81862080d.k8s-user.com
```
- 3 Maak een pijplijn die een container van uw applicatie, zoals WordPress, implementeert op uw Kubernetes-cluster voor ontwikkeling en stel de inputproperties voor de pijplijn in.
 - a Om ervoor te zorgen dat uw pijplijn een code-commit in GitHub herkent die de pijplijn activeert, klikt u in de pijplijn op het tabblad **Input** en selecteert u **Proprieties automatisch injecteren**.
 - b Voeg de eigenschap met de naam **GIT_COMMIT_ID** toe en klik op de bijbehorende ster.

Wanneer de pijplijn wordt uitgevoerd, wordt in de pijplijn-uitvoering de commit-ID weergegeven die door de Git-trigger wordt geretourneerd.

- 4 Voeg meldingen toe om een e-mail te verzenden wanneer de pijplijn slaagt of mislukt.
 - a Klik in de pijplijn op het tabblad **Meldingen** en klik op **Toevoegen**.
 - b Om een e-mailmelding toe te voegen voor het voltooiën van de pijplijn, selecteert u **E-mail** en selecteert u **Voltooit**. Selecteer vervolgens de e-mailserver, voer de e-mailadressen in en klik op **Opslaan**.
 - c Om nog een e-mailmelding toe te voegen voor een pijplijnfout, selecteert u **Mislukt** en klikt u op **Opslaan**.

Notification

Send notification type

☒ Email
 ☐ Ticket
 ☐ Webhook

When pipeline

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

Email server ⓘ *

--Select Email server-- ▾

Send Email

To ⓘ \$ *

Email IDs of recipients

Subject \$ *

Email Subject

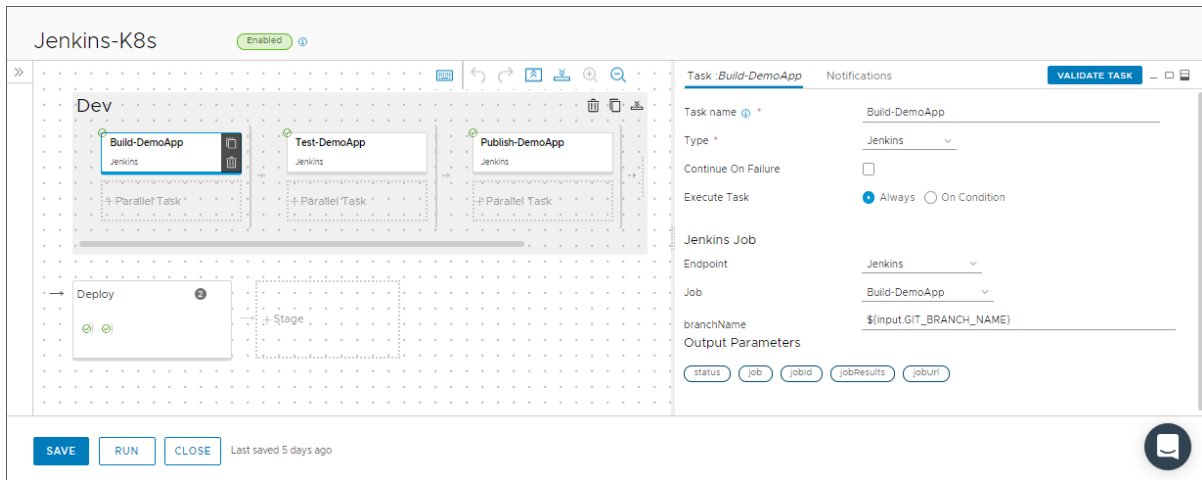
Body ⓘ \$ *

1

CANCEL

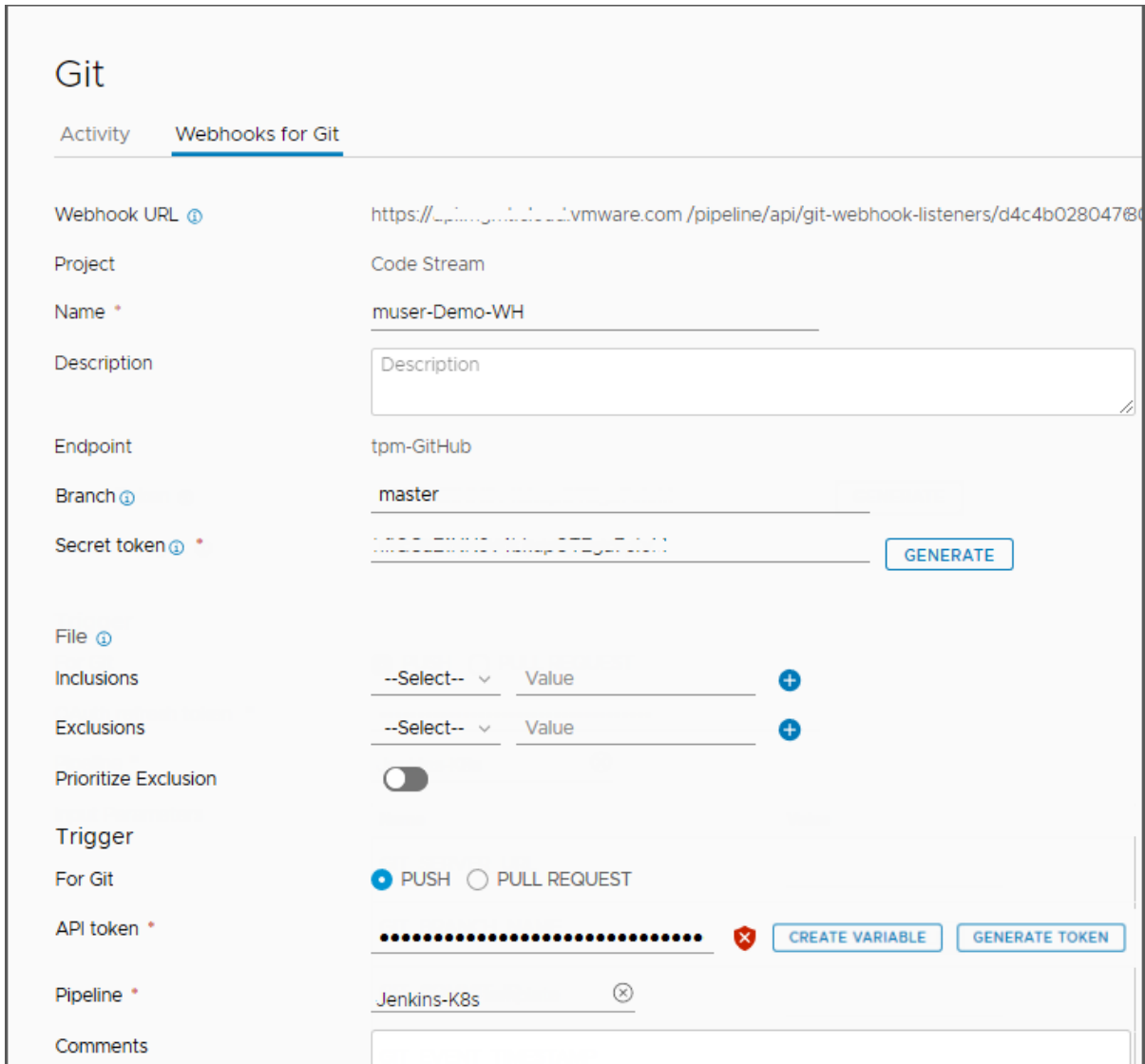
SAVE

- 5 Voeg een ontwikkelingsfase toe aan uw pijplijn en voeg taken toe die uw applicatie bouwen, testen en publiceren. Vervolgens valideert u elke taak.
 - a Om uw applicatie te bouwen, voegt u een Jenkins-taak toe die het Jenkins-eindpunt gebruikt en voert u een bouwtaak uit op de Jenkins-server. Voer vervolgens de Git-tak in deze vorm in, zodat de pijplijn uw code kan ophalen: `${input.GIT_BRANCH_NAME}`
 - b Om uw applicatie te testen, voegt u een Jenkins-taak toe die hetzelfde Jenkins-eindpunt gebruikt, en voert u een testtaak uit op de Jenkins-server. Voer vervolgens dezelfde Git-tak in.
 - c Als u uw applicatie wilt publiceren, voegt u een Jenkins-taak toe die hetzelfde Jenkins-eindpunt gebruikt, en voert u een publicatietaak uit vanaf de Jenkins-server. Voer vervolgens dezelfde Git-tak in.



- 6 Voeg een implementatiefase toe aan uw pijplijn en voeg vervolgens een taak toe die een goedkeuring vereist voor de implementatie van uw applicatie en een andere taak die de applicatie implementeert op uw Kubernetes-cluster. Vervolgens valideert u elke taak.
 - a Om een goedkeuring te vereisen voor de implementatie van uw applicatie, voegt u een gebruikersbewerkingstaak toe, voegt u de e-mailadressen toe voor de gebruikers die deze moeten goedkeuren en voert u een bericht in. Schakel vervolgens **E-mail verzenden** in.
 - b Als u uw applicatie wilt implementeren, voegt u een Kubernetes-taak toe. Selecteer vervolgens in de properties van de Kubernetes-taak uw Kubernetes-cluster voor ontwikkeling, selecteer de actie **Maken** en selecteer de ladingsbron **Lokale definitie**. Selecteer vervolgens uw lokale YAML-bestand.

- 7 Voeg een Git-webhook toe die Code Stream in staat stelt om de Git-trigger te gebruiken, waarmee de pijplijn wordt geactiveerd wanneer ontwikkelaars hun code doorvoeren.



- 8 Als u uw pijplijn wilt testen, gaat u naar uw GitHub-opslagplaats, werkt u het YAML-bestand van uw applicatie bij en voert u de wijziging door.
- Controleer in Code Stream of de commit wordt weergegeven.
 - Klik op **Triggers > Git > Activiteit**.
 - Zoek naar de trigger van uw pijplijn.
 - Klik op **Dashboards > Pijplijndashboards**.
 - Zoek in het pijplijndashboard naar de GIT_COMMIT_ID in het laatste gelukke wijzigingsgebied.
- 9 Controleer de pijplijncode en verifieer of de wijziging wordt weergegeven.

Resultaten

Gefeliciteerd! U heeft de implementatie van uw softwareapplicatie op uw Kubernetes-cluster geautomatiseerd.

Voorbeeld: Voorbeeld van een pijplijn-YAML die een applicatie implementeert op een Kubernetes-cluster

Voor het type pijplijn dat in dit voorbeeld wordt gebruikt, lijkt de YAML op de volgende code:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
eyJzeWlwaG9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJlc2VybmFtZSI6InRhbmRvLWJldGEyIiwicGFzc3dvcmQI Oi
JhRGstcmVOLWlUQil1IejciLCJlbWFPbCI6InRhbmRvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoizEdGdVoyOHRZbVYw
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

```

Wat nu te doen

Als u uw softwareapplicatie wilt implementeren op uw Kubernetes-cluster voor productie, voert u de stappen opnieuw uit en selecteert u uw productiecluster.

Voor meer informatie over het integreren van Code Stream met Jenkins, raadpleegt u [Hoe integreer ik Code Stream met Jenkins?](#).

Hoe implementeer ik mijn applicatie in Code Stream naar mijn Blue-Green-implementatie?

Blue-Green is een implementatiemodel dat gebruikmaakt van twee Docker-hosts die u op identieke wijze implementeert en configureert in een Kubernetes-cluster. Met het Blue-Green-implementatiemodel verlaagt u de uitvaltijd die in uw omgeving kan optreden wanneer uw pijplijnen in Code Stream uw applicaties implementeren.

De Blue-Green-instanties in uw implementatiemodel dienen elk een ander doel. Slechts één instantie tegelijk accepteert het live-verkeer dat uw applicatie implementeert en elke instantie accepteert dat verkeer op specifieke momenten. De Blue-instantie ontvangt de eerste versie van uw applicatie en de Green-instantie ontvangt de tweede.

De load balancer in uw Blue-Green-omgeving bepaalt welke routing het live-verkeer volgt terwijl het uw applicatie implementeert. Door het Blue-Green model te gebruiken blijft uw omgeving operationeel, bemerken gebruikers geen uitvaltijd en wordt uw applicatie voortdurend door uw pijplijn geïntegreerd en naar uw productieomgeving geïmplementeerd.

De pijplijn die u maakt in Code Stream vertegenwoordigt uw Blue-Green-implementatiemodel in twee fasen. Eén fase is voor ontwikkeling en de andere fase is voor productie.

De Code Stream-pijplijnwerkplek ondersteunt Docker en Kubernetes voor taken voor continue integratie en aangepaste taken.

Zie [De pijplijnwerkplek configureren](#) voor meer informatie over het configureren van de werkplek.

Tabel 5-2. Taken in de ontwikkelingsfase voor Blue-Green-implementatie

Taaktype	Taak
Kubernetes	Maak een naamruimte voor uw Blue-Green-implementatie.
Kubernetes	Maak een geheime sleutel voor Docker Hub.
Kubernetes	Maak de service die wordt gebruikt om de applicatie te implementeren.
Kubernetes	Maak de Blue-implementatie.
Poll	Controleer de Blue-implementatie.
Kubernetes	Verwijder de naamruimte.

Tabel 5-3. Taken in de productiefase voor Blue-Green-implementatie

Taaktype	Taak
Kubernetes	Green krijgt de servicegegevens van Blue.
Kubernetes	Haal de details op voor de Green-replicaset.
Kubernetes	Maak de Green-implementatie en gebruik de geheime sleutel om de containerimage op te halen.
Kubernetes	Werk de service bij.
Poll	Controleer of de implementatie is gelukt op de productie-URL.
Kubernetes	Voltooi de Blue-implementatie.
Kubernetes	Verwijder de Blue-implementatie.

Als u uw applicatie in uw eigen Blue-Green implementatiemodel wilt implementeren, maakt u een pijplijn in Code Stream die twee fasen bevat. De eerste fase bevat de Blue-taken die uw applicatie implementeren op de Blue-instantie en de tweede fase bevat de Green-taken die uw applicatie implementeren op de Green-instantie.

U kunt uw pijplijn maken met behulp van de slimme CICD-pijplijnsjabloon. De sjabloon maakt uw pijplijnfasen en -taken voor u en omvat de implementatieselecties.

Als u handmatig een pijplijn maakt, moet u de pijplijnfasen plannen. Zie [Een CICD-systeemeigen build plannen in Code Stream voordat u handmatig taken toevoegt](#) voor een voorbeeld.

In dit voorbeeld gebruikt u de slimme CICD-pijplijnsjabloon om uw Blue-Green-pijplijn te maken.

Voorwaarden

- Controleer of u toegang heeft tot een werkend Kubernetes-cluster op AWS.
- Controleer of u een Blue-Green-implementatieomgeving heeft ingesteld en dat u de Blue-Green-instanties heeft geconfigureerd om identiek te zijn.

- Maak een Kubernetes-eindpunt in Code Stream dat uw applicatie-image implementeert op het Kubernetes-cluster op AWS.
- Raak vertrouwd met het gebruik van de slimme CICD-pijplijnsjabloon. Zie [Een systeemeigen CICD-build plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt](#).

Procedure

- 1 Klik op **Pijplijnen > Nieuwe pijplijn > Slimme sjablonen > CI/CD-sjabloon**.
- 2 Voer de informatie voor het CI-gedeelte van de slimme CICD-pijplijnsjabloon in en klik op **Volgende**.

Zie [Een systeemeigen CICD-build plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt](#) voor hulp.

- 3 Voltooi het CD-gedeelte van de slimme pijplijnsjabloon
 - a Selecteer de omgevingen voor uw applicatie-implementatie. Bijvoorbeeld: **Ontwikkeling** en **Productie**.
 - b Selecteer de service die door de pijplijn wordt gebruikt voor de implementatie.
 - c Selecteer in het implementatiegebied het clustereindpunt voor de ontwikkelingsomgeving en de productieomgeving.
 - d Voor het productie-implementatiemodel selecteert u **Blue-Green** en klikt u op **Maken**.

Smart Template: CI/CD

Step 2 of 2

Environment * ☒ Dev ☒ Prod

K8s YAML files * SELECT PROCESS

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 *

CREATE BACK CANCEL



Resultaten

Gefeliciteerd! U heeft de slimme pijplijnsjabloon gebruikt om een pijplijn te maken die uw applicatie implementeert op uw Blue-Green-instanties in uw Kubernetes-cluster voor productie op AWS.

Voorbeeld: Voorbeeld van YAML-code voor bepaalde Blue-Green-implementatietaken

De YAML-code die wordt weergegeven in Kubernetes-pijplijntaken voor uw Blue-Green-implementatie, kan op de volgende voorbeelden lijken die de Naamruimte, Service en Implementatie maken. Als u een image moet downloaden uit een privéopslagplaats, moet het YAML-bestand een sectie bevatten met het Docker-configuratiegeheim. Zie het CD-gedeelte van [Een systeemeigen CI/CD-build plannen in Code Stream voordat u de slimme pijplijnsjabloon gebruikt](#).

Nadat de slimme pijplijnsjabloon uw pijplijn heeft gemaakt, kunt u de taken wijzigen zoals nodig voor uw eigen implementatie.

YAML-code om een voorbeeld-naamruimte te maken:

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

YAML-code om een voorbeeld-service te maken:

```
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
```

YAML-code om een voorbeeld-implementatie te maken:

```
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
```

Wat nu te doen

Zie [Hoofdstuk 5 Tutorials voor het gebruik van Code Stream](#) voor meer informatie over hoe u Code Stream kunt gebruiken.

Zie [Hoe kan ik mijn implementatie terugdraaien in Code Stream?](#) om een implementatie terug te draaien.

Zie [Meer resources voor Code Stream-beheerders en -ontwikkelaars](#) voor aanvullende referenties.

Hoe kan ik mijn eigen tools voor bouwen, testen en implementeren integreren met Code Stream?

Als DevOps-beheerder of -ontwikkelaar kunt u custom scripts maken om de mogelijkheden van Code Stream uit te breiden.

Met uw script kunt u Code Stream integreren met uw eigen tools voor continue integratie (CI) en continue levering (CD) en API's die uw applicaties bouwen, testen en implementeren. Custom scripts zijn vooral handig als u de API's van de applicatie niet openbaar weergeeft.

Uw aangepast script kan bijna alles doen wat u nodig heeft voor uw tools voor bouwen, testen en implementeren om te integreren met Code Stream. Uw script kan bijvoorbeeld samenwerken met uw pijplijnwerkplek om ondersteuning te bieden voor taken voor continue integratie te ondersteunen die uw applicatie bouwen en testen, en voor taken voor continue levering die uw applicatie implementeren. Het kan een bericht verzenden naar Slack wanneer een pijplijn is voltooid en nog veel meer.

De Code Stream-pijplijnwerkplek ondersteunt Docker en Kubernetes voor taken voor continue integratie en aangepaste taken.

Zie [De pijplijnwerkplek configureren](#) voor meer informatie over het configureren van de werkplek.

U schrijft uw custom script in een van de ondersteunde talen. In het script neemt u uw bedrijfslogica op en definieert u de input en output. Outputtypen kunnen onder meer getal, tekenreeks, tekst en wachtwoord zijn. U kunt meerdere versies van een custom script maken met verschillende bedrijfslogica, invoer en uitvoer.

U laat uw pijplijn een versie van uw script in een custom taak uitvoeren. De scripts die u maakt, bevinden zich in uw Code Stream-instantie.

Wanneer een pijplijn een custom integratie gebruikt, wordt een foutbericht weergegeven en wordt aangegeven dat u het niet kunt verwijderen als u de custom integratie probeert te verwijderen.

Als u een custom integratie verwijdert, worden alle versies van uw custom script verwijderd. Als u een bestaande pijplijn heeft met een custom taak die een willekeurige versie van het script gebruikt, zal deze pijplijn mislukken. Om ervoor te zorgen dat bestaande pijplijnen niet mislukken, kunt u de versie van uw script die u niet langer wilt gebruiken, afkeuren en intrekken. Als geen pijplijn die versie gebruikt, kunt u deze verwijderen.

Tabel 5-4. Wat u doet nadat u het custom script heeft geschreven

Wat u doet...	Meer informatie over deze actie...
Voeg een custom taak toe aan uw pijplijn.	De custom taak: <ul style="list-style-type: none"> ■ wordt uitgevoerd op dezelfde container als andere CI-taken in uw pijplijn. ■ bevat input- en outputvariabelen die door uw script worden gevuld voordat de pijplijn de custom taak uitvoert. ■ ondersteunt meerdere gegevenstypen en verschillende typen metagegevens die u definieert als input en output in uw script.
Selecteer uw script in de custom taak.	U declareert de input- en outputproperties in het script.
Sla de pijplijn op. Vervolgens schakelt u de pijplijn in en voert u deze uit.	Wanneer de pijplijn wordt uitgevoerd, roept de custom taak de versie van de gespecificeerde script aan en voert het de bedrijfslogica erin uit die uw tool voor bouwen, testen en implementeren met Code Stream integreert.
Bekijk de uitvoeringen nadat uw pijplijn is uitgevoerd.	Controleer of de pijplijn de verwachte resultaten heeft geleverd.

Wanneer u een aangepaste taak gebruikt die een versie van de aangepaste integratie aanroept, kunt u aangepaste omgevingsvariabelen als naam-waardeparen opnemen op het tabblad **Workspace** van de pijplijn. Wanneer de image van de opbouwfunctie de Workspace-container maakt die de CI-taak uitvoert en uw image implementeert, worden de omgevingsvariabelen door Code Stream doorgegeven aan die container.

Wanneer uw Code Stream-instantie bijvoorbeeld een webproxy vereist en u een Docker-host gebruikt om een container voor een aangepaste integratie te maken, wordt de pijplijn door Code Stream uitgevoerd en worden de variabelen van de webproxyinstelling doorgegeven aan die container.

Tabel 5-5. Voorbeeld van naam-waardeparen voor omgevingsvariabelen

Naam	Waarde
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

Tabel 5-5. Voorbeeld van naam-waardeparen voor omgevingsvariabelen (vervolg)

Naam	Waarde
http_proxy	http://10.0.0.254:1234
PAD	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Naam-waardeparen worden als volgt weergegeven in de gebruikersinterface:



Dit voorbeeld maakt een custom integratie die Code Stream verbindt met uw Slack-instantie en een bericht op een Slack-kanaal plaatst.

Voorwaarden

- Verifieer voor het schrijven van een aangepast script of u een van deze talen heeft: Python 2, Python 3, Node.js of een van de Shell-talen: Bash, sh of zsh.
- Genereer een containerimage met behulp van de geïnstalleerde Node.js of de Python-runtime.

Procedure

1 Maak de custom integratie.

- Klik op **Custom integraties > Nieuw** en voer een relevante naam in.
- Selecteer de gewenste runtimeomgeving.
- Klik op **Maken**.

Uw script wordt geopend en toont de code, die de vereiste runtimeomgeving omvat. Bijvoorbeeld: `runtime: "nodejs"`. Het script moet de runtime bevatten, die door de builderimage wordt gebruikt, zodat de custom taak die u aan uw pijplijn toevoegt, slaagt wanneer de pijplijn wordt uitgevoerd. Anders mislukt de custom taak.

De belangrijkste gebieden van uw custom integratie-YAML zijn onder meer de runtime, code, invoereigenschappen en uitvoereigenschappen. In deze procedure worden verschillende typen en syntaxis uitgelegd.

YAML-sleutels voor custom integratie	Beschrijving
<code>runtime</code>	Taakruntimeomgeving waarin Code Stream de code uitvoert. Dit kan een van de volgende hoofdlettergevoelige tekenreeksen zijn: <ul style="list-style-type: none"> ■ <code>nodejs</code> ■ <code>python2</code> ■ <code>python3</code> ■ <code>shell</code> Als er niets wordt opgegeven, is <code>shell</code> de veronderstelde standaardwaarde.
<code>code</code>	Custom bedrijfslogica die wordt uitgevoerd als onderdeel van de custom taak.
<code>inputProperties</code>	Reeks invoereigenschappen die moeten worden vastgelegd als onderdeel van de configuratie van de custom taak. Deze eigenschappen worden normaal gesproken gebruikt in de code.
<code>outputProperties</code>	Reeks uitvoereigenschappen die u kunt exporteren vanaf de custom taak om door te geven aan de pijplijn.

- 2 Declareer de inputproperties in uw script met behulp van de beschikbare gegevenstypen en metagegevens.

De inputproperties worden als context doorgegeven aan uw script in het `code:`-gedeelte van de YAML.

YAML-invoersleutels voor custom taak	Beschrijving	Vereist
<code>type</code>	Invoertypen om te renderen: <ul style="list-style-type: none"> ■ <code>text</code> ■ <code>textarea</code> ■ <code>number</code> ■ <code>checkbox</code> ■ <code>password</code> ■ <code>select</code> 	Ja
<code>name</code>	Naam of tekenreeks van de invoer voor de custom taak, die wordt geïnjecteerd in de YAML-code voor de custom integratie. Moet uniek zijn voor elke invoereigenschap die voor een custom integratie is gedefinieerd.	Ja
<code>title</code>	Teksttekenreekslabel van de invoereigenschap voor de custom taak op het pijplijnmodelcanvas. Als dit veld leeg blijft, wordt <code>name</code> standaard gebruikt.	Nee
<code>required</code>	Bepaalt of een gebruiker de invoereigenschap moet invoeren wanneer hij de custom taak configureert. Stel in op waar of onwaar. Indien waar, als een gebruiker geen waarde opgeeft wanneer hij de custom taak op het pijplijncanvas configureert, blijft de status van de taak ongeconfigureerd.	Nee

YAML- invoersleutels voor custom taak	Beschrijving	Vereist
placeholder	Standaardtekst voor het veld van de invoereigenschap als er geen waarde aanwezig is. Wordt toegewezen aan het kenmerk voor de tijdelijke HTML-aanduiding. Wordt alleen ondersteund voor bepaalde typen invoereigenschappen.	Nee
defaultValue	Dit is de standaardwaarde die het veld van de invoereigenschap vult wanneer de custom taak wordt weergegeven op de pijplijnmodelpagina.	Nee
bindable	Bepaalt of de invoereigenschap dollartekenvariabelen accepteert bij het modelleren van de custom taak op het pijplijncanvas. Hiermee voegt u de \$-indicator naast de titel toe. Wordt alleen ondersteund voor bepaalde typen invoereigenschappen.	Nee
labelMessage	Tekenreeks die fungeert als hulpknopinfo voor gebruikers. Voegt een pictogram voor knopinfo i toe naast de titel van de invoer.	Nee
enum	<p>Haalt een reeks waarden op die de opties voor het selecteren van de invoereigenschap weergeven. Alleen ondersteund voor bepaalde typen invoereigenschappen.</p> <p>Wanneer een gebruiker een optie selecteert en deze opslaat voor de custom taak, komt de waarde van inputProperty overeen met deze waarde en wordt deze weergegeven in de custom taakmodellering.</p> <p>Bijvoorbeeld: de waarde 2015.</p> <ul style="list-style-type: none"> ■ 2015 ■ 2016 ■ 2017 ■ 2018 ■ 2019 ■ 2020 	Nee
options	<p>Neemt een reeks objecten met behulp van optionKey en optionValue.</p> <ul style="list-style-type: none"> ■ optionKey. Waarde die is doorgegeven aan het codegedeelte van de taak. ■ optionValue. Tekenreeks die de optie in de gebruikersinterface weergeeft. <p>Alleen ondersteund voor bepaalde typen invoereigenschappen.</p> <p>Opties:</p> <p>optionKey: key1. De waarde van deze inputProperty komt overeen met key1 in de codesectie wanneer deze is geselecteerd en opgeslagen voor de aangepaste taak.</p> <p>optionValue: 'Label for 1'. Weergegeven waarde weer voor key1 in de gebruikersinterface. Deze wordt nergens anders weergegeven voor de custom taak.</p> <p>optionKey: key2</p> <p>optionValue: 'Label for 2'</p> <p>optionKey: key3</p> <p>optionValue: 'Label for 3'</p>	Nee

YAML- invoersleutels voor custom taak	Beschrijving	Vereist
minimum	Heeft een getal dat fungeert als de minimumwaarde die geldig is voor deze invoereigenschap. Wordt alleen ondersteund voor de invoereigenschap van het type getal.	Nee
maximum	Heeft een getal dat fungeert als de maximumwaarde die geldig is voor deze invoereigenschap. Wordt alleen ondersteund voor de invoereigenschap van het type getal.	Nee

Tabel 5-6. Ondersteunde gegevenstypen en metagegevens voor custom scripts

Ondersteunde gegevenstypen	Ondersteunde metagegevens voor input
<ul style="list-style-type: none"> ■ Tekenreeks ■ Tekst ■ Lijst: als een lijst van een willekeurig type ■ Kaart: als map[string]any ■ Veilig: weergegeven als tekstvak voor wachtwoord, versleuteld wanneer u de custom taak opslaat ■ Getal ■ Boole: wordt weergegeven als tekstvakken ■ URL: hetzelfde als tekenreeks, met aanvullende validatie ■ Selectie, keuzerondje 	<ul style="list-style-type: none"> ■ type: Een van Tekenreeks Tekst ... ■ standaard: Standaardwaarde ■ opties: Lijst of een kaart met opties, die moeten worden gebruikt met selectie of keuzerondje ■ min: Minimum waarde of grootte ■ max.: Maximum waarde of grootte ■ titel: Gedetailleerde naam van het tekstvak ■ tijdelijke aanduiding: tijdelijke aanduiding voor UI ■ beschrijving: wordt een knopinfo

Bijvoorbeeld:

```
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 Declareer de outputproperties in uw script.

In het script worden outputproperties vastgelegd uit het `code:-`gedeelte met bedrijfslogica van uw script, waar u de context voor de output declareert.

Wanneer de pijplijn wordt uitgevoerd, kunt u de responscode voor de taakoutput invoeren. Bijvoorbeeld **200**.

Sleutels die door Code Stream worden ondersteund voor elke **outputProperty**.

sleutel	Beschrijving
type	Bevat momenteel een enkele waarde voor label1 .
name	Sleutel die het codeblok van de custom integratie-YAML afgeeft.
title	Label in de gebruikersinterface die outputProperty weergeeft.

Bijvoorbeeld:

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

- 4 Als u wilt communiceren met de input en output van uw custom script, haalt u een inputproperty op of stelt u een outputproperty in met behulp van **context**.

Voor een inputproperty: `(context.getInput("key"))`

Voor een outputproperty: `(context.setOutput("key", "value"))`

Voor Node.js:

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

Voor Python:

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

Voor Shell:

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

- 5 In het `code:-`gedeelte declareert u alle bedrijfslogica voor uw custom integratie.

Bijvoorbeeld, in de runtimeomgeving van 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)
        }
    };

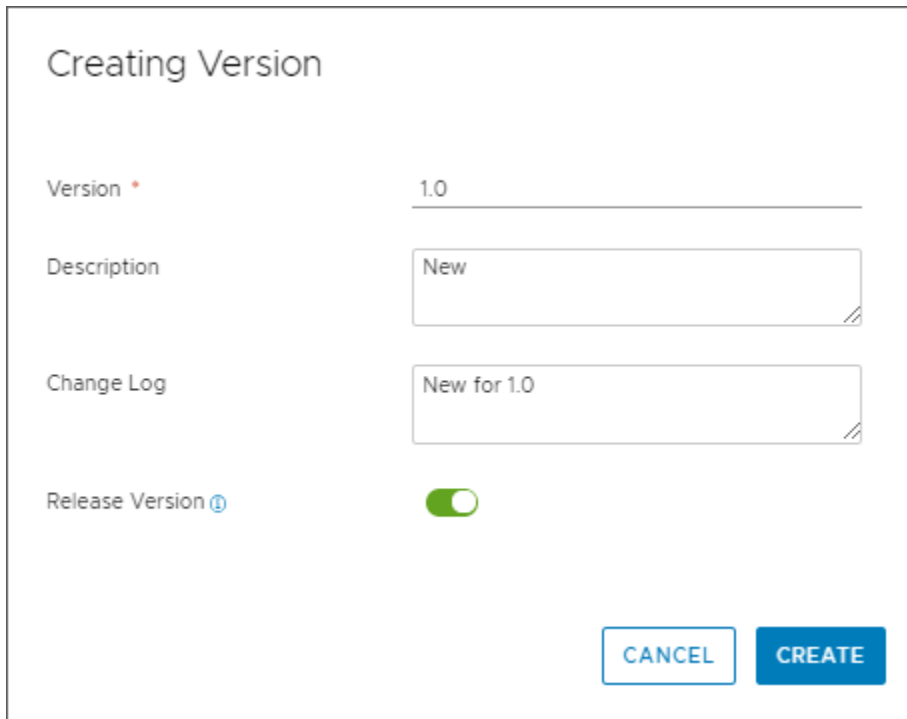
    // Makes a https request and sets the output with statusCode which
    // will be displayed in task result page after execution
    const req = https.request(options, (res) => {
        context.setOutput("statusCode", res.statusCode);
    });

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

```

- 6 Download het contextbestand voor Python of Node.js en test de bedrijfslogica die u in het script heeft opgenomen voordat u het custom integratiescript versioneert en vrijgeeft.
 - a Plaats de aanwijzer in het script en klik vervolgens op de knop voor het contextbestand bovenaan het canvas. U klikt bijvoorbeeld, als uw script in Python is, op **CONTEXT.PY**.
 - b Pas het bestand aan en sla het op.
 - c Voer in uw ontwikkelingssysteem uw custom script uit en test het met behulp van het contextbestand.
- 7 Pas een versie toe op uw custom integratiescript.
 - a Klik op **Versie**.
 - b Voer de versiegegevens in.

- c Klik op **Versie vrijgeven**, zodat u het script in uw custom taak kunt selecteren.
- d Als u de versie wilt maken, klikt u op **Maken**.



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

Creating Version

Version * 1.0

Description New

Change Log New for 1.0

Release Version ⓘ ☒

CANCEL CREATE

- 8 Klik op **Opslaan** om het script op te slaan.
- 9 Configureer de werkplek in uw pijplijn.
In dit voorbeeld wordt een Docker-werkplek gebruikt.

- a Klik op het tabblad **Werkruimte**.
- b Selecteer de Docker-host en de builderimage-URL.

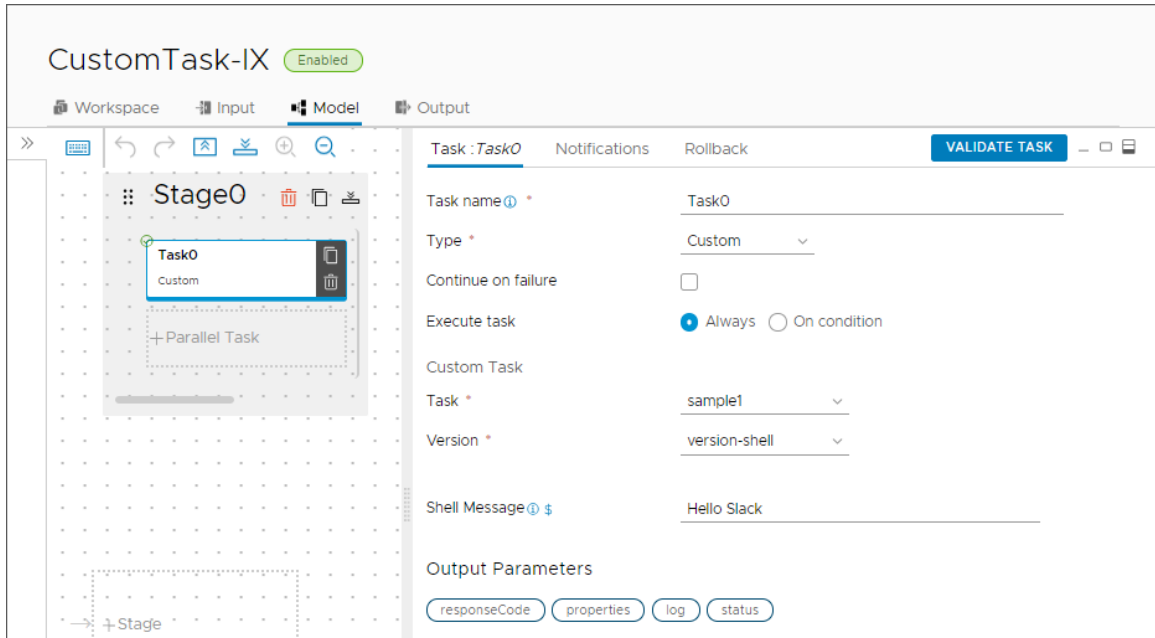
The screenshot shows the configuration interface for a custom task named "Demo-customTask-nodejs", which is currently "Enabled". The interface has four tabs: "Workspace" (selected), "Input", "Model", and "Output". Under the "Workspace" tab, there are several configuration fields:

- 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 input field.
- Cache**: An empty text input field with a blue plus icon to its right.
- Git clone**: A checkbox that is currently unchecked.

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

- 10 Voeg een custom taak toe aan uw pijplijn en configureer deze.
 - a Klik op het tabblad **Model**.
 - b Voeg een taak toe, selecteer het type als **Custom** en voer een relevante naam in.

- c Selecteer het custom integratiescript en de versie.
- d Als u een custom bericht in Slack wilt weergeven, voert u de tekst van het bericht in.
Alle tekst die u invoert, overschrijft de `defaultValue` in uw custom integratiescript.
Bijvoorbeeld:



- 11 Sla uw pijplijn op en schakel deze in.
 - a Klik op **Opslaan**.
 - b Op het tabblad Pijplijn klikt u op **Pijplijn inschakelen**, zodat de cirkel naar rechts wordt verplaatst.
- 12 Voer uw pijplijn uit.
 - a Klik op **Uitvoeren**.
 - b Bekijk de pijplijn-uitvoering.

- c Controleer of de output de verwachte statuscode, responscode, status en gedeclareerde output bevat.

U heeft **statusCode** gedefinieerd als een outputproperty. Bijvoorbeeld: een **statusCode** van 200 kan duiden op een succesvol Slack-bericht en een **responseCode** van 0 kan aangeven dat het script zonder fouten is geslaagd.

- d Om de output in de uitvoeringslogboeken te bevestigen, klikt u op **Uitvoeringen**, klikt u op de link naar uw pijplijn, klikt u op de taak en kijkt u naar de gegevens in het logboek. Bijvoorbeeld:

The screenshot displays the vRealize Automation interface for a pipeline named "custom-int-demo #5". The pipeline is in a "COMPLETED" state. Below the pipeline name, a summary bar shows "Stage0" with a green checkmark, and "Task0" and "Task1" also with green checkmarks. The main section details "Task1", which is of type "Custom" and has a status of "COMPLETED" with the message "Execution Completed.". The duration is listed as "6s (12/21/2018 3:04 AM - 12/21/2018 3:04 AM)". The "Continue on failure" checkbox is unchecked, and the "Execute task" setting is "Always". The "Output" section shows "statusCode" as 200 and "Response code" as 0. The "Logs" section displays a snippet of a command: "1 + node -r ./context.js app.js", "2", and "3 |". A "View Full Log" link is at the bottom.

13 Als er een fout optreedt, lost u het probleem op en voert u de pijplijn opnieuw uit.

Als een bestand of module in de basisimage bijvoorbeeld ontbreekt, moet u een andere basisimage maken die het ontbrekende bestand bevat. Geef vervolgens het Docker-bestand op en push de image via de pijplijn.

Resultaten

Gefeliciteerd! U heeft een custom integratiescript gemaakt waarmee Code Stream wordt verbonden met uw Slack-instantie en een bericht wordt geplaatst op een Slack-kanaal.

Wat nu te doen

Ga door met het maken van custom integraties om het gebruik van custom taken in uw pijplijnen te ondersteunen, zodat u de mogelijkheden van Code Stream bij de automatisering van uw software-release-levenscyclus kunt uitbreiden.

Hoe gebruik ik de resource-eigenschappen van een cloudsjabloontaak in mijn volgende taak

Wanneer u een cloudsjabloontaak in Code Stream gebruikt, is een veelvoorkomende vraag hoe u de uitvoer van die taak kunt gebruiken in een volgende taak in uw pijplijn. Als u de uitvoer van een cloudsjabloontaak zoals een cloudmachine wilt gebruiken, moet u weten hoe u de resource-eigenschappen in de implementatiegegevens van de cloudsjabloontaak en het IP-adres van de cloudmachine kunt vinden.

De gegevens van de implementatie van een VMware-cloudsjabloon bevatten bijvoorbeeld de cloudmachineresource en het IP-adres. In uw pijplijn kunt u de cloudmachine en het IP-adres gebruiken als variabele om een cloudsjabloontaak te binden aan een REST-taak.

De methode die u gebruikt om het IP-adres voor de cloudmachine te vinden is niet standaard, omdat de implementatie van de VMware-cloudsjabloon moet worden voltooid voordat de gegevens van de implementatie beschikbaar zijn. Vervolgens kunt u de resources van de implementatie van de VMware-cloudsjabloon gebruiken om uw pijplijntaken te binden.

- De resource-eigenschappen die worden weergegeven in een cloudsjabloontaak in uw pijplijn, worden gedefinieerd in de VMware-cloudsjabloon in Cloud Assembly.
- Mogelijk weet u niet wanneer een implementatie van die cloudsjabloon is voltooid.
- Een cloudsjabloontaak in Code Stream kan alleen de uitvoereigenschappen van de VMware-cloudsjabloon weergeven nadat de implementatie is voltooid.

Dit voorbeeld is vooral nuttig als u een applicatie implementeert en verschillende API's aanroept. Als u bijvoorbeeld een cloudsjabloontaak gebruikt die een VMware-cloudsjabloon aanroept, waarvoor een WordPress-applicatie met een REST API wordt geïmplementeerd, kunt u het IP-adres van de geïmplementeerde machine vinden in de implementatiegegevens en de API gebruiken om deze te testen.

De cloudsjabloontaak ondersteunt u om een variabele binding te gebruiken door de gegevens van het automatisch aanvullen voor snel typen weer te geven. U beslist hoe u de variabele bindt.

In dit voorbeeld ziet u hoe u kunt:

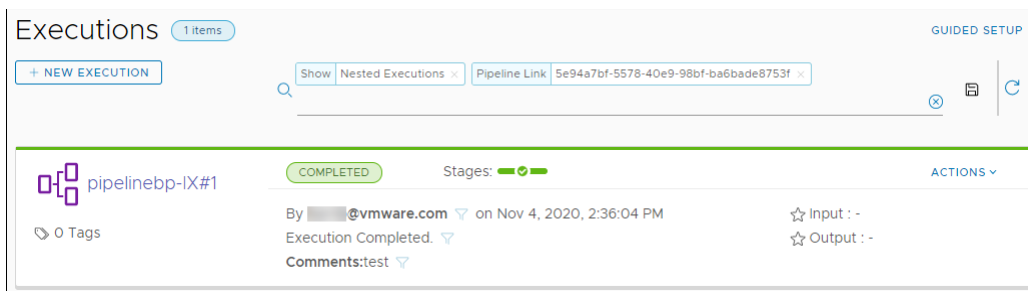
- Zoeken naar de implementatiegegevens en resource-eigenschappen voor uw cloudsjabloontaak in een pijplijn die is uitgevoerd en voltooid.
- Zoeken naar het IP-adres van de cloudmachine in de sectie Resources van de implementatiegegevens.
- Voeg een REST-taak toe die volgt op de cloudsjabloontaak in uw pijplijn.
- Bind de cloudsjabloontaak aan de REST-taak door gebruik te maken van het IP-adres van de cloudmachine in de URL van de REST-taak.
- Voer uw pijplijn uit en bekijk hoe de binding werkt van de cloudsjabloontaak tot de REST-taak.

Voorwaarden

- Controleer of u beschikt over een werkende VMware-cloudsjabloon met versienummer.
- Controleer of de implementatie van de VMware-cloudsjabloon is voltooid in Cloud Assembly.
- Controleer of u een pijplijn hebt die een cloudsjabloontaak bevat die de VMware-cloudsjabloon gebruikt.
- Controleer of uw pijplijn is uitgevoerd en is voltooid.

Procedure

- 1 Zoek in uw pijplijn het IP-adres van de cloudmachine in de sectie Resources van de implementatiegegevens van uw cloudsjabloontaak.
 - a Klik op **Acties > Uitvoeringen bekijken**.
 - b In een pijplijnuitvoering die is voltooid, klikt u op de link naar de pijplijnuitvoering.



- c Klik onder de naam van de pijplijn op de link naar de **taak**.

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

- d Zoek in de sectie Uitvoer naar de gegevens van de implementatie.

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-
4     cbbbc4b16c89",
5   "description": "Pipeline Service triggered operation",
6   "orgId": "434f6917-4e34-4537-b6c0-30f3630871bc",
7   "blueprintId": "8d1dd801-3a32-4f30-adde-27f8163dfe6f",
8   "blueprintVersion": "4",
9   "createdAt": "2020-11-04T21:36:14.500036Z",
10  "createdBy": "kernb@vmware.com",
11  "lastUpdatedAt": "2020-11-04T21:52:45.243028Z",
12  "lastUpdatedBy": "kernb@vmware.com",
13  "inputs": {},
14  "simulated": false,
15  "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
16  "resources": {
17    "Cloud_Machine_1[0]": {
18      "id": "/resources/compute/f5a846f3-c97c-4145-9e28-
19        951c36bd721c",
20      "name": "Cloud_Machine_1[0]",
21      "powerState": "ON"
22    }
23  }
24 }

```

Input

Action: Create Deployment

Cloud template: bhawesh

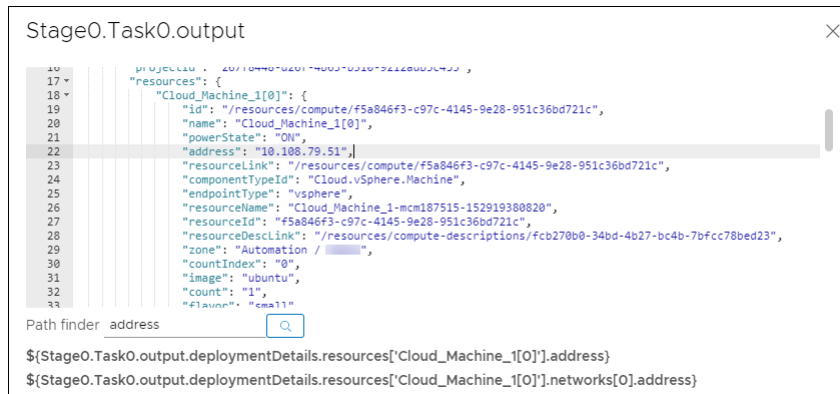
Cloud template version: 4

- e Zoek in de sectie Resources van de implementatiegegevens naar de naam van de cloudmachine.

U neemt de syntaxis voor de naam van de cloudmachine op in de URL van uw REST-taak.

- f Als u de bindingsexpressie voor de uitvoereigenschap van de cloudsjabloon taak wilt vinden, klikt u op **UITVOER-JSON WEERGEVEN**, zoekt u de adreseigenschap en zoekt u naar het IP-adres van de cloudmachine.

De bindingsexpressie wordt weergegeven onder de eigenschap en het zoekpictogram in de JSON-uitvoer.



De adresresource-eigenschap toont het IP-adres van de cloudmachine. Bijvoorbeeld:

```

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

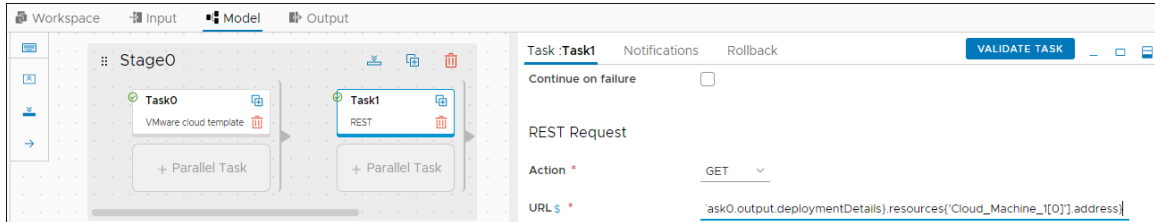
```

- 2 Ga terug naar uw pijplijnmodel en voer de URL in uw REST-taak in.
 - a Klik op **Acties > Pijplijn weergeven**.
 - b Klik op de REST-taak.

- c Voer **\$** in het gebied URL van REST-aanvraag in, selecteer de **fase, taak, uitvoer, implementatiegegevens** en voer **resources** in.

De mogelijkheid om snel te typen met automatisch aanvullen, is beschikbaar tot het punt dat u **resources** moet invoeren.

- d Voer de rest van de cloudmachineresource van de implementatiegegevens in als:
`{ 'Cloud_Machine_1[0]' }.address}`



Voor de cloudmachinevermelding moet u de notatie met vierkante haakjes gebruiken zoals hier wordt weergegeven.

De volledige URL-indeling is: \$

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

- 3 Voer uw pijplijn uit en bekijk hoe de REST-taak de cloudmachine en het IP-adres van de uitvoer van uw cloudsjabloontaak gebruikt als de URL die u wilt testen.

Resultaten

Gefeliciteerd! U hebt de naam en het IP-adres van de cloudmachine gevonden in de implementatiegegevens en de JSON-uitvoer van een cloudsjabloontaak, en u hebt ze gebruikt om de uitvoer van uw cloudsjabloontaak te binden aan de URL-invoer van uw REST-taak in uw pijplijn.

Wat nu te doen

Ga door met het verkennen van het gebruik van bindingsvariabelen van resources in de cloudsjabloontaak met andere taken in uw pijplijn.

Hoe gebruik ik een REST API om Code Stream te integreren met andere applicaties?

Code Stream biedt een REST-invoegtoepassing waarmee u Code Stream kunt integreren met andere applicaties die gebruikmaken van een REST API, zodat u voortdurend softwareapplicaties kunt ontwikkelen en leveren die met elkaar moeten communiceren. De REST-invoegtoepassing roept een API aan, die informatie verzendt en ontvangt tussen Code Stream en een andere applicatie.

Met de REST-invoegtoepassing kunt u volgende doen:

- Externe REST API-gebaseerde systemen in een Code Stream-pijplijn integreren.
- Een Code Stream-pijplijn als onderdeel van de stroom van externe systemen integreren.

De REST-invoegtoepassing werkt met elke REST API en ondersteunt GET-, POST-, PUT-, PATCH- en DELETE-methoden om informatie te verzenden of te ontvangen tussen Code Stream en andere applicaties.

Tabel 5-7. Een pijplijn voorbereiden om te communiceren via de REST API

Wat u doet	Wat gebeurt er als gevolg
Voeg een REST-taak toe aan uw pijplijn.	De REST-taak communiceert informatie tussen applicaties en kan statusinformatie geven voor een opeenvolgende taak in de pijplijnfase.
Selecteer in de REST-taak de REST-actie en neem de URL op.	De pijplijntaak roept de URL op wanneer de pijplijn wordt uitgevoerd. Voor POST-, PUT- en PATCH-acties moet u een lading toevoegen. In de lading kunt u de pijplijn en taakproperties binden wanneer de pijplijn wordt uitgevoerd.
Overweeg dit voorbeeld.	Voorbeeld van het gebruik van de REST-invoegtoepassing: U kunt een REST-taak toevoegen om een tag te maken op een Git-commit voor een build en de taak een aanvraag te laten indienen om de incheck-ID uit de opslagplaats te halen. De taak kan een lading naar uw opslagplaats sturen en een tag voor de build maken en de opslagplaats kan het antwoord met de tag retourneren.

Net als bij het gebruik van de REST-invoegtoepassing om een API aan te roepen, kunt u een polltaak in uw pijplijn opnemen om een REST API aan te roepen en deze te pollen totdat deze is voltooid en de pijplijntaak voldoet aan de afsluitcriteria.

U kunt ook REST API's gebruiken om een pijplijn te importeren en exporteren en de voorbeeldscripts gebruiken om een pijplijn uit te voeren.

Deze procedure haalt een eenvoudige URL op.

Procedure

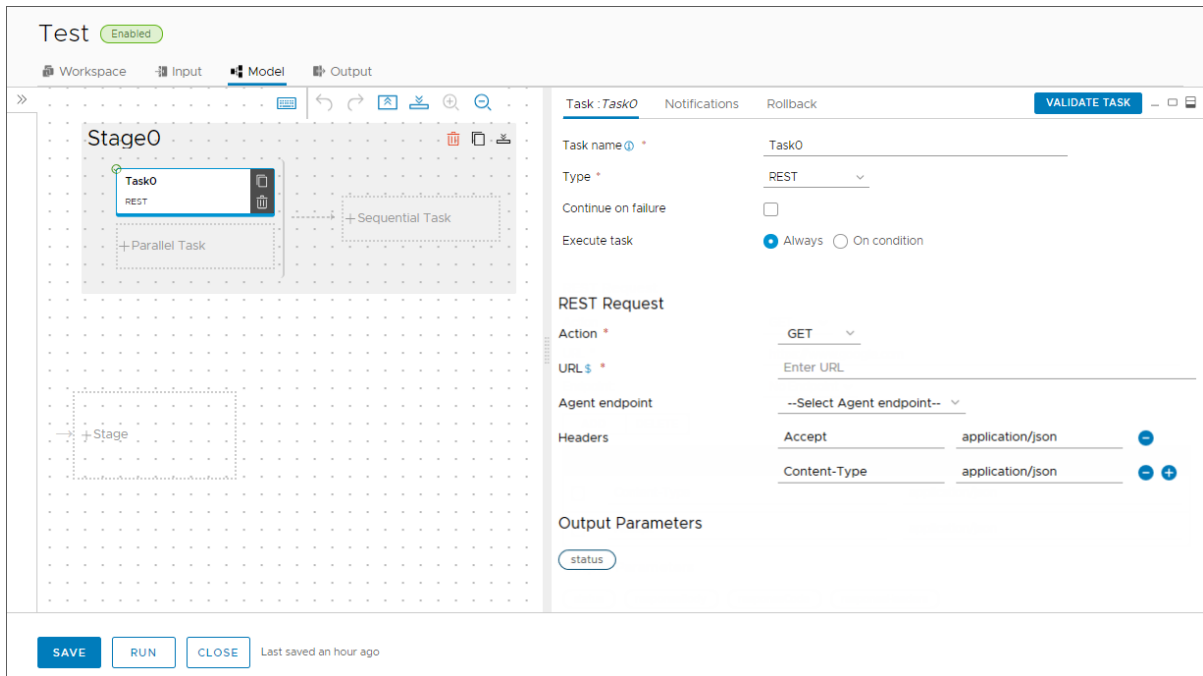
- 1 Als u een pijplijn wilt maken, klikt u op **Pijplijnen > Nieuwe pijplijn > Blanco canvas**.
- 2 Klik in uw pijplijnfase op **+ Sequentiële taak**.
- 3 Voeg in het taakdeelvenster de REST-taak toe:
 - a Voer een naam in voor de taak.
 - b Selecteer **REST** in het vervolgkeuzemenu Type.
 - c Selecteer **GET** in het REST-aanvraaggebied.

Als u wilt dat de REST-taak gegevens uit een andere applicatie opvraagt, selecteert u de GET-methode. Als u gegevens naar een andere applicatie wilt verzenden, selecteert u de POST-methode.

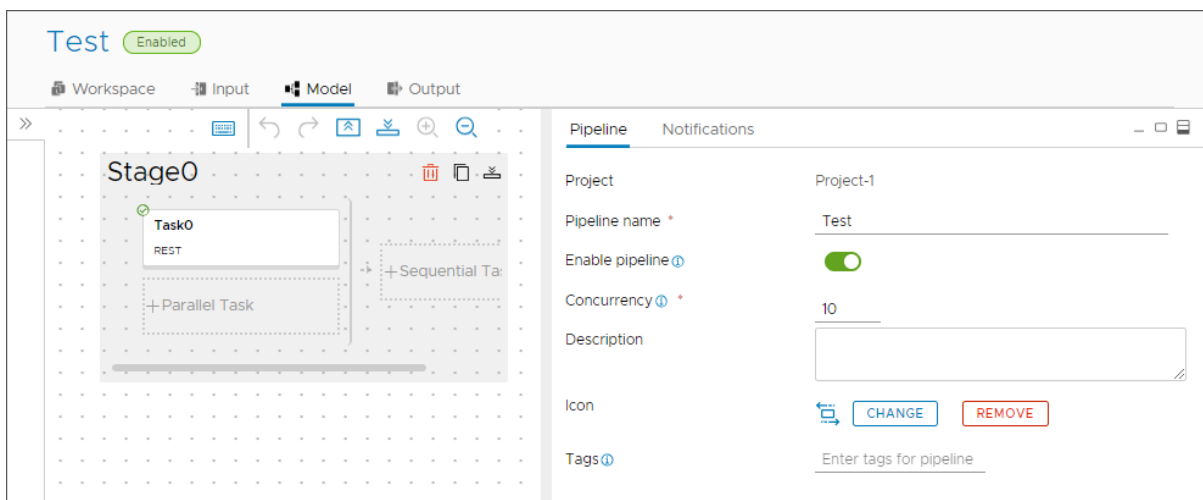
- d Voer de URL in voor de identificatie van het REST API-eindpunt. Bijvoorbeeld: `https://www.google.com`.

Voor een REST-taak om gegevens uit een andere applicatie te importeren, kunt u de ladingsvariabele opnemen. Voor een importactie kunt u bijvoorbeeld `{Stage0.export.responseBody}` invoeren. Als de responsgegevens groter dan 5 MB zijn, kan de REST-taak mislukken.

- e Als u autorisatie voor de taak wilt opgeven, klikt u op **Kopteksten toevoegen** voert u een koptekstsleutel en -waarde in.




- 4 Klik op **Opslaan** om uw pijplijn op te slaan.
- 5 Klik op het tabblad Pijplijn op **Pijplijn inschakelen**.



- 6 Klik op **Opslaan** en klik vervolgens op **Sluiten**.

- 7 Klik op **Uitvoeren**.
- 8 Klik op **Uitvoeren** om de uitvoering van de pijplijn te bekijken.

The screenshot displays the 'Executions' interface in vRealize Automation. At the top, the title 'Executions' is followed by a badge indicating '10 items'. Below this is a '+ NEW EXECUTION' button and a search bar. The main content area shows a single execution entry for 'Test#1'. The status is 'RUNNING', and a progress bar for 'Stages' is visible. The execution was initiated 'By system-user on 11/26/2018 3:11 PM'. The 'ACTIONS' menu is open, showing 'Input : n/a' and 'Output : n/a'.

Icon	Name	Status	Stages	Actions
	Test#1	RUNNING	Stages: <div><div></div></div>	<div>ACTIONS ▾ ☆ Input : n/a ☆ Output : n/a</div>

- 9 Om te controleren of de REST-taak de verwachte informatie retourneert, bekijkt u de pijplijnuitvoering en de taakresultaten.
 - a Nadat de pijplijn is voltooid, kunt u bevestigen dat de andere applicatie de door u aangevraagde gegevens heeft geretourneerd door op de link naar de pijplijn-uitvoering te klikken.
 - b Klik op de REST-taak in de pijplijn.
 - c In de pijplijnuitvoering klikt u op de taak, bekijkt u de taakgegevens en controleert u of de REST-taak de verwachte resultaten heeft geretourneerd.

In de taakgegevens worden de responscode, de tekst, de koptekstsleutels en de waarden weergegeven.

[< 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/google_standard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="aMlw/ydugkGr9CHU6QQGzg==">(function(){window.google={kEI:'cnf8W6KpJleVkuXx-aLoDA',kEXPI:'0,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,605,6,636,2239,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284,2,57,9,727,612,1820,58,2,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978,430,2487,850,525,2,2,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483,698,59,318,273,108,167,323,744,101,1119,38,363,557,4,38,135,145,155,497,2,718,383,978,487,47,1080,901,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,24,7,109,1049,14,758,7,127,179,9,21,261,1413,5977597,12,1861,681,134,43,5997424,90,2800095,4,1572,549,332,445,1,2,80,1,90,0,583,6,307,1,8,1,2,2132,1,1,1,1,1,414,1,748,141,297,169,301,24,2,8,96,50,2,47,22307501',authuser:0,kscs:'c9c918f0_cnf8W6KpJleVkuXx-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=a.getAttribute('!'))){a=a.p
```

Headers

Header Key	Header Value
X-Frame-Options	SAMEORIGIN
Transfer-Encoding	chunked
Cache-Control	private, max-age=0
Server	gws
All-Sub	quic="442" msa=3503000 us="44.42.30.25"

10 Als u de JSON-output wilt zien, klikt u op **JSON-OUTPUT WEERGEVEN**.

```

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
    =RTUkVjVhyg9KvAZR1S8yCCSEw8WosYf9mWdFQ1N5fNd5DvXUm5B3J8PyKMX1Z_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/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
    ,2,579,727,612,1820,58,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
    ,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
  }
  
```

Path finder Enter key

Resultaten

Gefeliciteerd! U heeft een REST-taak geconfigureerd die een REST API heeft aangeroepen en informatie heeft verzonden tussen Code Stream en een andere applicatie met behulp van de REST-invoegtoepassing.

Wat nu te doen

Blijf REST-taken in uw pijplijnen gebruiken om opdrachten uit te voeren en Code Stream te integreren met andere applicaties, zodat u uw softwareapplicaties kunt ontwikkelen en leveren. Overweeg het gebruik van pollingtaken om de API te pollen totdat deze is voltooid en de pijplijntaak voldoet aan de afsluitcriteria.

Hoe kan ik een pijplijn gebruiken als code in Code Stream?

Als DevOps-beheerder of -ontwikkelaar wilt u mogelijk een pijplijn in Code Stream maken met behulp van YAML-code in plaats van de gebruikersinterface te gebruiken. Wanneer u pijplijnen als code maakt, kunt u een willekeurige editor gebruiken en opmerkingen in de pijplijncode invoegen.

In uw pijplijncode kunt u verwijzen naar externe configuraties zoals omgevingsvariabelen en beveiligingsreferenties. Wanneer u variabelen bijwerkt die u in uw pijplijncode gebruikt, kunt u deze bijwerken zonder dat u de pijplijncode hoeft bij te werken.

U kunt de YAML-code voor de pijplijn gebruiken als sjabloon voor het klonen en maken van andere pijplijnen en het delen van de sjablonen met anderen.

U kunt sjablonen voor pijplijncodes opslaan in een opslagplaats voor bronbeheer, waarin versies worden gemaakt en updates worden bijgehouden. Met behulp van een bronbeheersysteem kunt u eenvoudig een back-up maken van uw pijplijncode en deze indien nodig herstellen.

Voorwaarden

- Controleer of u een code-editor heeft.
- Als u van plan bent uw pijplijncode op te slaan in een bronbeheeropslagplaats moet u controleren of u toegang heeft tot een werkende instantie.

Procedure

- 1 Maak een bestand in de code-editor.
- 2 Kopieer en plak de voorbeeldpijplijncode en werk deze bij zodat die overeenkomt met uw specifieke pijplijnbehoeften.
- 3 Om een eindpunt toe te voegen aan uw pijplijncode kopieert en plakt u de voorbeeldeindpuntcode en werkt u deze bij zodat die overeenkomt met uw eindpunt.

Wanneer u een Kubernetes API-eindpunt in de pijplijnwerkplek gebruikt, maakt Code Stream de nodige Kubernetes-resources zoals ConfigMap, Geheim en Pod om de taak voor continue integratie (CI) of de aangepaste taak uit te voeren. Code Stream communiceert met de container via de NodePort.

De Code Stream-pijplijnwerkplek ondersteunt Docker en Kubernetes voor taken voor continue integratie en aangepaste taken.

Zie [De pijplijnwerkplek configureren](#) voor meer informatie over het configureren van de werkplek.

- 4 Sla de code op.
- 5 Om uw pijplijncode op te slaan en te controleren, controleert u de code in uw opslagplaats voor bronbeheer.
- 6 Wanneer u een pijplijn voor continue integratie en levering maakt, moet u het Kubernetes YAML-bestand importeren.

Om het Kubernetes YAML-bestand te importeren, selecteert u het in het gedeelte Continue levering van de slimme pijplijnsjabloon en klikt u op **Proces**. Of gebruik de API.

Resultaten

Door de codevoorbeelden te gebruiken, heeft u de YAML-code gemaakt die uw pijplijn en eindpunten weergeeft.

Voorbeeld: Voorbeeld van een YAML-code voor een pijplijn en eindpunten

Dit voorbeeld van een YAML-code bevat onderdelen die de werkplek voorstellen voor de systeemeigen build van Code Stream, fasen, taken, meldingen en meer in een pijplijn.

Zie [Hoofdstuk 6 Code Stream verbinden met eindpunten](#) voor voorbeelden van code voor ondersteunde invoegtoepassingen

```
---
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}
---

```

Deze YAML-code geeft een voorbeeld van een Jenkins-eindpunt weer.

```

---
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
---

```

Deze YAML-code geeft een voorbeeld van een Kubernetes-eindpunt weer.

```

---
name: my-k8s
tags: [
]
kind: ENDPOINT
properties:
  kubernetesURL: https://urlname.examplelocation.amazonaws.com
  userName: admin
  password: encryptedpassword

```

```
description: ''  
type: kubernetes:KubernetesServer  
isLocked: false  
---
```

Wat nu te doen

Voer uw pijplijn uit en maak zo nodig aanpassingen. Zie [Hoe kan ik een pijplijn uitvoeren en de resultaten bekijken?](#).

Code Stream verbinden met eindpunten

6

Code Stream integreert met ontwikkelingstools via invoegtoepassingen. Ondersteunde invoegtoepassingen zijn Jenkins, Bamboo, vRealize Operations, Bugzilla, Team Foundation Server, Git, enz.

U kunt ook uw eigen invoegtoepassingen ontwikkelen om Code Stream te integreren met andere ontwikkelingsapplicaties.

Als u Code Stream met Jira wilt integreren, hebt u geen externe invoegtoepassing nodig, omdat Code Stream ook de mogelijkheid heeft om Jira-tickets te maken als meldingstype. U moet een Jira-eindpunt toevoegen als u Jira-tickets over de pijplijntatus wilt maken.

Dit hoofdstuk omvat de volgende onderwerpen:

- [Wat zijn eindpunten in Code Stream?](#)
- [Hoe integreer ik Code Stream met Jenkins?](#)
- [Hoe integreer ik Code Stream met Git?](#)
- [Hoe integreer ik Code Stream met Gerrit?](#)
- [Hoe integreer ik Code Stream met vRealize Orchestrator?](#)

Wat zijn eindpunten in Code Stream?

Een eindpunt is een instantie van een DevOps-applicatie die verbinding maakt met Code Stream en gegevens levert die uw pijplijnen kunnen uitvoeren, zoals een gegevensbron, opslagplaats of meldingssysteem.

Uw rol in Code Stream bepaalt hoe u eindpunten gebruikt.

- Beheerders en ontwikkelaars kunnen eindpunten maken, bijwerken, verwijderen en bekijken.
- Beheerders kunnen een eindpunt als beperkt markeren en pijplijnen uitvoeren die beperkte eindpunten gebruiken.
- Gebruikers met de lezersrol kunnen eindpunten zien, maar kunnen ze niet maken, bijwerken of verwijderen.

Zie [Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream?](#) voor meer informatie.

Als u Code Stream wilt verbinden met een eindpunt, voert u deze stappen uit.

- 1 Een taak toevoegen aan uw pijplijn
- 2 Configureer de taak zodat deze communiceert met het eindpunt.
- 3 Controleer of Code Stream verbinding kan maken met het eindpunt door op **Valideren** te klikken.
- 4 Wanneer u vervolgens de pijplijn uitvoert, maakt de taak verbinding met het eindpunt zodat dit de taak kan uitvoeren.

Zie [Welke typen taken zijn beschikbaar in Code Stream](#) voor informatie over de taaktypen die deze eindpunten gebruiken.

Tabel 6-1. Eindpunten die door Code Stream worden ondersteund

Eindpunt	Wat het biedt	Ondersteunde versies	Vereisten
Bamboo	Maakt bouwplannen.	6.9.*	
Docker	Systeemeigen builds kunnen Docker-hosts gebruiken voor implementatie.		Wanneer een pijplijn een image van Docker Hub bevat, moet u ervoor zorgen dat <code>cURL</code> of <code>wget</code> in de image is ingesloten voordat u de pijplijn uitvoert. Wanneer de pijplijn wordt uitgevoerd, downloadt Code Stream een binair bestand dat gebruikmaakt van <code>cURL</code> of <code>wget</code> om commando's uit te voeren.
Docker-register	Registreert containerimages zodat een Docker-buildhost images kan ophalen.	2.7.1	
Gerrit	Maakt verbinding met een Gerrit-server voor controles en triggers	2.14.*	
Git	Activeert pijplijnen wanneer ontwikkelaars code bijwerken en inchecken naar de opslagplaats.	Git Hub Enterprise 2.1.8 Git Lab Enterprise 11.9.12-ee	
Jenkins	Bouwt codeartefacten.	1.6.* en 2.*	
Jira	Maakt een JIRA-ticket wanneer een pijplijntaak mislukt.	8.3.*	

Tabel 6-1. Eindpunten die door Code Stream worden ondersteund (vervolg)

Eindpunt	Wat het biedt	Ondersteunde versies	Vereisten
Kubernetes	Automatiseert de stappen die gecontainerde applicaties implementeren, schalen en beheren.	Alle ondersteunde versies voor Cloud Assembly 8.4 en hoger 1.18 voor Cloud Assembly 8.3 en ouder	Wanneer u een Kubernetes API-eindpunt in de pijplijnwerkplek gebruikt, maakt Code Stream de nodige Kubernetes-resources zoals ConfigMap, Geheim en Pod om de taak voor continue integratie (CI) of de aangepaste taak uit te voeren. Code Stream communiceert met de container via de NodePort. Zie De pijplijnwerkplek configureren voor meer informatie over het configureren van de werkplek.
PowerShell	Maak taken die PowerShell-scripts uitvoeren op Windows- of Linux-machines.	4 en 5	
SSH	Maakt taken die SSH-scripts uitvoeren op Windows- of Linux-machines.	7.0	
TFS, Team Foundation Server	Beheert broncode, geautomatiseerde builds, tests en gerelateerde activiteiten.	2015 en 2017	
vRealize Orchestrator	Rangschikt en automatiseert de werkstromen in uw bouwproces.	7.* en 8.*	

Voor beeld van YAML-code voor een GitHub-eindpunt

Deze voorbeeld YAML-code definieert een GitHub-eindpunt waarnaar u kunt verwijzen in een Git-taak.

```
---
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
---
```

Hoe integreer ik Code Stream met Jenkins?

Code Stream biedt een Jenkins-invoegtoepassing die Jenkins-taken activeert om uw broncode te bouwen en testen. De Jenkins-invoegtoepassing voert testcases uit en kan custom scripts gebruiken.

Om een Jenkins-taak in uw pijplijn uit te voeren, gebruikt u een Jenkins-server en voegt u het Jenkins-eindpunt toe in Code Stream. Vervolgens maakt u een pijplijn en voegt u hieraan een Jenkins-taak toe.

Wanneer u de Jenkins-taak en een Jenkins-eindpunt in Code Stream gebruikt, kunt u een pijplijn maken die opdrachten met meerdere takken in Jenkins ondersteunt. De opdracht met meerdere takken omvat individuele opdrachten in elke tak van een Git-opslagplaats. Wanneer u pijplijnen in Code Stream maakt die opdrachten met meerdere takken ondersteunen:

- De Jenkins-taak kan Jenkins-opdrachten uitvoeren die zich in meerdere mappen op de Jenkins-server bevinden.
- U kunt het pad naar de map in de Jenkins-taakconfiguratie overschrijven zodat deze een ander mappad gebruikt, dat het standaardpad overschrijft dat op het Jenkins-eindpunt in Code Stream is gedefinieerd.
- Pijplijnen met meerdere takken in Code Stream detecteren Jenkins-opdrachtbestanden van het type `.groovy` in een Git-opslagplaats of een GitHub-opslagplaats en beginnen met het maken van opdrachten voor elke tak die in de opslagplaats wordt gescand.
- U kunt het standaardpad dat op het Jenkins-eindpunt is gedefinieerd, overschrijven met een pad dat is opgegeven in de Jenkins-taakconfiguratie en een opdracht en pijplijn uitvoeren die zijn gekoppeld aan elke tak in een Jenkins-hoofdopdracht.

Voorwaarden

- Stel een Jenkins-server in waarop versie 1.561 of hoger wordt uitgevoerd.
- Controleer of u lid bent van een project in Code Stream. Als u geen lid bent, vraagt u de Code Stream-beheerder om u als lid van een project toe te voegen. Zie [Hoe voeg ik een project toe in Code Stream?](#).
- Controleer of er een taak bestaat op de Jenkins-server, zodat de pijplijntaak deze kan uitvoeren.

Procedure

- 1 Een Jenkins-eindpunt toevoegen en valideren.
 - a Klik op **Eindpunten > Nieuw eindpunt**.
 - b Selecteer een project en selecteer **Jenkins** als eindpunttype. Voer vervolgens een naam en een beschrijving in.
 - c Als dit eindpunt een bedrijfskritisch onderdeel in uw infrastructuur is, moet u **Markeren als beperkt** inschakelen.

- d Voer de URL voor de Jenkins-server in.

- e Voer de gebruikersnaam en het wachtwoord in om u aan te melden op de Jenkins-server. Voer vervolgens de resterende informatie in.

Tabel 6-2. Resterende informatie voor het Jenkins-eindpunt

Eindpunt-invoer	Beschrijving
Pad naar de map	<p>Pad voor de map waarin uw taken zijn gegroepeerd. Jenkins kan alle taken in de map uitvoeren. U kunt submappen maken. Bijvoorbeeld:</p> <ul style="list-style-type: none"> ■ <code>folder_1</code> kan <code>job_1</code> bevatten ■ <code>folder_1</code> kan <code>folder_2</code> bevatten, die <code>job_2</code> kan bevatten <p>Wanneer u een eindpunt maakt voor <code>folder_1</code>, is het pad naar de map <code>job/folder_1</code> en vermeldt het eindpunt alleen <code>job_1</code>.</p> <p>Om de lijst met opdrachten in de onderliggende map <code>folder_2</code> te verkrijgen, moet u een ander eindpunt maken dat <code>/job/folder_1/job/folder_2/</code> als pad naar de map gebruikt.</p>
Mappad voor Jenkins-opdrachten met meerdere takken	<p>Om Jenkins-opdrachten met meerdere takken te ondersteunen, voert u in de Jenkins-taak het volledige pad in dat de Jenkins-server-URL en het volledige opdracht-pad bevat. Wanneer u een mappad in de Jenkins-taak opneemt, overschrijft dat pad het pad dat op het Jenkins-eindpunt wordt weergegeven. Met het aangepaste mappad in de Jenkins-taak, geeft Code Stream alleen opdrachten weer die in die map aanwezig zijn.</p> <ul style="list-style-type: none"> ■ Bijvoorbeeld: <code>https://server.yourcompany.com/job/project</code> ■ Als de pijplijn ook de belangrijkste Jenkins-opdracht moet activeren, gebruikt u: <code>https://server.yourcompany.com/job/project/job/main</code>
URL	<p>Host-URL van de Jenkins-server. Voer de URL in als <code>protocol://host:port</code>. Bijvoorbeeld: <code>http://192.10.121.13:8080</code></p>
Pollinginterval	<p>Intervalduur voor Code Stream om de Jenkins-server te pollen voor updates.</p>

Tabel 6-2. Resterende informatie voor het Jenkins-eindpunt (vervolg)

Eindpunt-invoer	Beschrijving
Aantal keer dat een aanvraag herhaald wordt	Aantal keren dat de geplande bouw aanvraag voor de Jenkins-server opnieuw moet worden geprobeerd.
Wachttijd voor nieuwe poging	Aantal seconden dat moet worden gewacht voordat de bouw aanvraag voor de Jenkins-server opnieuw wordt geprobeerd.

- f Klik op **Valideren** en controleer of het eindpunt verbinding maakt met Code Stream. Corrigeer eventuele fouten als er geen verbinding wordt gemaakt en klik vervolgens op **Opslaan**.

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 Om uw code te bouwen, maakt u een pijplijn en voegt u een taak toe die gebruikmaakt van uw Jenkins-eindpunt.
 - a Klik op **Pijplijnen > Nieuwe pijplijn > Leeg canvas**.
 - b Klik op de standaardfase.
 - c Voer in het Taak-gebied een naam voor de taak in.
 - d Selecteer **Jenkins** als taaktype.
 - e Selecteer het Jenkins-eindpunt dat u heeft gemaakt.
 - f Selecteer in het vervolgkeuzemenu een taak van de Jenkins-server die door uw pijplijn wordt uitgevoerd.
 - g Voer de parameters voor de taak in.
 - h Voer het verificatietoken voor de Jenkins-taak in.

The screenshot displays the vRealize Automation Code Stream interface for configuring a pipeline named 'Build and Deploy' (Status: Enabled).

Left Pane (Canvas):

- Stage0:** Contains two tasks: 'Build' and 'Test', both of type 'Jenkins'.
- Below Stage0:** A dashed box labeled '+ Parallel Task *' and another labeled '+ Stage'.

Right Pane (Task Configuration for 'Build'):

- Task name:** Build
- Type:** Jenkins
- Continue On Failure:** ☐
- Execute Task:** ☒ Always ☐ On Condition
- Jenkins:**
 - Endpoint:** aa
 - Job:** add_numbers
 - Num1:** 22
 - Num2:** 22
 - Token:** (empty field)
- Output Parameters:** status, job, jobId, jobResults, jobUrl

Bottom Bar:

- Buttons: SAVE, RUN, CLOSE
- Status: Last saved a month ago
- Help icon (bottom right)

- 3 Schakel uw pijplijn in en voer deze uit. Bekijk de uitvoering van de pijplijn.

[< BACK](#)

Build and Deploy #28 COMPLETED [ACTIONS](#)


Stage0

- Build
- Test
- Approval for Deployment
- Deployment
- Wait for application to start

Task name	Build VIEW OUTPUT JSON														
Type	Jenkins														
Status	COMPLETED Execution Completed.														
Duration	11s (08/06/2018 12:27 AM - 08/06/2018 12:27 AM)														
Continue On Failure	<input type="checkbox"/>														
Execute Task	<input checked="" type="radio"/> Always <input type="radio"/> On Condition														
Jenkins Job															
Endpoint	aa														
Job Name	add_numbers														
Job ID	1428														
Job URL	http://.../job/add_numbers/1428/														
Job Result	<table border="1"> <thead> <tr> <th>Key</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>junitResponse.failCount</td> <td>0</td> </tr> <tr> <td>junitResponse.skipCount</td> <td>0</td> </tr> <tr> <td>junitResponse.totalCount</td> <td>0</td> </tr> <tr> <td>junitResponse.successCount</td> <td>0</td> </tr> <tr> <td>jacocoResponse.lineCoverage</td> <td>0</td> </tr> <tr> <td>jacocoResponse.classCoverage</td> <td>0</td> </tr> </tbody> </table>	Key	Value	junitResponse.failCount	0	junitResponse.skipCount	0	junitResponse.totalCount	0	junitResponse.successCount	0	jacocoResponse.lineCoverage	0	jacocoResponse.classCoverage	0
Key	Value														
junitResponse.failCount	0														
junitResponse.skipCount	0														
junitResponse.totalCount	0														
junitResponse.successCount	0														
jacocoResponse.lineCoverage	0														
jacocoResponse.classCoverage	0														

- 4 Bekijk de details en status van de uitvoering op het pijplijndashboard.

U kunt eventuele fouten identificeren en zien waarom het is mislukt. U kunt ook trends zien over de uitvoeringsduur, voltooiingen en fouten van de pijplijn.


Build and Deploy
CLONE BACK

1D
7D
14D
↺

Recent Executions

Execution#/Stages	Stage0
#29	<div></div>
#28	<div></div>
#27	<div></div>
#26	<div></div>
#25	<div></div>
#24	<div></div>
#23	<div></div>
#22	<div></div>
#21	<div></div>
#20	<div></div>

● Completed
● Failed
● Running
● Waiting

Execution Details

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

🖨
▶
○
●
○
💬

Resultaten

Gefeliciteerd! U heeft Code Stream geïntegreerd met Jenkins door een eindpunt toe te voegen, een pijplijn te maken en een Jenkins-taak te configureren die uw code bouwt.

Voorbeeld: Voorbeeld van YAML voor een Jenkins-bouwtaak

Voor het type Jenkins-bouwtaak dat in dit voorbeeld wordt gebruikt, lijkt de YAML op de volgende code, waarbij meldingen zijn ingeschakeld:

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

Wat nu te doen

Bekijk de overige secties voor meer informatie. Zie [Hoofdstuk 6 Code Stream verbinden met eindpunten](#).

Hoe integreer ik Code Stream met Git?

Code Stream biedt een manier om een pijplijn te activeren als er een codewijziging optreedt in uw GitHub-, GitLab- of Bitbucket-opslagplaats. De Git-trigger gebruikt een Git-eindpunt op de tak van de opslagplaats die u wilt bewaken. Code Stream maakt verbinding met het Git-eindpunt via een webhook.

Om een Git-eindpunt in Code Stream te definiëren, selecteert u een project en voert u de tak in van de Git-opslagplaats waar het eindpunt zich bevindt. Het project groepeerde de pijplijn met het eindpunt en andere gerelateerde objecten. Wanneer u het project in uw webhook-definitie kiest, selecteert u het eindpunt en de pijplijn die u wilt activeren.

Opmerking Als u een webhook met uw eindpunt definieert en u het eindpunt later bewerkt, kunt u de gegevens van het eindpunt niet wijzigen in de webhook. Als u de eindpuntgegevens wilt wijzigen, moet u de webhook met het eindpunt verwijderen en opnieuw definiëren. Zie [Hoe gebruik ik de Git-trigger in Code Stream om een pijplijn uit te voeren?](#).

U kunt meerdere webhooks voor verschillende takken maken door hetzelfde Git-eindpunt te gebruiken en verschillende waarden voor de naam van de tak op te geven op de pagina voor webhookconfiguratie. Als u een andere webhook voor een andere tak in dezelfde Git-opslagplaats wilt maken, hoeft u het Git-eindpunt niet meerdere keren te klonen voor meerdere takken. In plaats daarvan geeft u de naam van de tak op in de webhook, zodat u het Git-eindpunt opnieuw kunt gebruiken. Als de tak op de Git-webhook hetzelfde is als de tak op het eindpunt, hoeft u geen taknaam op de pagina van de Git-webhook op te geven.

Voorwaarden

- Controleer of u toegang heeft tot de GitHub-, GitLab- of Bitbucket-opslagplaats waarmee u verbinding wilt maken.
- Controleer of u lid bent van een project in Code Stream. Als u geen beheerder bent, vraagt u de Code Stream-beheerder om u als lid toe te voegen aan een project. Zie [Hoe voeg ik een project toe in Code Stream?](#).

Procedure

- 1 Definieer een Git-eindpunt.
 - a Klik op **Eindpunten > Nieuw eindpunt**.
 - b Selecteer een project en selecteer **Git** als type eindpunt. Voer vervolgens een naam en een beschrijving in.

- c Als dit eindpunt een bedrijfskritisch onderdeel in uw infrastructuur is, moet u **Markeren als beperkt** inschakelen.

Wanneer u een beperkt eindpunt in een pijplijn gebruikt, kan een beheerder de pijplijn uitvoeren en moet de beheerder de pijplijnuitvoering goedkeuren. Als een eindpunt of variabele als beperkt is gemarkeerd, en een niet-administratieve gebruiker de pijplijn activeert, wordt de pijplijn onderbroken bij die taak en wordt gewacht totdat een beheerder deze heeft hervat.

Een projectbeheerder kan een pijplijn starten die beperkte eindpunten of variabelen bevat, als deze resources zich in het project bevinden waarvoor de gebruiker projectbeheerder is.

Wanneer een gebruiker die geen beheerder is een pijplijn probeert uit te voeren die een beperkte resource bevat, stopt de pijplijn bij de taak die de beperkte resource gebruikt. Vervolgens moet een beheerder de pijplijn hervatten.

Voor meer informatie over beperkte resources en aangepaste rollen met het recht **Beperkte pijplijnen beheren** zie:

- [Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream?](#)
- [Hoofdstuk 2 Code Stream instellen om mijn releaseproces te modelleren](#)

- d Selecteer een van de ondersteunde Git-servertypen.
- e Voer de URL voor de opslagplaats in met de API-gateway voor de server in het pad. Bijvoorbeeld:

Voer voor GitHub `https://api.github.com/vmware-example/repo-example` in.

Voer voor BitBucket het volgende in: `https://api.bitbucket.org/{user}/{repo name}` of `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`

- f Voer de tak in de opslagplaats in waar het eindpunt zich bevindt.
- g Selecteer het verificatietype en voer de gebruikersnaam in voor GitHub, GitLab of Bitbucket. Voer vervolgens het privétoken voor de gebruikersnaam in.
- Wachtwoord. Als u later een webhook wilt maken, moet u het privétoken voor het wachtwoord invoeren. Webhooks voor Git ondersteunen geen eindpunten die zijn gemaakt met basisverificatie.

Gebruik geheime variabelen om gevoelige informatie te verbergen en te versleutelen. Gebruik een beperkte variabele voor tekenreeksen, wachtwoorden en URL's die moeten worden verborgen en versleuteld, en om het gebruik in uitvoeringen te beperken. Gebruik bijvoorbeeld een geheime variabele voor een wachtwoord of URL. U kunt geheime en beperkte variabelen gebruiken in elk type taak in uw pijplijn.
 - Privétoken. Deze token is Git-specifiek en biedt toegang tot een specifieke actie. Zie https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html. U kunt ook een variabele voor het privétoken maken.

- 2 Klik op **Valideren** en controleer of het eindpunt verbinding maakt met Code Stream.

Corrigeer eventuele fouten als er geen verbinding wordt gemaakt en klik vervolgens op **Maken**.

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

Wat nu te doen

Zie de andere secties voor meer informatie. Zie [Hoe gebruik ik de Git-trigger in Code Stream om een pijplijn uit te voeren?](#).

Hoe integreer ik Code Stream met Gerrit?

Met Code Stream kunt u een pijplijn activeren als er een codecontrole plaatsvindt in uw Gerrit-project. De definitie van de Gerrit-trigger bevat het Gerrit-project en de pijplijnen die moeten worden uitgevoerd voor verschillende gebeurtenistypen.

De trigger voor Gerrit gebruikt een Gerrit-listener op de Gerrit-server die u wilt bewaken. Als u een Gerrit-eindpunt in Code Stream wilt definiëren, selecteert u een project en voert u de URL voor de Gerrit-server in. Vervolgens geeft u het eindpunt op wanneer u een Gerrit-luisteraar op die server maakt.

Als u een Gerrit-server gebruikt als Code Stream-eindpunt in een vRealize Automation-instantie waarvoor FIPS is ingeschakeld, moet u controleren of het configuratiebestand voor de Gerrit-server de juiste berichtverificatiesleutels bevat. Als het configuratiebestand voor de Gerrit-server niet de juiste berichtverificatiesleutels bevat, kan de server niet correct worden gestart en wordt het volgende bericht weergegeven: `PrivateKey/PassPhrase is incorrect`

Voorwaarden

- Controleer of u toegang heeft tot de Gerrit-server waarmee u verbinding wilt maken.
- Controleer of u lid bent van een project in Code Stream. Als u geen lid bent, vraagt u de Code Stream-beheerder om u als lid van een project toe te voegen. Zie [Hoe voeg ik een project toe in Code Stream?](#).

Procedure

1 Definieer een Gerrit-eindpunt.

- a Klik op **Configureren > Eindpunten** en klik op **Nieuw eindpunt**.
- b Selecteer een project en selecteer **Gerrit** als eindpunttype. Voer vervolgens een naam en een beschrijving in.
- c Als dit eindpunt een bedrijfskritisch onderdeel in uw infrastructuur is, moet u **Markeren als beperkt** inschakelen.
- d Voer de URL voor de Gerrit-server in.

Als u de standaardpoort wilt gebruiken, kunt u een poortnummer opgeven met de URL of de waarde leeg laten.

- e Voer een gebruikersnaam en wachtwoord in voor de Gerrit-server.

Als het wachtwoord moet worden versleuteld, klikt u op **Variabele maken** en selecteert u het type:

- **Geheim**. Het wachtwoord wordt omgezet wanneer een gebruiker met een rol de pijplijn uitvoert.
- **Beperkt**. Het wachtwoord wordt omgezet wanneer een gebruiker met een beheerdersrol de pijplijn uitvoert.

Voer als waarde het wachtwoord in dat beveiligd moet zijn, zoals het wachtwoord van een Jenkins-server.

- f Voer als persoonlijke sleutel de Secure Shell-sleutel in die wordt gebruikt om veilig toegang te krijgen tot de Gerrit-server.

Deze sleutel is de persoonlijke RSA-sleutel in de directory `.ssh`.

- g (Optioneel) Als een wachtwoordzin is gekoppeld aan de persoonlijke sleutel, voert u de wachtwoordzin in.

Als u de wachtwoordzin wilt versleutelen, klikt u op **Variabele maken** en selecteert u het type:

- Geheim. Het wachtwoord wordt omgezet wanneer een gebruiker met een rol de pijplijn uitvoert.
- Beperkt. Het wachtwoord wordt omgezet wanneer een gebruiker met een beheerdersrol de pijplijn uitvoert.

Voer als waarde de wachtwoordzin in die beveiligd moet zijn, zoals de wachtwoordzin voor een SSH-server.

- 2 Klik op **Valideren** en controleer of het Gerrit-eindpunt in Code Stream verbinding maakt met de Gerrit-server.

Corrigeer eventuele fouten als er geen verbinding wordt gemaakt en klik vervolgens op **Valideren**.

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>CREATE VARIABLE</div>
Private Key *	<div>-----BEGIN RSA PRIVATE KEY-----</div> <div>Proc-Type: 4,ENCRYPTED</div> <div>DEK-Info: AES-128-CBC,F00CE0B6526AF67DC77ADCD0962DBF92</div>
Pass Phrase ⓘ	<div>.....</div> <div>CREATE VARIABLE</div>

CREATE

VALIDATE

CANCEL

- 3 Klik op **Maken**.
- 4 Controleer of FIPS is ingeschakeld voor de vRealize Automation-omgeving of laat uw Jenkins-taak de omgeving maken waarbij FIPS is ingeschakeld met behulp van de Jenkins-URL.
 - a Als u de opdracht via de opdrachtregel wilt uitvoeren, maakt u via SSH verbinding met uw vRealize Automation 8.x-appliance en meldt u zich als rootgebruiker aan. Maak bijvoorbeeld verbinding met de URL van uw volledig gekwalificeerde domeinnaam, zoals `https://cava-1-234-567.yourcompanyFQDN.com` op poort 22, 5480 of 443.
 - b Voer de opdracht `vracli security fips` uit om te controleren op FIPS op vRealize Automation.
 - c Controleer of de opdracht `FIPS mode: strict` retourneert.

- 5 Als uw Gerrit-server een eindpunt is in een vRealize Automation-instantie waarvoor FIPS is ingeschakeld, moet u controleren of het configuratiebestand voor de Gerrit-server de juiste MAC-sleutels (Message Authentication Code) bevat.
 - a Open Gerrit en maak een SSH-sleutelpaar.
 - b Zoek het configuratiebestand van de Gerrit-server op '`$site_path`'/`etc/gerrit.config`.
 - c Controleer of het configuratiebestand van de Gerrit-server een of meer MAC-sleutels bevat, met uitzondering van `hmac-MD5`.

Opmerking In de FIPS-modus is `hmac-MD5` geen ondersteund MAC-algoritme. Om ervoor te zorgen dat de Gerrit-server correct wordt gestart, moet het configuratiebestand van de Gerrit-server dit algoritme uitsluiten. Als de Gerrit-server niet correct wordt gestart, wordt het volgende bericht weergegeven: `PrivateKey/PassPhrase is incorrect`

Ondersteunde MAC-sleutelnamen (Message Authentication Code) die beginnen met een plusteken (+) zijn ingeschakeld. De MAC-sleutelnamen die beginnen met een koppelteken (-), worden verwijderd uit de lijst met standaard-MAC's. Deze ondersteunde MAC's zijn standaard beschikbaar in Code Stream voor de Gerrit-server:

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

Wat nu te doen

Zie de andere secties voor meer informatie. Zie [Hoe gebruik ik de Gerrit-trigger in Code Stream om een pijplijn uit te voeren?](#).

Hoe integreer ik Code Stream met vRealize Orchestrator?

Code Stream kan integreren met vRealize Orchestrator (vRO) om de capaciteit ervan uit te breiden door vRO-werkstromen uit te voeren. vRealize Orchestrator bevat een groot aantal vooraf gedefinieerde werkstromen die kunnen integreren met tools van derden. Met deze werkstromen kunt u uw DevOps-processen automatiseren en beheren, bulkbewerkingen automatiseren en nog veel meer.

U kunt bijvoorbeeld een werkstroom in een vRO-taak in uw pijplijn gebruiken om een gebruiker in te schakelen, een gebruiker te verwijderen, VM's te verplaatsen, te integreren met testframeworks om uw code te testen terwijl de pijplijn wordt uitgevoerd en nog veel meer. U kunt bladeren in voorbeelden van code voor vRealize Orchestrator-werkstromen in code.vmware.com.

Met een vRealize Orchestrator-werkstroom kan uw pijplijn een actie uitvoeren tijdens het bouwen, testen en implementeren van uw applicatie. U kunt vooraf gedefinieerde werkstromen in uw pijplijn opnemen of u kunt custom werkstromen maken en gebruiken. Elke werkstroom bevat input, taken en output.

Om een vRO-werkstroom in uw pijplijn uit te voeren, moet de werkstroom worden weergegeven in de lijst met beschikbare werkstromen in de vRO-taak die u in uw pijplijn heeft opgenomen.

Voordat de werkstroom kan worden weergegeven in de vRO-taak in uw pijplijn, moet een beheerder de volgende stappen uitvoeren in vRealize Orchestrator:

- 1 Pas de tag CODESTREAM toe op de vRO-werkstroom.
- 2 Markeer de vRO-werkstroom als algemeen.

Voorwaarden

- Controleer of u toegang als beheerder heeft tot een vRealize Orchestrator-instantie op locatie. Raadpleeg uw eigen beheerder en de [vRealize Orchestrator-documentatie](#) voor hulp.
- Controleer of u lid bent van een project in Code Stream. Als u geen beheerder bent, vraagt u de Code Stream-beheerder om u als lid toe te voegen aan een project. Zie [Hoe voeg ik een project toe in Code Stream?](#).
- Maak in Code Stream een pijplijn en voeg een fase toe.

Procedure

- 1 Bereid als beheerder een vRealize Orchestrator-werkstroom voor om uw pijplijn te laten uitvoeren.
 - a Zoek in vRealize Orchestrator de werkstroom die u nodig heeft om in uw pijplijn te gebruiken, zoals een werkstroom om een gebruiker in te schakelen.
Als u een werkstroom nodig heeft die niet bestaat, kunt u deze maken.
 - b Voer in de zoekbalk **Label werkstroom** in om de werkstroom met de naam `Label werkstroom` te vinden.
 - c Klik op de kaart met de naam `Label werkstroom` op **Uitvoeren**, waarmee het configuratiegebied wordt weergegeven.
 - d Voer in het tekstgebied `Gelabelde werkstroom` de naam van de werkstroom in die u wilt gebruiken in uw Code Stream-pijplijn en selecteer deze in de lijst.
 - e In de `Label`- en `Waarde`-tekstgebieden voert u `CODESTREAM` met hoofdletters in.
 - f Klik op het selectievakje met de naam **Globale label**.

- g Klik op **Uitvoeren**, waarmee het label CODESTREAM wordt gekoppeld aan de werkstroom die u moet selecteren in uw Code Stream-pijplijn.
- h Klik in het navigatiedeelvenster op **Werkstromen** en controleer of het label met de naam CODESTREAM wordt weergegeven op de werkstroomkaart die uw pijplijn zal uitvoeren.

Nadat u bent aangemeld bij Code Stream en een vRO-taak aan uw pijplijn toevoegt, wordt de gelabelde werkstroom weergegeven in de werkstroomlijst.

2 Maak in Code Stream een eindpunt voor uw vRealize Orchestrator-instantie.

- a Klik op **Eindpunten > Nieuw eindpunt**.
- b Selecteer een project.
- c Voer een relevante naam in.
- d Voer de URL van het vRealize Orchestrator-eindpunt in.

Gebruik de indeling: **https://vro-appliance.yourdomain.local:8281**

Gebruik niet de volgende indeling: https://vro-appliance.yourdomain.local:8281/vco/api

De URL voor een vRealize Orchestrator-instantie die wordt ingesloten in de vRealize Automation-appliance, is de FQDN voor de appliance zonder poort. Bijvoorbeeld:

https://vra-appliance.yourdomain.local/vco

Voor externe vRealize Orchestrator-appliances die wordt gestart met vRealize Automation 8.x, is **https://vro-appliance.yourdomain.local** de FQDN voor de appliance.

Voor externe vRealize Orchestrator-appliances die deel uitmaken van vRealize Automation 7.x, is **https://vro-appliance.yourdomain.local:8281/vco** de FQDN voor de appliance

Als er een probleem optreedt wanneer u het eindpunt toevoegt, moet u mogelijk een YAML-configuratie importeren met een SHA-256-certificaatvingerafdruk waarbij de dubbelepunten zijn verwijderd. Bijvoorbeeld: **B0:01:A2:72...** wordt **B001A272...** Het voorbeeld van de YAML-code ziet er zo uit:

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

- e Klik op **Certificaat accepteren** als de URL die u heeft ingevoerd een certificaat nodig heeft.
- f Voer de gebruikersnaam en het wachtwoord voor de vRealize Orchestrator-server in.

Als u een niet-lokale gebruiker voor verificatie gebruikt, moet u het domeingedeelte van de gebruikersnaam weglaten. Als u bijvoorbeeld wilt verifiëren met **svc_vro@yourdomain.local**, moet u **svc_vro** invoeren in het tekstgebied **Username**.

3 Bereid uw pijplijn voor om de vRO-taak uit te voeren.

- a Voeg een vRO-taak toe aan uw pijplijnfase.
- b Voer een relevante naam in.
- c Selecteer in het Werkstroomproperties-gebied het vRealize Orchestrator-eindpunt.
- d Selecteer de werkstroom die u heeft gelabeld als CODESTREAM in vRealize Orchestrator.

Als u een custom werkstroom selecteert die u heeft gemaakt, moet u mogelijk de waarden voor de inputparameters invoeren.

- e Klik op **Op voorwaarde** voor **Taak uitvoeren**.

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 corner. The configuration fields are as follows:

- Task name**: vRO workflow
- Type**: vRO (dropdown)
- Duration**: NaNms (-)
- Continue on failure**: ☐
- Execute task**: ☐ Always ☒ On condition
- Condition**: An empty text box with an information icon (i) on the right.
- Workflow Properties**:
 - Endpoint**: vROEP (dropdown)
 - Workflow**: Test (dropdown)
 - Greeting**: Hello! (text field)
- Output Parameters**: Two buttons labeled "status" and "properties".

- f Voer de voorwaarden in die van toepassing zijn wanneer de pijplijn wordt uitgevoerd.

Wanneer de pijplijn moet worden uitgevoerd...	Selecteer voorwaarden...
Op voorwaarde	<p>De pijplijntaak wordt alleen uitgevoerd als de gedefinieerde voorwaarde als waar wordt geëvalueerd. Als de voorwaarde onwaar is, wordt de taak overgeslagen.</p> <p>Met de vRO-taak kunt u een Boole-expressie opnemen, die de volgende operanden en operatoren gebruikt.</p> <ul style="list-style-type: none"> ■ Pijplijnvariabelen zoals <code>\${pipeline.variableName}</code>. Gebruik alleen gekrulde haakjes wanneer u variabelen invoert. ■ Outputvariabelen voor taken zoals <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code>. ■ Standaard bindingsvariabelen voor pijplijnen zoals <code>\${releasePipelineName}</code>. ■ Niet-hoofdlettergevoelige Boole-waarden zoals <code>true</code>, <code>false</code>, <code>'true'</code>, <code>'false'</code>. ■ Gehele of decimale waarden zonder aanhalingstekens. ■ Tekenreekswaarden die worden gebruikt met enkele of dubbele aanhalingstekens, zoals <code>"test"</code>, <code>'test'</code>. ■ Tekenreeks- en numerieke waarden zoals <code>==</code> <code>Equals</code> en <code>!=</code> <code>Not Equals</code>. ■ Relationele operatoren zoals <code>></code>, <code>>=</code>, <code><</code> en <code><=</code>. ■ Boole-logica zoals <code>&&</code> en <code> </code>. ■ Rekenkundige operatoren zoals <code>+</code>, <code>-</code>, <code>*</code> en <code>/</code>. ■ Geneste expressies met ronde haakjes. ■ Tekenreeksen die de letterlijke waarde <code>ABCD</code> bevatten, worden als onwaar geëvalueerd en de taak wordt overgeslagen. ■ Unaire operatoren worden niet ondersteund. <p>Een voorbeeld van een voorwaarde is <code>\${Stage1.task1.output} == "Passed" \${pipeline.variableName} == 39</code></p>
Altijd	Als u Altijd selecteert, voert de pijplijn de taak uit zonder voorwaarden.

- g Voer een bericht in voor de begroeting.
- h Klik op **Taak valideren** en corrigeer eventuele fouten.
- 4 Sla uw pijplijn op, schakel deze in en voer deze uit.
- 5 Bekijk de resultaten nadat de pijplijn is uitgevoerd.
- a Klik op **Uitvoeringen**.
- b Klik op de pijplijn.

- c Klik op de taak.
- d Bekijk de resultaten, de inputwaarde en de properties.

U kunt de uitvoerings-ID van de werkstroom, de persoon die heeft gereageerd op de taak, wanneer de persoon heeft gereageerd en alle opmerkingen die deze persoon heeft opgenomen, identificeren.

Resultaten

Gefeliciteerd! U heeft een vRealize Orchestrator-werkstroom gelabeld voor gebruik in Code Stream en u heeft een vRO-taak toegevoegd aan uw Code Stream-pijplijn zodat deze een werkstroom uitvoert die een actie in uw DevOps-omgeving automatiseert.

Voorbeeld: Outputindeling voor vRO-taak

De outputindeling voor een vRO-taak lijkt op dit voorbeeld.

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

Wat nu te doen

Ga door met het toevoegen van vRO-werkstroomtaken in uw pijplijnen zodat u taken kunt automatiseren in uw ontwikkelings-, test- en productieomgevingen.

Pijplijnen in Code Stream activeren

7

U kunt Code Stream een pijplijn laten activeren wanneer bepaalde gebeurtenissen optreden.

Bijvoorbeeld:

- De Docker-trigger kan een pijplijn uitvoeren wanneer een nieuw artefact wordt gemaakt of bijgewerkt.
- De Git-trigger kan een pijplijn activeren wanneer ontwikkelaars code bijwerken.
- De Gerrit-trigger kan een pijplijn activeren wanneer ontwikkelaars code controleren.

Dit hoofdstuk omvat de volgende onderwerpen:

- [Hoe kan ik de Docker-trigger gebruiken in Code Stream om een continue leveringspijplijn uit te voeren?](#)
- [Hoe gebruik ik de Git-trigger in Code Stream om een pijplijn uit te voeren?](#)
- [Hoe gebruik ik de Gerrit-trigger in Code Stream om een pijplijn uit te voeren?](#)

Hoe kan ik de Docker-trigger gebruiken in Code Stream om een continue leveringspijplijn uit te voeren?

Als Code Stream-beheerder of -ontwikkelaar kunt u de Docker-trigger in Code Stream gebruiken. Gebruik de Docker-trigger om een standalone pijplijn voor continue levering (CD) te activeren wanneer een build-artefact wordt gemaakt of bijgewerkt. De Docker-trigger voert de CD-pijplijn uit die het nieuwe of bijgewerkte artefact pusht als een containerimage naar een Docker Hub-opslagplaats. De CD-pijplijn kan worden uitgevoerd als onderdeel van uw geautomatiseerde builds.

Als u bijvoorbeeld uw bijgewerkte containerimage via uw CD-pijplijn continu wilt implementeren, gebruikt u de Docker-trigger. Wanneer uw containerimage wordt ingecheckt in het Docker-register krijgt Code Stream een melding van de webhook in Docker Hub dat de image is gewijzigd. Deze melding activeert de CD-pijplijn die wordt uitgevoerd met de bijgewerkte containerimage en uploadt de image naar de Docker Hub-opslagplaats.

Als u de Docker-trigger wilt gebruiken, voert u meerdere stappen in Code Stream uit.

Tabel 7-1. De Docker-trigger gebruiken

Wat u doet...	Meer informatie over deze actie...
Maak een Docker-registereindpunt.	Om Code Stream uw pijplijn te laten activeren, heeft u een Docker-registereindpunt nodig. Als het eindpunt niet bestaat, kunt u een optie selecteren die het maakt wanneer u de webhook voor de Docker-trigger toevoegt. Het Docker-registereindpunt bevat de URL naar de Docker Hub-opslagplaats.
Voeg inputparameters toe aan de pijplijn die automatisch Docker-parameters injecteren wanneer de pijplijn wordt uitgevoerd.	U kunt Docker-parameters injecteren in de pijplijn. Parameters kunnen de naam van de eigenaar van de Docker-gebeurtenis, de image, de opslagplaatsnaam, de naamruimte van de opslagplaats en het label bevatten. In de CD-pijplijn neemt u inputparameters op die de Docker-webhook aan de pijplijn doorgeeft voordat de pijplijn wordt geactiveerd.
Maak een Docker-webhook.	<p>Wanneer u de Docker-webhook maakt in Code Stream, wordt ook een overeenkomstige webhook in Docker Hub gemaakt. De Docker-webhook in Code Stream maakt verbinding met de Docker Hub via de URL die u opneemt in de webhook.</p> <p>De webhooks communiceren met elkaar en activeren de pijplijn wanneer een artefact wordt gemaakt of bijgewerkt in de Docker Hub.</p> <p>Als u de Docker-webhook in Code Stream bijwerkt of verwijdert, wordt de webhook in Docker Hub ook bijgewerkt of verwijderd.</p>
Een Kubernetes-taak toevoegen en configureren in uw pijplijn.	Wanneer een artefact wordt gemaakt of bijgewerkt in de Docker Hub-opslagplaats wordt de pijplijn geactiveerd. Vervolgens wordt het artefact via de pijplijn geïmplementeerd naar de Docker-host in uw Kubernetes-cluster.
Neem een lokale YAML-definitie op in de-taak.	De YAML-definitie die u toepast op de implementatietask omvat de Docker-containerimage. Als u een image moet downloaden uit een privéopslagplaats, moet het YAML-bestand een sectie bevatten met het Docker-configuratiegeheim. Zie het CD-gedeelte van Een systeemeigen CICD-build plannen in Code Stream voordat u de slimme pijlijnsjabloon gebruikt

Wanneer een artefact wordt gemaakt of bijgewerkt in de Docker Hub-opslagplaats, geeft de webhook in de Docker Hub dit door aan de webhook in Code Stream, waardoor de pijplijn wordt geactiveerd. De volgende acties worden uitgevoerd:

- 1 Docker Hub verzendt een POST-aanvraag naar de URL in de webhook.
- 2 Code Stream voert de Docker-trigger uit.
- 3 De Docker-trigger start uw CD-pijplijn.
- 4 De CD-pijplijn pusht het artefact naar de Docker Hub-opslagplaats.

- 5 Code Stream activeert de Docker-webhook die een CD-pijplijn uitvoert die het artefact implementeert op uw Docker-host.

In dit voorbeeld maakt u een Docker-eindpunt en een Docker-webhook in Code Stream die uw applicatie implementeren op uw Kubernetes-ontwikkelingscluster. De stappen omvatten de voorbeeldcode voor de lading die de Docker naar de URL van de webhook plaatst, de gebruikte API-code en de verificatiecode met het beveiligde token.

Voorwaarden

- Controleer of er een pijplijn voor continue levering (CD) bestaat in uw Code Stream-instantie. Controleer ook of het een of meer Kubernetes-taken bevat die uw applicatie implementeren. Zie [Hoofdstuk 4 Plannen om op een systeemeigen manier uw code te bouwen, integreren en leveren in Code Stream](#).
- Controleer of u toegang heeft tot een bestaand Kubernetes-cluster waarop uw CD-pijplijn uw applicatie kan implementeren voor ontwikkeling.
- Controleer of u lid bent van een project in Code Stream. Als u geen beheerder bent, vraagt u de Code Stream-beheerder om u als lid toe te voegen aan een project. Zie [Hoe voeg ik een project toe in Code Stream?](#).

Procedure

- 1 Maak een Docker-registereindpunt.
 - a Klik op **Eindpunten**.
 - b Klik op **Nieuw eindpunt**.
 - c Begin met het typen van de naam van een bestaand project.
 - d Selecteer **Docker-register** als type.
 - e Voer een relevante naam in.
 - f Selecteer **Docker Hub** als servertype.
 - g Voer de URL in naar de Docker Hub-opslagplaats.
 - h Voer de naam en het wachtwoord in die toegang geven tot de opslagplaats.

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 Stel in de CD-pijplijn de inputproperties in om Docker-parameters automatisch te injecteren wanneer de pijplijn wordt uitgevoerd.

sm-1 Enabled

Workspace **Input** Model Output

Input Parameters ⓘ

Auto inject parameters ☐ Gerrit ☐ Git ☒ Docker ☐ None

ADD ADD/REMOVE INJECTED PARAMETERS

Starred ⓘ	Name
⋮ ☆	DOCKER_EVENT_OWNER_NAME
⋮ ☆	DOCKER_IMAGE
⋮ ☆	DOCKER_REPO_NAME
⋮ ☆	DOCKER_REPO_NAMESPACE
⋮ ☆	DOCKER_TAG

- 3 Maak een Docker-webhook.
 - a Klik op **Triggers > Docker**.
 - b Klik op **Nieuwe webhook voor Docker**.
 - c Selecteer een project.
 - d Voer een relevante naam in.
 - e Selecteer het Docker-registereindpunt.

Als het eindpunt nog niet bestaat, klikt u op **Eindpunt maken** en maakt u het.

- f Selecteer de pijplijn met Docker-geïnjekteerde parameters voor de webhook die moet worden geactiveerd. Zie [Stap 2](#).

Als de pijplijn is geconfigureerd met aangepaste toegevoegde invoerparameters, worden parameters en waarden weergegeven in de lijst Invoerparameters. U kunt waarden invoeren voor inputparameters die worden doorgegeven naar de pijplijn met de triggergebeurtenis. Of u kunt de waarden leeg laten of de standaardwaarden gebruiken, indien gedefinieerd.

Zie [Een CI/CD-systeemeigen build plannen in Code Stream voordat u handmatig taken toevoegt](#) voor meer informatie over de parameters op het tabblad Invoer.

- g Voer de API-token in.

Het CSP API-token verifieert u voor externe API-verbindingen met Code Stream. Om het API-token te verkrijgen:

- 1 Klik op **Token genereren**.
- 2 Voer het e-mailadres in dat is gekoppeld aan uw gebruikersnaam en wachtwoord en klik op **Genereren**.

Het token dat u genereert, is zes maanden geldig. Het wordt ook wel een vernieuwingstoken genoemd.

- Als u het token als een variabele wilt behouden voor toekomstig gebruik klikt u op **Variabele maken**, voert u een naam voor de variabele in en klikt u op **Opslaan**.
- Als u het token als tekstwaarde wilt behouden voor toekomstig gebruik klikt u op **Kopiëren** en plakt u het token in een tekstbestand om lokaal op te slaan.

U kunt ervoor kiezen om beide een variabele te maken en het token in een tekstbestand op te slaan voor toekomstig gebruik.

- 3 Klik op **Sluiten**.

- h Voer de buildimage in.

- i Voer een tag in.

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 * `[redacted]` [?] **CREATE VARIABLE** **GENERATE TOKEN**

Image [?] `Image`

Tag [?] `Tags`

SAVE **CANCEL**

- j Klik op **Opslaan**.

De webhookkaart wordt weergegeven wanneer de Docker-webhook is ingeschakeld. Als u een dummy push wilt uitvoeren naar de Docker Hub-opslagplaats zonder de Docker-webhook te activeren en een pijplijn uit te voeren, klikt u op **Uitschakelen**.

- 4 Configureer uw Kubernetes-implementatietask in uw CD-pijplijn.
 - a Selecteer in de Kubernetes-taakproperties uw Kubernetes-ontwikkelingscluster.
 - b Selecteer de actie **Maken**.

- c Selecteer de **Lokale definitie** voor de ladingsbron.
- d Selecteer vervolgens uw lokale YAML-bestand.

Docker Hub kan bijvoorbeeld deze lokale YAML-definitie als de lading naar de URL in de webhook plaatsen:

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

De API die de webhook in Docker Hub maakt, gebruikt deze

vorm: https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/

De hoofdtekst van de JSON-code ziet er ongeveer zo uit:

```
{
  "name": "demo_webhook",
```

```
"webhooks": [
{
"name": "demo_webhook",
"hook_url": "http://www.google.com"
}
]
}
```

Als u gebeurtenissen van de Docker Hub-server wilt ontvangen, gebruikt het verificatieschema voor de Docker-webhook die u maakt in Code Stream een allowlist-verificatiemechanisme met een willekeurige tekenreeks als token voor de webhook. Hiermee worden gebeurtenissen gefilterd op basis van de beveiligde token, die u kunt toevoegen aan `hook_url`.

Code Stream kan elke aanvraag van de Docker Hub-server verifiëren met behulp van de geconfigureerde beveiligde token. Bijvoorbeeld: `hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""`

- 5 Maak een Docker-artefact in uw Docker Hub-opslagplaats. U kunt ook een bestaand artefact bijwerken.
- 6 Als u wilt controleren of de trigger heeft plaatsgevonden en de activiteit op de Docker-webhook wilt zien, klikt u op **Triggers > Docker > Activiteit**.

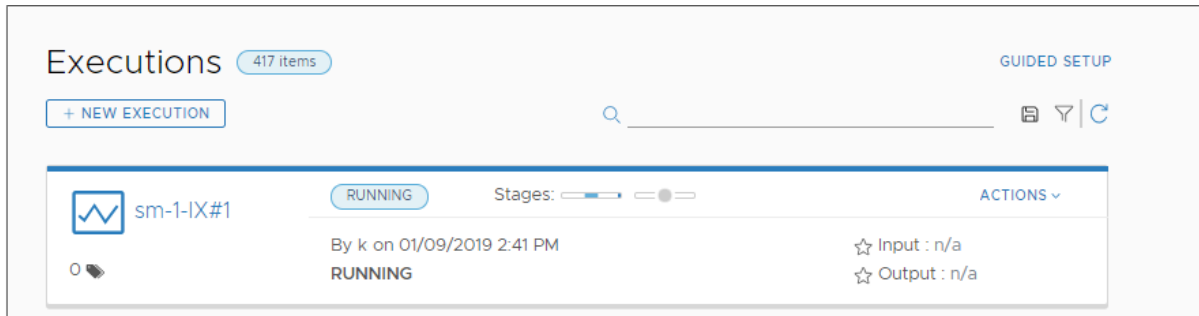
Docker

GUIDED SETUP

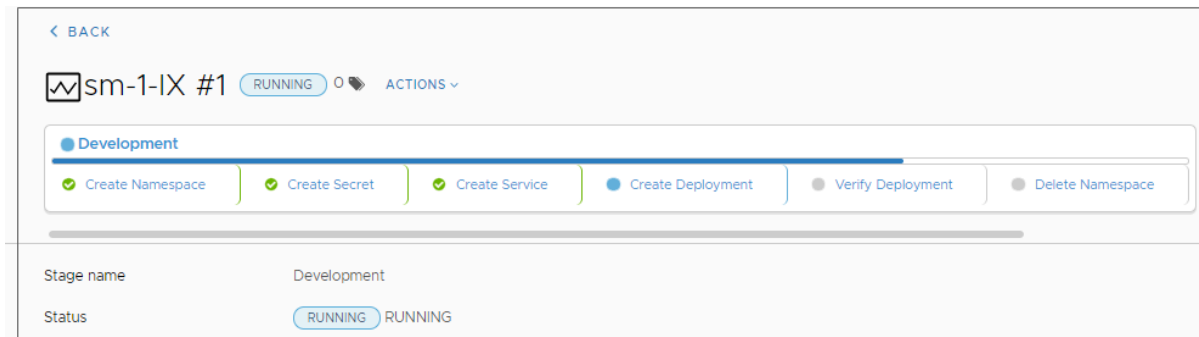
Activity
Webhooks for Docker

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 Klik op **Uitvoeringen** en bekijk uw pijplijn tijdens de uitvoering.



- 8 Klik op de actieve fase en bekijk de taken terwijl de pijplijn wordt uitgevoerd.



Resultaten

Gefeliciteerd! U stelt de Docker-trigger in om uw CD-pijplijn continu uit te voeren. Uw pijplijn kan nu nieuwe en bijgewerkte Docker-artefacten uploaden naar de Docker Hub-opslagplaats.

Wat nu te doen

Controleer of uw nieuwe of bijgewerkte artefact is geïmplementeerd naar de Docker-host in uw Kubernetes-ontwikkelingscluster.

Hoe gebruik ik de Git-trigger in Code Stream om een pijplijn uit te voeren?

Als Code Stream-beheerder of -ontwikkelaar kunt u Code Stream met de Git-levenscyclus integreren met behulp van de Git-trigger. Wanneer u code wijzigt in GitHub, GitLab of Bitbucket Enterprise, communiceert de gebeurtenis met Code Stream via een webhook en wordt een pijplijn geactiveerd. De webhook werkt met GitLab-, GitHub- en Bitbucket-bedrijfsversies op locatie wanneer zowel Cloud Assembly als de bedrijfsversie bereikbaar zijn op hetzelfde netwerk.

Wanneer u de webhook voor Git toevoegt in Code Stream, wordt ook een webhook in de GitHub-, GitLab- of Bitbucket-opslagplaats gemaakt. Als u de webhook later bijwerkt of verwijdert, wordt de webhook ook door die actie bijgewerkt of verwijderd in GitHub, GitLab of Bitbucket.

Uw webhookdefinitie moet een Git-eindpunt bevatten op de tak van de opslagplaats die u wilt bewaken. Om een webhook te maken, gebruikt Code Stream het Git-eindpunt. Als het eindpunt niet bestaat, kunt u dit maken wanneer u de webhook toevoegt. In dit voorbeeld wordt ervan uitgegaan dat u een vooraf gedefinieerd Git-eindpunt heeft in GitHub.

Opmerking Als u een webhook wilt maken, moet uw Git-eindpunt een privétoken voor verificatie gebruiken. Er kan geen wachtwoord worden gebruikt.

U kunt meerdere webhooks voor verschillende takken maken door hetzelfde Git-eindpunt te gebruiken en verschillende waarden voor de naam van de tak op te geven op de pagina voor webhookconfiguratie. Als u een andere webhook voor een andere tak in dezelfde Git-opslagplaats wilt maken, hoeft u het Git-eindpunt niet meerdere keren te klonen voor meerdere takken. In plaats daarvan geeft u de naam van de tak op in de webhook, zodat u het Git-eindpunt opnieuw kunt gebruiken. Als de tak op de Git-webhook hetzelfde is als de tak op het eindpunt, hoeft u geen taknaam op de pagina van de Git-webhook op te geven.

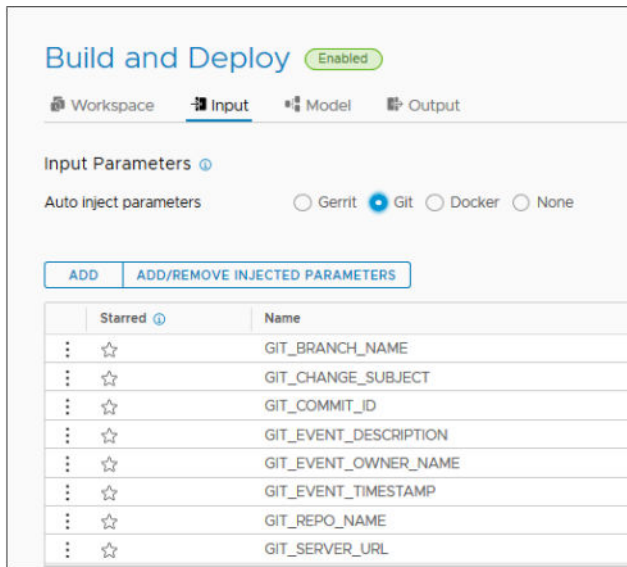
In dit voorbeeld ziet u hoe u de Git-trigger kunt gebruiken met een GitHub-opslagplaats, maar de vereisten omvatten voorbereidingen die vereist zijn als u een ander Git-servertype gebruikt.

Voorwaarden

- Controleer of u lid bent van een project in Code Stream. Als u geen beheerder bent, vraagt u de Code Stream-beheerder om u als lid toe te voegen aan een project. Zie [Hoe voeg ik een project toe in Code Stream?](#).
- Controleer of u een Git-eindpunt heeft op de GitHub-tak die u wilt bewaken. Zie [Hoe integreer ik Code Stream met Git?](#).
- Controleer of u rechten heeft om een webhook te maken in de Git-opslagplaats.
- Als u een webhook in GitLab configureert, wijzigt u de standaardnetwerkinstellingen in GitLab Enterprise om uitgaande aanvragen in te schakelen en om het creëren van lokale webhooks mogelijk te maken.

Opmerking Deze wijziging is alleen vereist voor GitLab Enterprise. Deze instellingen zijn niet van toepassing op GitHub of Bitbucket.

- a Meld u aan bij uw GitLab Enterprise-instantie als beheerder.
 - b Ga naar netwerkinstellingen met behulp van een URL, zoals `http://{gitlab-server}/admin/application_settings/network`.
 - c Vouw **Uitgaande aanvragen** uit en klik op:
 - Aanvragen voor het lokale netwerk toestaan via webhooks en services.
 - Aanvragen voor het lokale netwerk vanuit de systeemhaak toestaan.
- Controleer voor de pijplijnen die u wilt activeren of u de invoereigenschappen heeft ingesteld om Git-parameters te injecteren wanneer de pijplijn wordt uitgevoerd.



Zie [Een CI/CD-systeemeigen build plannen in Code Stream voordat u handmatig taken toevoegt](#) voor informatie over invoerparameters.

Procedure

- 1 Klik in Code Stream op **Triggers > Git**.
- 2 Klik op het tabblad **Webhooks voor Git** en klik vervolgens op **Nieuwe webhook voor Git**.
 - a Selecteer een project.
 - b Voer een relevante naam en beschrijving voor de webhook in.

- c Selecteer een Git-eindpunt dat is geconfigureerd op de tak die u wilt bewaken.

Wanneer u uw webhook maakt, bevat de webhook-definitie de huidige eindpuntdetails.

- Als u later het Git-type, het Git-servertype of de URL van de Git-opslagplaats in het eindpunt wijzigt, kan de webhook geen pijplijn meer activeren omdat deze toegang probeert te krijgen tot de Git-opslagplaats met behulp van de oorspronkelijke eindpuntgegevens. U moet de webhook verwijderen en deze opnieuw maken met het eindpunt.
- Als u later het verificatietype, de gebruikersnaam of het privétoken in het eindpunt wijzigt, blijft de webhook werken.
- Als u een BitBucket-opslagplaats gebruikt, moet de URL voor de opslagplaats een van de volgende indelingen hebben: `https://api.bitbucket.org/{user}/{repo name}` of `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`.

Opmerking Als u eerder een webhook hebt gemaakt met een Git-eindpunt dat een wachtwoord voor basisverificatie gebruikt, moet u de webhook verwijderen en opnieuw definiëren met een Git-eindpunt dat een privétoken voor verificatie gebruikt.

Zie [Hoe integreer ik Code Stream met Git?](#).

- d (Optioneel) Voer de tak in die de webhook moet bewaken.

Als u de tak niet opgeeft, controleert de webhook de tak die u hebt geconfigureerd voor het Git-eindpunt.

- e (Optioneel) Genereer een geheime token voor de webhook.

Als u een geheim token gebruikt, genereert Code Stream een willekeurig tekenreekstoken voor de webhook. Wanneer de webhook vervolgens gegevens van de Git-gebeurtenis ontvangt, verzendt deze de gegevens met de geheime token. Code Stream gebruikt de informatie om te bepalen of de oproepen afkomstig zijn van de verwachte bron, zoals de geconfigureerde GitHub-instantie, -opslagplaats en -tak. De geheime token biedt een extra beveiligingslaag die wordt gebruikt om te verifiëren of de gegevens van de Git-gebeurtenis afkomstig zijn van de juiste bron.

f (Optioneel) Geef bestandsopnamen of -uitsluitingen als voorwaarden voor de trigger op.

- Bestandsopnamen. Als een van de bestanden in een commit overeenkomt met de bestanden die zijn opgegeven in de opnamepaden of regex, activeert de commit de pijplijnen. Met een opgegeven regex worden door Code Stream alleen de pijplijnen geactiveerd wanneer bestandsnamen in de wijzigingsset overeenkomen met de opgegeven expressie. Het regex-filter is handig bij het configureren van een trigger voor meerdere pijplijnen op één opslagplaats.
- Bestandsuitsluitingen. Wanneer alle bestanden in een commit overeenstemmen met de bestanden die zijn opgegeven in de uitsluitingspaden of regex, worden de pijplijnen niet geactiveerd.
- Uitsluitingen prioriteren. Wanneer Uitsluiting prioriteren is ingeschakeld, wordt ervoor gezorgd dat er geen pijplijnen worden geactiveerd, zelfs niet als een van de bestanden in een commit overeenkomt met de bestanden die in de uitsluitingspaden of regex zijn opgegeven. De standaardinstelling is uitgeschakeld.

Als voorwaarden voldoen aan zowel bestandsopnamen als bestandsuitsluitingen, worden pijplijnen niet geactiveerd.

In het volgende voorbeeld zijn zowel bestandsopnamen als -uitsluitingen voorwaarden voor de trigger.

The screenshot shows a configuration window titled "File" with an information icon. It contains two main sections: "Inclusions" and "Exclusions". Each section has a list of rules with columns for the filter type (PLAIN or REGEX), the path or expression, and buttons to add or remove the rule. At the bottom, there is a "Prioritize Exclusion" toggle switch which is currently turned off.

Section	Filter Type	Path/Expression	Buttons
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: ☐

- Bij bestandsopnamen activeert een commit met elke wijziging van `runtime/src/main/a.java` of elk Java-bestand pijplijnen die in de gebeurtenisconfiguratie zijn ingesteld.
- Bij bestandsuitsluitingen worden de pijplijnen die in de gebeurtenisconfiguraties zijn ingesteld, niet geactiveerd door een commit met alleen wijzigingen in beide bestanden.

g Voor de Git-gebeurtenis selecteert u een **Push**- of **Pull**-aanvraag.

h Voer de API-token in.

Het CSP API-token verifieert u voor externe API-verbindingen met Code Stream. Om het API-token te verkrijgen:

- 1 Klik op **Token genereren**.
- 2 Voer het e-mailadres in dat is gekoppeld aan uw gebruikersnaam en wachtwoord en klik op **Genereren**.

Het token dat u genereert, is zes maanden geldig. Het wordt ook wel een vernieuwingstoken genoemd.

- Als u het token als een variabele wilt behouden voor toekomstig gebruik klikt u op **Variabele maken**, voert u een naam voor de variabele in en klikt u op **Opslaan**.
- Als u het token als tekstwaarde wilt behouden voor toekomstig gebruik klikt u op **Kopiëren** en plakt u het token in een tekstbestand om lokaal op te slaan.

U kunt ervoor kiezen om beide een variabele te maken en het token in een tekstbestand op te slaan voor toekomstig gebruik.

- 3 Klik op **Sluiten**.

i Selecteer de pijplijn om de webhook te activeren.

Als de pijplijn aangepaste toegevoegde invoerparameters bevat, worden parameters en waarden weergegeven in de lijst Invoerparameters. U kunt waarden invoeren voor invoerparameters die worden doorgegeven naar de pijplijn met de triggergebeurtenis. Of u kunt de waarden leeg laten of de standaardwaarden gebruiken, indien gedefinieerd.

Zie de [Vereisten](#) voor informatie over het Automatisch injecteren van invoerparameters voor Git-triggers.

j Klik op **Maken**.

De webhook wordt weergegeven als een nieuwe kaart.

3 Klik op de webhook-kaart.

Wanneer het gegevensformulier van de webhook opnieuw wordt weergegeven, ziet u een webhook-URL die is toegevoegd aan de bovenkant van het formulier. De Git-webhook maakt verbinding met de GitHub-opslagplaats via de webhook-URL.

Git

Activity

Webhooks for Git

Webhook URL ⓘ

https://ca811b68-184c-484c-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

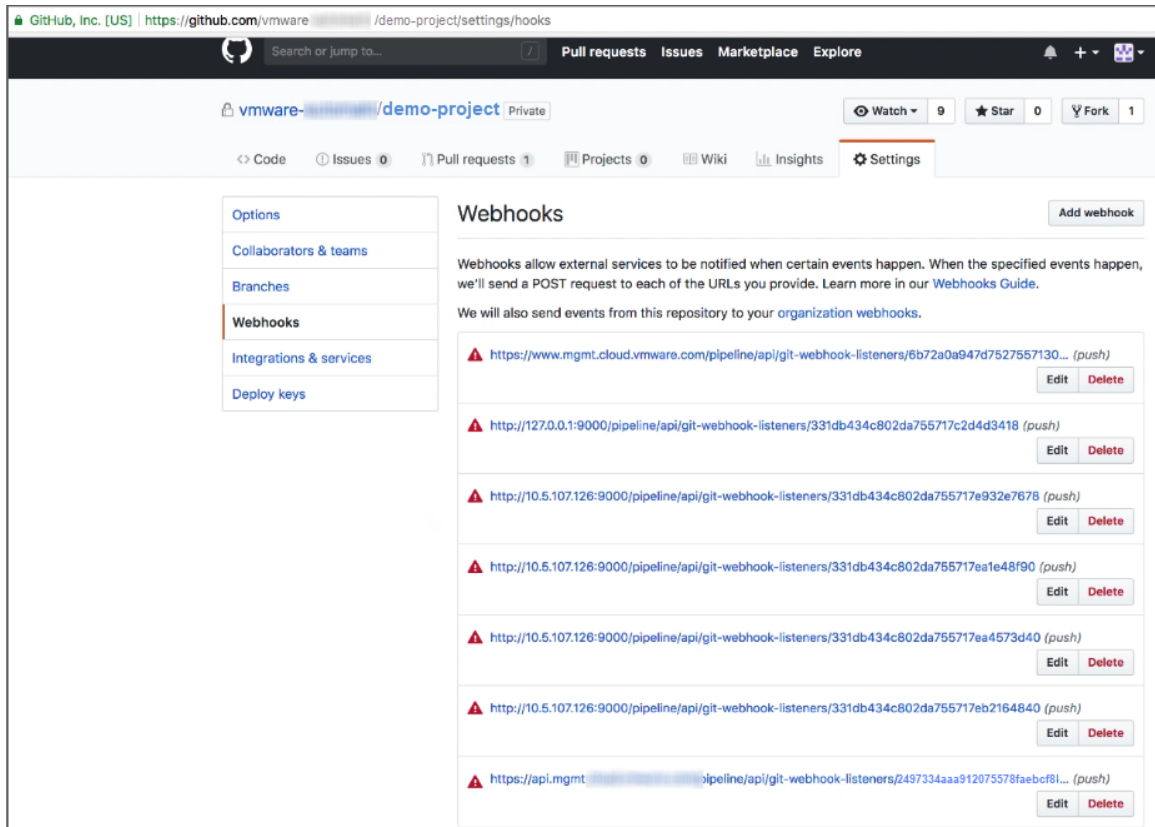
⌵

SAVE

CANCEL

- 4 Open in een nieuw browservenster de GitHub-opslagplaats die is verbonden via de webhook.
 - a Om de webhook te zien die u heeft toegevoegd in Code Stream, klikt u op het tabblad **Instellingen** en selecteert u **Webhooks**.

Onder in de lijst met webhooks ziet u dezelfde webhook-URL.



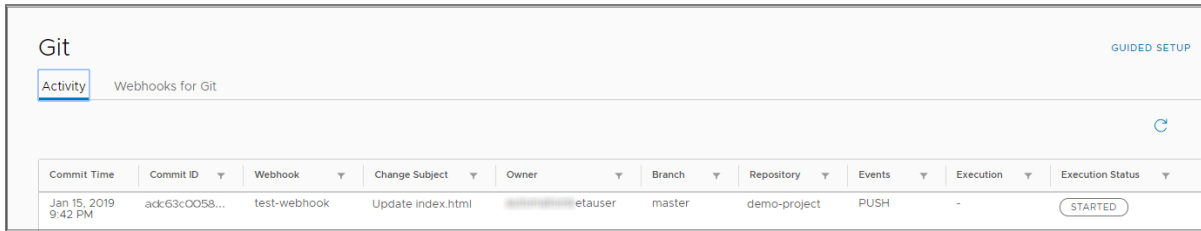
- b Als u code wilt wijzigen, klikt u op het tabblad **Code** en selecteert u een bestand in de tak. Nadat u het bestand hebt bewerkt, voert u de wijziging door.
 - c Om te controleren of de URL van de webhook werkt, klikt u op het tabblad **Instellingen** en selecteert u opnieuw **Webhooks**.

Onderaan de lijst met webhooks wordt een groen vinkje weergegeven naast de URL van de webhook.



- 5 Ga terug naar Code Stream om de activiteit op de Git-webhook te bekijken. Klik op **Triggers > Git > Activiteit**.

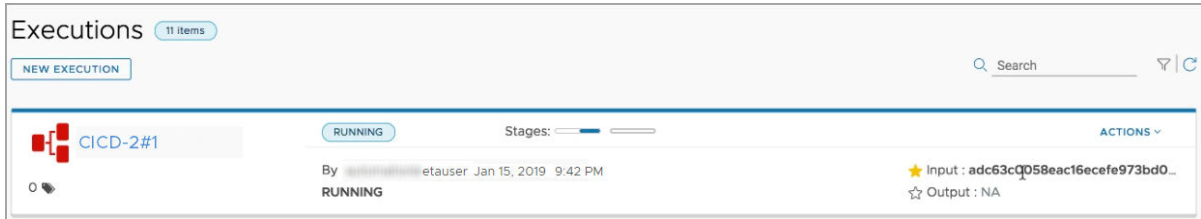
Controleer onder Uitvoeringsstatus of de pijplijn-uitvoering is gestart.




Commit Time	Commit ID	Webhook	Change Subject	Owner	Branch	Repository	Events	Execution	Execution Status
Jan 15, 2019 9:42 PM	ack63c0058...	test-webhook	Update index.html	etauser	master	demo-project	PUSH	-	STARTED

6 Klik op **Uitvoeringen** en volg uw pijplijn tijdens de uitvoering.

Om de pijplijnuitvoering te bekijken, kunt u op Vernieuwen klikken.



Executions		11 Items
NEW EXECUTION		
<div>  CICD-2#1 </div>		
<div> RUNNING </div>		
<div> Stages: 0/1 </div>		
<div> By etauser Jan 15, 2019 9:42 PM </div>		
<div> RUNNING </div>		
<div> Input : adc63c0058ac16ecef973bd0... </div>		
<div> Output : NA </div>		

Resultaten

Gefeliciteerd! U hebt de trigger voor Git met succes gebruikt.

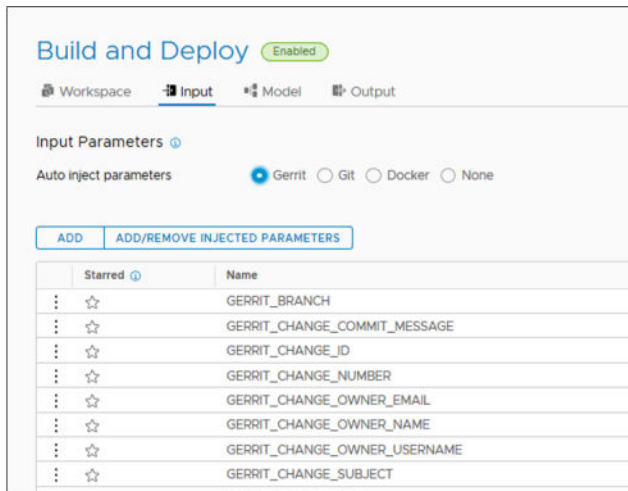
Hoe gebruik ik de Gerrit-trigger in Code Stream om een pijplijn uit te voeren?

Als Code Stream-beheerder of -ontwikkelaar kunt u Code Stream integreren met de beoordelingslevenscyclus van de Gerrit-code met behulp van de Gerrit-trigger. De gebeurtenis zorgt ervoor dat een pijplijn wordt uitgevoerd wanneer u een patchset maakt, concepten publiceert, codewijzigingen samenvoegt in het Gerrit-project of rechtstreeks wijzigingen aanbrengt in de Git-tak.

Wanneer u de Gerrit-trigger toevoegt, selecteert u een Gerrit-listener, een Gerrit-project op de Gerrit-server, en configureert u Gerrit-gebeurtenissen. In dit voorbeeld configureert u eerst een Gerrit-listener en vervolgens gebruikt u die listener in een Gerrit-trigger met twee gebeurtenissen op drie verschillende pijplijnen.

Voorwaarden

- Controleer of u lid bent van een project in Code Stream. Als u geen beheerder bent, vraagt u de Code Stream-beheerder om u als lid toe te voegen aan een project. Zie [Hoe voeg ik een project toe in Code Stream?](#).
- Controleer of u een Gerrit-eindpunt heeft geconfigureerd in Code Stream. Zie [Hoe integreer ik Code Stream met Gerrit?](#).
- Zorg ervoor dat u weet welke Gerrit-releaseversie u heeft.
- Om pijplijnen te activeren, moet u controleren of u de invoereigenschappen van de pijplijn hebt ingesteld op **Gerrit**, waardoor de pijplijn de Gerrit-parameters als invoer kan ontvangen wanneer de pijplijn wordt uitgevoerd.



Zie [Een CI/CD-systeemeigen build plannen in Code Stream voordat u handmatig taken toevoegt](#) voor informatie over invoerparameters.

Procedure

- 1 Klik in Code Stream op **Triggers > Gerrit**.
- 2 (Optioneel) Klik op het tabblad **Luisteraars** en klik vervolgens op **Nieuwe luisteraar**.

Opmerking Als de Gerrit-listener die u voor de Gerrit-trigger wilt gebruiken al is gedefinieerd, slaat u deze stap over.

- a Selecteer een project.
- b Voer een naam in voor Gerrit-luisteraar.
- c Selecteer een Git-eindpunt.

- d Voer de API-token in.

Het CSP API-token verifieert u voor externe API-verbindingen met Code Stream. Om het API-token te verkrijgen:

- 1 Klik op **Token genereren**.
- 2 Voer het e-mailadres in dat is gekoppeld aan uw gebruikersnaam en wachtwoord en klik op **Genereren**.

Het token dat u genereert, is zes maanden geldig. Het wordt ook wel een vernieuwingstoken genoemd.

- Als u het token als een variabele wilt behouden voor toekomstig gebruik klikt u op **Variabele maken**, voert u een naam voor de variabele in en klikt u op **Opslaan**.
- Als u het token als tekstwaarde wilt behouden voor toekomstig gebruik klikt u op **Kopiëren** en plakt u het token in een tekstbestand om lokaal op te slaan.

U kunt ervoor kiezen om beide een variabele te maken en het token in een tekstbestand op te slaan voor toekomstig gebruik.

- 3 Klik op **Sluiten**.

Als u een variabele hebt gemaakt, wordt in het API-token de naam van de variabele weergegeven die u hebt ingevoerd met de dollarbinding. Als u de token heeft gekopieerd, geeft het API-token de gemaskeerde token weer.

The screenshot shows the 'Gerrit' configuration interface with the 'Listeners' tab selected. The form contains the following fields and controls:

- Project**: A text field containing 'test1' with a clear button (X).
- Name**: A text field containing 'Gerrit-Demo-Listener'.
- Endpoint**: A dropdown menu showing 'corporate-gerrit'.
- API token**: A text field containing the variable reference '\$var.CSuser API Token' with a green checkmark icon.
- Buttons**: At the bottom are three buttons: 'CREATE' (solid blue), 'VALIDATE' (outline blue), and 'CANCEL' (outline blue). To the right of the API token field are two buttons: 'CREATE VARIABLE' and 'GENERATE TOKEN' (both outline blue).

- e Om de gegevens van het token en eindpunt te valideren, klikt u op **Valideren**.

Uw token verloopt na 90 dagen.

f Klik op **Maken**.

g Klik op **Verbinden** op de luisteraarskaart.

De luisteraar start de bewaking van alle activiteiten op de Gerrit-server en luistert om een ingeschakelde triggers op die server te horen. Als u wilt stoppen met luisteren naar een trigger op die server, deactiveert u de trigger.

Opmerking Om een Gerrit-eindpunt bij te werken dat is verbonden met een luisteraar, moet u de verbinding met de luisteraar verbreken voordat u het eindpunt bijwerkt.

- Klik op **Configureren > Triggers > Gerrit**.
 - Klik op het tabblad **Luisteraars**.
 - Klik op **Loskoppelen** op de luisteraar die is verbonden met het eindpunt dat u wilt bijwerken.
-

3 Klik op het tabblad **Triggers** en klik vervolgens op **Nieuwe trigger**.

4 Selecteer een project op de Gerrit-server.

5 Voer een naam in.

De naam van de Gerrit-trigger moet uniek zijn.

6 Selecteer een geconfigureerde Gerrit-luisteraar.

Door de Gerrit-listener te gebruiken, biedt Code Stream een lijst met Gerrit-projecten die beschikbaar zijn op de server.

7 Selecteer een project op de Gerrit-server.

8 Voer de tak in de opslagplaats in die de Gerrit-listener zal bewaken.

9 (Optioneel) Geef bestandsopnamen of -uitsluitingen als voorwaarden voor de trigger op.

- U geeft bestandsopnamen op die de pijplijnen activeren. Wanneer een van de bestanden in een commit overeenkomt met de bestanden die zijn opgegeven in de opnamepaden of regex, worden pijplijnen geactiveerd. Met een opgegeven regex worden door Code Stream alleen de pijplijnen geactiveerd met bestandsnamen in de wijzigingsset die overeenkomen met de opgegeven expressie. Het regex-filter is handig bij het configureren van een trigger voor meerdere pijplijnen op één opslagplaats.
- U geeft bestandsuitsluitingen op die voorkomen dat pijplijnen worden geactiveerd. Wanneer alle bestanden in een commit overeenstemmen met de bestanden die zijn opgegeven in de uitsluitingspaden of regex, worden de pijplijnen niet geactiveerd.
- Wanneer **Uitsluiting prioriteren** is ingeschakeld, zorgt u ervoor dat pijplijnen niet worden geactiveerd. De pijplijnen worden niet geactiveerd, ook als een van de bestanden in een commit overeenkomt met de bestanden die zijn opgegeven in de uitsluitingspaden of regex. De standaardinstelling voor **Uitsluiting prioriteren** is uitgeschakeld.

Als de voorwaarden voldoen aan zowel de bestandsopname als de bestandsuitsluiting, worden pijplijnen niet geactiveerd.

In het volgende voorbeeld zijn zowel bestandsopnamen als -uitsluitingen voorwaarden voor de trigger.

The screenshot shows a configuration window titled 'File' with an information icon. It contains two sections: 'Inclusions' and 'Exclusions'. Each section has a list of items with a dropdown menu for the match type and a button to add or remove the item. At the bottom, there is a 'Prioritize Exclusion' toggle switch.

Section	Match Type	File Path	Action
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: ☐

- Bij bestandsopnamen activeert een commit die een wijziging in `runtime/src/main/a.java` of een Java-bestand heeft, de pijplijnen die in de gebeurtenisconfiguratie zijn ingesteld.
- Bij bestandsuitsluitingen worden de pijplijnen die in de gebeurtenisconfiguratie zijn ingesteld, niet geactiveerd door een commit met alleen wijzigingen in beide bestanden.

10 Klik op **Nieuwe configuratie**.

- Selecteer voor de Gerrit-gebeurtenis **Patchset gemaakt**, **Concept gepubliceerd** of **Wijziging samengevoegd**. Voor een directe push naar Git die Gerrit omzeilt, selecteert u **Directe Git-push**.

Opmerking Vanaf Gerrit-releaseversie 2.15 worden conceptwijzigingen en conceptwijzigingssets niet langer ondersteund. Als u Gerrit-releaseversie 2.15 of hoger uitvoert, is **Draft Published** geen ondersteunde gebeurtenis.

- Selecteer de pijplijn die wordt geactiveerd.

Als de pijplijn aangepaste toegevoegde invoerparameters bevat, worden parameters en waarden weergegeven in de lijst Invoerparameters. U kunt waarden invoeren voor invoerparameters die worden doorgegeven naar de pijplijn met de triggergebeurtenis. Of u kunt de waarden leeg laten of de standaardwaarden gebruiken.

Opmerking Als standaardwaarden worden gedefinieerd:

- Alle waarden die u voor de invoerparameters invoert, overschrijven de standaardwaarden die in het pijplijnmodel zijn gedefinieerd.
- De standaardwaarden in de triggerconfiguratie worden niet gewijzigd als de parameterwaarden in het pijplijnmodel worden gewijzigd.

Zie de [Vereisten](#) voor informatie over het Automatisch injecteren van invoerparameters voor Gerrit-triggers.

- c Voor **Patchset gemaakt**, **Concept gepubliceerd** en **Wijzigingen samengevoegd** worden sommige acties standaard met labels weergegeven. U kunt het label wijzigen of opmerkingen toevoegen. Wanneer de pijplijn wordt uitgevoerd, verschijnt het label of de opmerking op het tabblad **Activiteit** als de **Ondernomen actie** voor de pijplijn.

Met de configuratie van de Gerrit-gebeurtenis kunt u opmerkingen invoeren met behulp van een variabele voor de opmerking Gelukt of Mislukt. Bijvoorbeeld: `${var.success}` en `${var.failure}`.

- d Klik op **Opslaan**.

Als u meerdere triggergebeurtenissen wilt toevoegen aan meerdere pijplijnen, klikt u opnieuw op **Nieuwe configuratie**.

In het volgende voorbeeld ziet u gebeurtenissen voor drie pijplijnen:

- Als een gebeurtenis **Wijziging samengevoegd** in het Gerrit-project plaatsvindt, wordt de pijplijn met de naam **Gerrit-pijplijn** geactiveerd.
- Als een gebeurtenis **Patchset gemaakt** plaatsvindt in het Gerrit-project, worden de pijplijnen met de naam **Gerrit-trigger-pijplijn** en de **Gerrit-demo-pijplijn** geactiveerd.

Gerrit

GUIDED SETUP

Activity
Triggers
Listeners

Project *

test1

ⓧ

Name *

Gerrit-Demo-Trigger

Gerrit Listener *

Gerrit-Demo-Listener

▼

Gerrit project *

Gerrit-Demo-Project

▼

Branch *

master

File ⓘ

Inclusions

-- Select Type --

value

+

Exclusions

-- Select Type --

value

+

Prioritize Exclusion

+ NEW CONFIGURATION

	Event Type	Pipeline	Label
⋮	Change Merged	Gerrit-Pipeline	Verified
⋮	Patchset Created	Gerrit-Trigger-Pipeline	Verified
⋮	Patchset Created	Gerrit-Demo-Pipeline	Verified
3 configurations			

11 Klik op **Maken**.

De Gerrit-trigger wordt weergegeven als nieuwe kaart op het tabblad **Triggers** en is standaard ingesteld op **Uitgeschakeld**.

12 Klik op de triggerkaart op **Inschakelen**.

Nadat de trigger is ingeschakeld, gebruikt deze de Gerrit-listener, die begint met het bewaken van gebeurtenissen die plaatsvinden op de tak van het Gerrit-project.

Als u een trigger wilt maken met dezelfde voorwaarden voor bestandopname of de voorwaarden voor bestandsuitsluiting, maar met een andere opslagplaats dan die u hebt opgenomen toen u de trigger hebt gemaakt, klikt u op de triggerkaart op **Acties > Klonen**. Klik vervolgens op de gekloonde trigger op **Openen** en wijzig de parameters.

Resultaten

Gefeliciteerd! U hebt een Gerrit-trigger met twee gebeurtenissen op drie verschillende pijplijnen geconfigureerd.

Wat nu te doen

Nadat u een codewijziging hebt doorgevoerd in het Gerrit-project, bekijkt u het tabblad **Activiteit** voor de Gerrit-gebeurtenis in Code Stream. Controleer of de lijst met activiteiten vermeldingen bevat die overeenkomen met elke pijplijnuitvoering in de triggerconfiguratie.

Wanneer een gebeurtenis plaatsvindt, kunnen alleen pijplijnen in de Gerrit-trigger die betrekking hebben op het specifieke type gebeurtenis worden uitgevoerd. Als in dit voorbeeld een patchset is gemaakt, worden alleen de **Gerrit-trigger-pijplijn** en de **Gerrit-demo-pijplijn** uitgevoerd.

Informatie in de kolommen op het tabblad **Activiteit** beschrijven elke Gerrit-triggergebeurtenis. U kunt de kolommen selecteren die worden weergegeven door op het kolompictogram te klikken dat onder de tabel wordt weergegeven.

- De kolommen **Onderwerp wijzigen** en **Uitvoeren** zijn leeg als de trigger een directe Git-push was.
- De kolom **Gerrit-trigger** toont de trigger die de gebeurtenis heeft gemaakt.
- De kolom **Listener** is standaard uitgeschakeld. Wanneer u deze selecteert, wordt in de kolom de Gerrit-listener weergegeven die de gebeurtenis heeft ontvangen. Eén listener kan worden weergegeven als gekoppeld aan meerdere triggers.
- De kolom **Triggertype** is standaard uitgeschakeld. Wanneer u deze selecteert, wordt in de kolom het type trigger weergegeven als AUTOMATISCH of HANDMATIG.
- Andere kolommen zijn **Committijd**, **Wijzigingsnr.**, **Status**, **Bericht**, **Ondernomen actie**, **Gebruiker**, **Gerrit-project**, **Tak** en **Gebeurtenis**.

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: opnyan@vm	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: opnyan@vm	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: opnyan@vm	test1	Gerrit-Demo-Trigger	master	Patchset created
			1111Dummy	Gerrit-Trigger-Pipeline #1	WAITING	Stage0.Task0: Execution Waiting for User Action.		Orlando: opnyan@vm	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

Als u de activiteit voor een voltooide of mislukte pijplijnuitvoering wilt regelen, klikt u op de drie punten links van een item in het scherm Activiteit.

- Als de pijplijn niet kan worden uitgevoerd vanwege een fout in het pijplijnmodel of een ander probleem, corrigeert u de fout en selecteert u **Opnieuw uitvoeren**, waarmee de pijplijn opnieuw wordt uitgevoerd.
- Als de pijplijn niet kan worden uitgevoerd vanwege een probleem met de netwerkverbinding of een ander probleem, selecteert u **Hervatten**, waarmee dezelfde pijplijnuitvoering wordt hervat en de runtime wordt opgeslagen.
- Gebruik **Uitvoering weergeven**, waarmee de weergave van de pijplijnuitvoering wordt geopend. Zie [Hoe kan ik een pijplijn uitvoeren en de resultaten bekijken?](#).
- Gebruik **Verwijderen** om de vermelding uit het Activiteiten-scherm te verwijderen.

Als een Gerrit-gebeurtenis een pijplijn niet activeert, kunt u handmatig op **Handmatig activeren** klikken, vervolgens de Gerrit-trigger selecteren, de wijzigings-id invoeren en op **Uitvoeren** klikken.

Pijplijnen bewaken in Code Stream



Als Code Stream-beheerder of -ontwikkelaar moet u inzicht hebben in de prestaties van uw pijplijnen in Code Stream. U moet weten hoe effectief uw pijplijnen code vrijgeven vanaf de ontwikkeling, via het testen en naar de productie.

Om inzicht te krijgen, gebruikt u Code Stream-dashboards om de trends en resultaten van een pijplijnuitvoering te bewaken. U kunt de standaard pijplijndashboards gebruiken om één pijplijn te bewaken of om custom dashboards te maken om meerdere pijplijnen te bewaken.

- Pijplijn-metrieken omvatten statistieken zoals gemiddelde tijden, die beschikbaar zijn op het pijplijndashboard.
- Gebruik de custom dashboards om metrieken over meerdere pijplijnen te bekijken.

Dit hoofdstuk omvat de volgende onderwerpen:

- [Wat toont het pijplijndashboard mij in Code Stream?](#)
- [Hoe kan ik aangepaste dashboards gebruiken om KPI's voor mijn pijplijn in Code Stream te volgen?](#)

Wat toont het pijplijndashboard mij in Code Stream?

Een pijplijndashboard is een weergave van de resultaten voor een specifieke pijplijn die is uitgevoerd, zoals trends, meest voorkomende fouten en geslaagde wijzigingen. Code Stream maakt het pijplijndashboard wanneer u een pijplijn maakt.

Het dashboard bevat de widgets waarin de resultaten van de pijplijnuitvoering worden weergegeven.

De widget Aantal uitvoeringsstatussen van pijplijn

U kunt het totale aantal uitvoeringen van een pijplijn gedurende een bepaalde tijdsperiode weergeven, gegroepeerd op status: Voltooid, Mislukt of Geannuleerd. Als u wilt zien hoe de uitvoeringsstatus van de pijplijn over langere of kortere tijdsperioden is gewijzigd, wijzigt u de duur op het scherm.

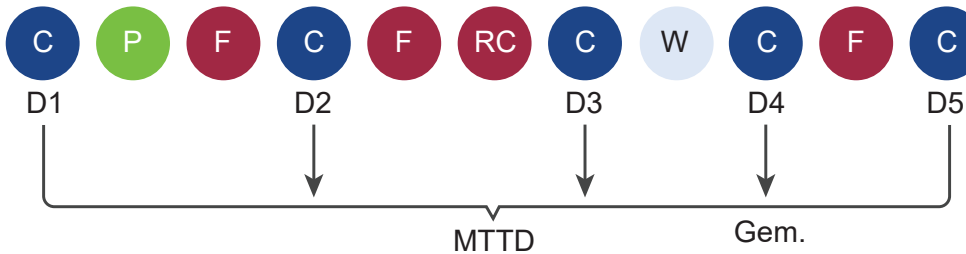
Widget Uitvoeringsstatistieken van pijplijn

De uitvoeringsstatistieken van de pijplijn bevatten de gemiddelde tijden voor het herstellen, leveren of mislukken van een pijplijn in de tijd.

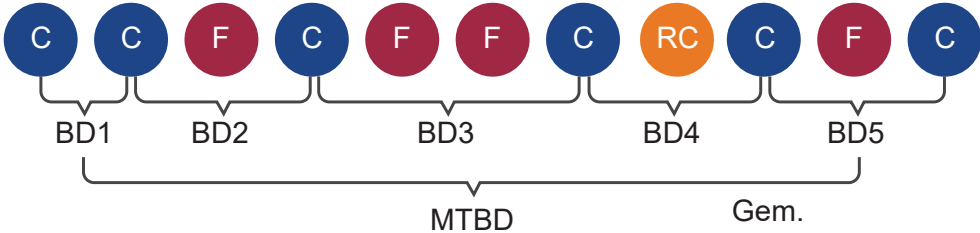
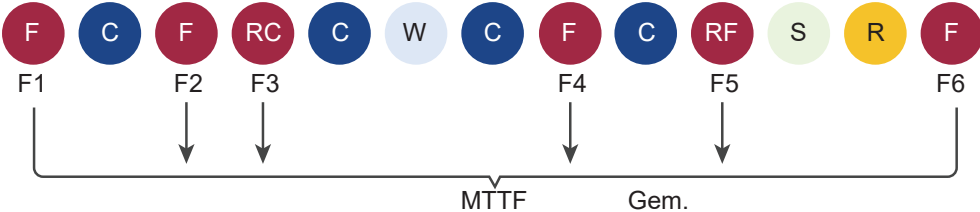
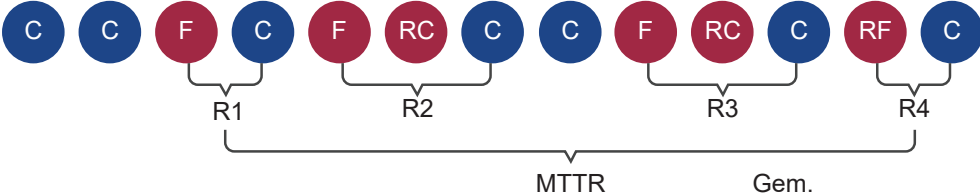
De volgende staten zijn van toepassing op alle pijplijnuitvoeringen:

- Voltooid
- Mislukt
- Wachten
- Wordt uitgevoerd
- Geannuleerd
- In wachtrij
- Niet gestart
- Terugdraaien
- Terugdraaien voltooid
- Terugdraaien mislukt
- Onderbroken

Tabel 8-1. Gemiddelde tijden meten

Wat wordt gemeten...	Wat dit betekent...
Gemiddelde CI	Gemiddelde tijd die wordt besteed aan de continue integratiefase, gemeten als tijd in het CI-taaktype.
Gemiddelde tijd tot levering (MTTD)	<p>Gemiddelde duur van alle VOLTOOIDE uitvoeringen gedurende een bepaalde tijdsperiode. D1, D2, enz. is de hoeveelheid tijd voor het leveren van elke VOLTOOIDE uitvoering.</p> 

Tabel 8-1. Gemiddelde tijden meten (vervolg)

Wat wordt gemeten...	Wat dit betekent...
Gemiddelde tijd tussen leveringen (MTBD)	<p>Gemiddelde tijd die is verstreken tussen geslaagde leveringen in een bepaalde tijdsperiode. De tijd die is verstreken tussen twee opeenvolgende VOLTOOIDE uitvoeringen is de tijd tussen succesvolle leveringen, zoals BD1, BD2, enz. MTBD geeft aan hoe vaak een productieomgeving wordt bijgewerkt.</p> 
Gemiddelde tijd tot fout (MTTF)	<p>Gemiddelde duur van uitvoeringen die eindigen met MISLUKT, TERUGDRAAIEN_VOLTOOID of TERUGDRAAIEN_MISLUKT over een bepaalde tijdsperiode. F1, F2, enzovoort is de hoeveelheid tijd dat een uitvoering nodig heeft om te eindigen met MISLUKT, TERUGDRAAIEN_VOLTOOID of TERUGDRAAIEN_MISLUKT.</p> 
Gemiddelde tijd tot herstel (MTTR)	<p>Gemiddelde tijd om te herstellen van een fout in een bepaalde tijdsperiode. De tijd om te herstellen van een fout is de tijd die is verstreken tussen een uitvoering met de eindstatus MISLUKT, TERUGDRAAIEN_VOLTOOID of TERUGDRAAIEN_MISLUKT en de volgende onmiddellijke succesvolle uitvoering met de status VOLTOOID. R1, R2, enz. is de hoeveelheid tijd om te herstellen na elke uitvoering met status MISLUKT of TERUGDRAAIEN_MISLUKT.</p> 

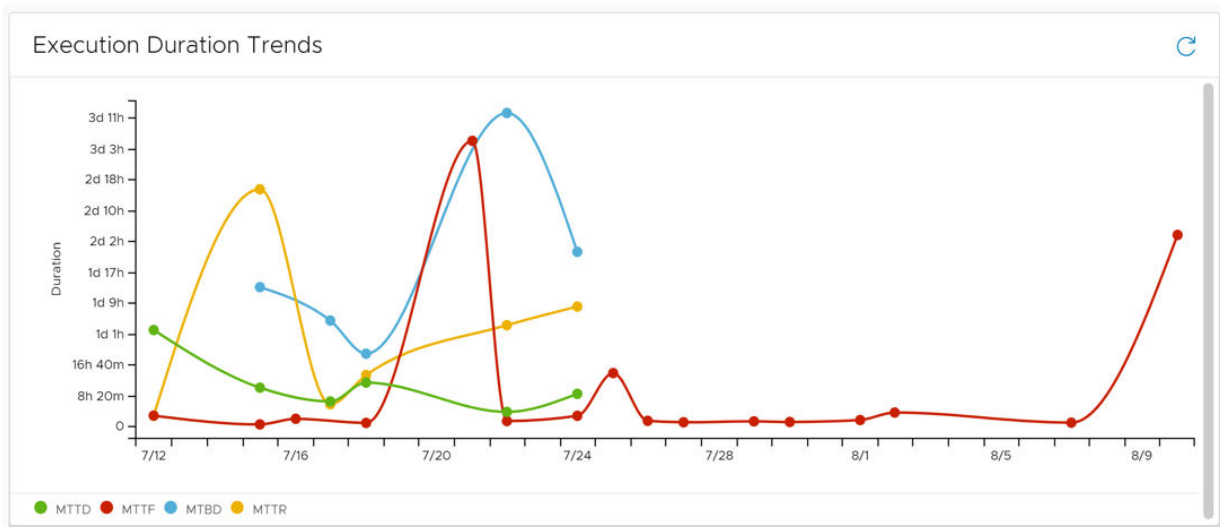
Widgets Belangrijkste mislukte fasen en Belangrijkste mislukte taken

Twee widgets geven de voornaamste mislukte fasen en taken in een pijplijn weer. Elke meting rapporteert het aantal en het percentage van de fouten voor ontwikkelings- en post-ontwikkelingsomgevingen voor elke pijplijn en elk project, gemiddeld over een week of maand. U bekijkt de meest voorkomende fouten om problemen in het release-automatiseringsproces op te lossen.

U kunt bijvoorbeeld de weergave configureren voor een bepaalde duur, zoals de laatste zeven dagen, en de meest voorkomende mislukte taken gedurende die tijdsperiode opmerken. Als u een wijziging aanbrengt in uw omgeving of pijplijn en de pijplijn opnieuw uitvoert, controleert u de meest voorkomende mislukte taken gedurende een langere tijdsperiode, zoals de laatste 14 dagen, aangezien de meest voorkomende mislukte taken kunnen zijn veranderd. Met dat resultaat weet u dat de wijziging in uw release-automatiseringsproces het succespercentage van uw pijplijn-uitvoering heeft verbeterd.

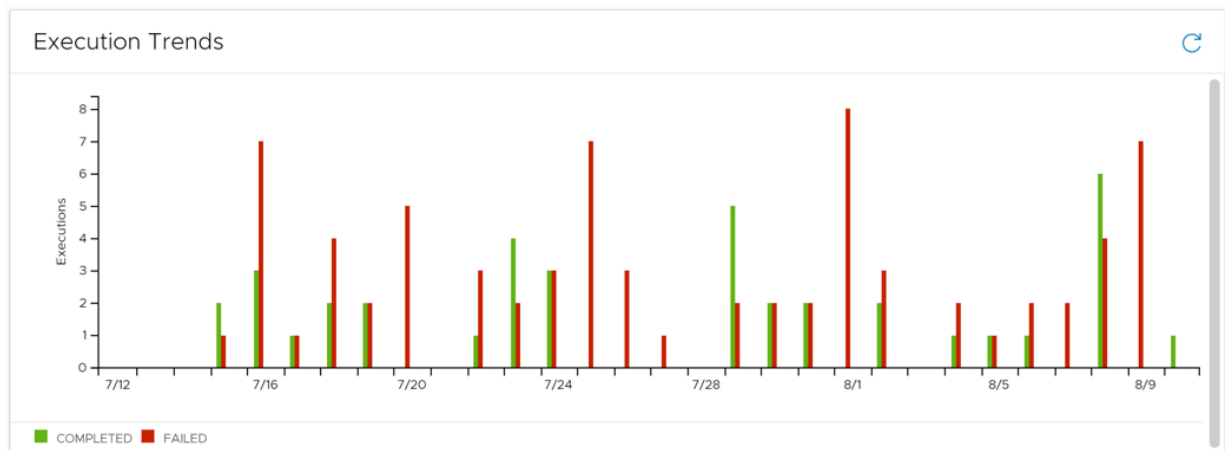
Widget Trends inzake uitvoeringsduur van pijplijn

Trends voor de uitvoeringsduur van pijplijnen tonen MTBD, MTTF, MTBD en MTTR in een bepaalde tijdsperiode.



Widget Uitvoeringstrends van pijplijn

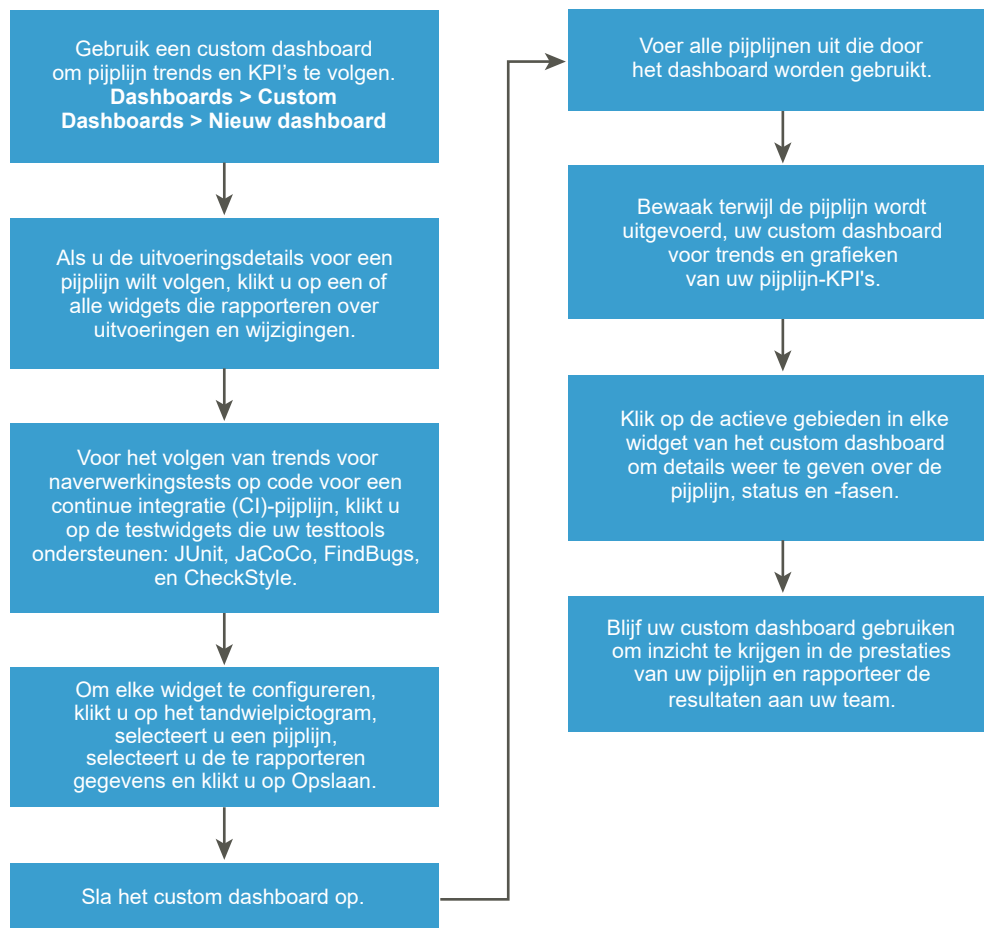
Uitvoeringstrends van pijplijn tonen de totale dagelijkse uitvoeringen van een pijplijn, gegroepeerd op status over een bepaalde tijdsperiode. Behalve voor de huidige dag worden voor de meeste dagelijkse aggregatietellingen alleen VOLTOOIDE en MISLUKTE uitvoeringen weergegeven.



Hoe kan ik aangepaste dashboards gebruiken om KPI's voor mijn pijplijn in Code Stream te volgen?

Als Code Stream-beheerder of -ontwikkelaar maakt u het aangepaste dashboard om de resultaten weer te geven die u wilt zien voor een of meer pijplijnen die zijn uitgevoerd. U kunt bijvoorbeeld een dashboard voor projecten maken met KPI's en metrieken die uit meerdere pijplijnen zijn verzameld. Als er een uitvoeringswaarschuwing of -fout wordt gemeld, kunt u het dashboard gebruiken om de fout op te lossen.

Om trends en KPI's voor uw pijplijnen te volgen met behulp van een custom dashboard, voegt u widgets toe aan het dashboard en configureert u ze om over uw pijplijnen te rapporteren.



Voorwaarden

- Verifieer dat een of meerdere pijplijnen bestaan. Klik in de gebruikersinterface op **Pijplijnen**.
- Controleer of de pijplijnen die u wilt bewaken met succes zijn uitgevoerd. Klik op **Uitvoeringen**.

Procedure

- 1 Als u een custom dashboard wilt maken, klikt u op **Dashboards > Custom dashboards > Nieuw dashboard**.

- 2 Klik op een widget om het dashboard zodanig aan te passen dat het specifieke trends en KPI's voor uw pijplijn rapporteert.

Als u bijvoorbeeld details wilt weergeven over de status van de pijplijn, fasen, taken, hoe lang deze zijn uitgevoerd en wie deze heeft uitgevoerd, klikt u op de **Uitvoeringsgegevens**-widget. Of, voor een pijplijn met continue integratie (CI), kunt u de naverwerkingstrends volgen met behulp van de widgets voor JUnit, JaCoCo, FindBugs en CheckStyle.

IX KPIs							
EDIT DELETE CLONE BACK							
Execution Details							
	Execution ID	Execution#	Status	Status Message	Stages	Tasks	Task0 (Stage0)
⋮	178f62eef...	#2	WAITING	Stage0.Task0 Execution Waiting for User Action.	●	⚠	⚠
⋮	5503c1e51...	#1	COMPLETED	Execution Completed.	●	✓	✓

- 3 Configureer elke widget die u toevoegt.
 - a Klik op de widget op het tandwielpictogram.
 - b Selecteer een pijplijn, stel de beschikbare opties in en selecteer de kolommen die u wilt weer geven.
 - c Klik op **Opslaan** om de configuratie van de widget op te slaan.
 - d Als u het custom dashboard wilt opslaan, klikt u op **Opslaan** en klikt u op **Sluiten**.
- 4 Als u meer informatie over de pijplijn wilt weergeven, klikt u op de actieve gebieden op de widgets.

Klik bijvoorbeeld: in de **Uitvoeringsgegevens**-widget op een vermelding in de kolom Status om meer informatie over het uitvoeren van de pijplijn weer te geven. Of als u in de **Laatste succesvolle wijziging**-widget een overzicht van de pijplijnfase en taak wilt weergeven, klikt u op de actieve link.

Resultaten

Gefeliciteerd! U heeft een custom dashboard gemaakt dat trends en KPI's voor uw pijplijnen bewaakt.

Wat nu te doen

Blijf de prestaties van uw pijplijnen in Code Stream bewaken en deel de resultaten met uw manager en teams om het proces voor het vrijgeven van uw applicaties te blijven verbeteren.

Meer informatie over Code Stream

9

Er zijn veel manieren voor Code Stream-beheerders en -ontwikkelaars om meer te weten te komen over Code Stream en wat u hiermee kunt doen.

U kunt deze documentatie gebruiken om meer te weten te komen over pijplijnen en de uitvoering ervan, hoe u eindpunten toevoegt, hoe u projecten toevoegt en meer.

Begrijp de rechten die rollen bieden. Ontdek hoe u beperkte resources gebruikt en goedkeuringen op pijplijnen vereist. Zie [Hoe beheer ik gebruikerstoegang en goedkeuringen in Code Stream?](#).

Ontdek de waarde van zoeken door te zien waar specifieke taken of onderdelen zich in uw pijplijnen, uitvoeringen of eindpunten bevinden.

Dit hoofdstuk omvat de volgende onderwerpen:

- [Wat is zoeken in Code Stream?](#)
- [Meer resources voor Code Stream-beheerders en -ontwikkelaars](#)

Wat is zoeken in Code Stream?

U gebruikt zoeken om te bepalen waar specifieke items of andere onderdelen zich bevinden. U kunt bijvoorbeeld zoeken naar geactiveerde of gedeactiveerde pijplijnen. Als een pijplijn is gedeactiveerd, kan deze namelijk niet worden uitgevoerd.

Waar kan ik in zoeken?

U kunt zoeken in:

- Projecten
- Eindpunten
- Pijplijnen
- Uitvoeringen
- Pijplijndashboards, custom dashboards
- Gerrit-triggers en -servers
- Git-webhooks

- Docker-webhooks

U kunt een zoekopdracht op basis van kolommen uitvoeren in:

- Gebruikersbewerkingen
- Variabelen
- Triggeractiviteit voor Gerrit, Git en Docker

U kunt een op rasters gebaseerde gefilterde zoekopdracht uitvoeren op de pagina **Activiteit** voor elke trigger.

Hoe werkt zoeken?

De zoekcriteria variëren afhankelijk van de pagina waarop u zich bevindt. Elke pagina heeft verschillende zoekcriteria.

Waar u zoekt	Criteria om voor zoeken te gebruiken
Pijplijndashboards	Project, naam, beschrijving, tags, link
Custom dashboards	Project, naam, beschrijving, link (UUID van een item op het dashboard)
Uitvoeringen	Naam, opmerkingen, reden, tags, index, status, project, weergeven, uitgevoerd door, uitgevoerd door mij, link (UUID van de uitvoering) en invoerparameters, uitvoerparameters of statusbericht door middel van de volgende indeling: <key>:<value>
Pijplijnen	Naam, beschrijving, status, tags, gemaakt door, gemaakt door mij, bijgewerkt door, bijgewerkt door mij, project
Projecten	Naam, beschrijving
Eindpunten	Naam, beschrijving, type, bijgewerkt door, project
Gerrit-triggers	Naam, status, project
Gerrit-servers	Naam, server-URL, project
Git-webhooks	Naam, servertype, opslagplaats, tak, project

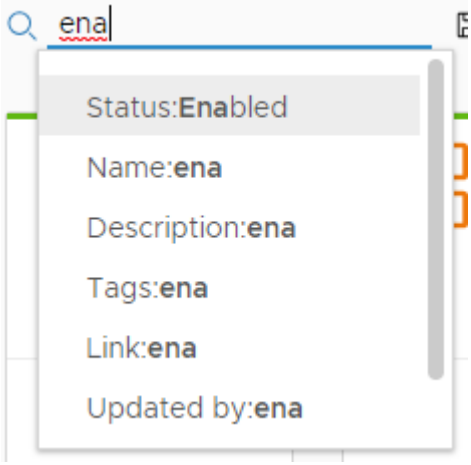
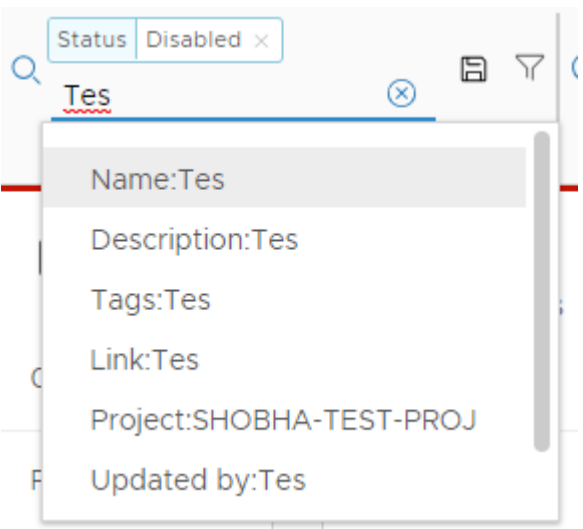
Waarbij:

- Link de UUID is van een pijplijn, uitvoering of widget op een dashboard.
- De invoerparameter-, uitvoerparameter- en statusberichtnotatie en -voorbeelden omvatten o.a.:
 - Notatie: `input.<inputKey>:<inputValue>`
 Voorbeeld: `input.GERRIT_CHANGE_OWNER_EMAIL:joe_user`
 - Notatie: `output.<outputKey>:<outputValue>`
 Voorbeeld: `output.BuildNo:29`
 - Notatie: `statusMessage:<value>`

Voorbeeld: **statusMessage:Execution failed**

- De status is afhankelijk van de zoekpagina.
 - Voor uitvoeringen zijn mogelijke waarden onder meer: voltooid, mislukt, rollback_failed of geannuleerd.
 - Mogelijke statuswaarden voor pijplijnen zijn onder meer: ingeschakeld, uitgeschakeld of vrijgegeven.
 - Mogelijke statuswaarden voor triggers zijn onder meer: ingeschakeld of uitgeschakeld.
- Uitgevoerd, gemaakt of bijgewerkt door mij verwijst naar mij, de aangemelde gebruiker.

De zoekopdracht wordt weergegeven in de rechterbovenhoek van elke geldige pagina. Wanneer u begint te typen in het lege veld voor de zoekopdracht, kent Code Stream de context van de pagina en stelt de opties voor de zoekopdracht voor.

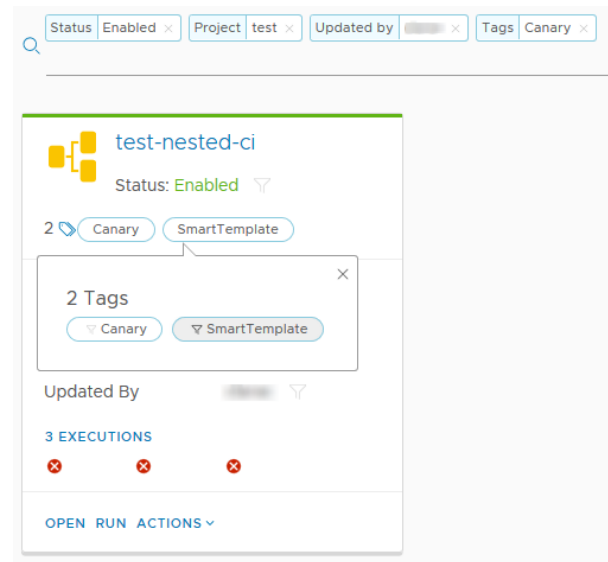
Methoden die u kunt gebruiken om te zoeken	Hoe u dit invoert
<p>Typ een deel van de zoekparameter.</p> <p>Om een statusfilter toe te voegen waarin alle ingeschakelde pijplijnen worden vermeld, typt u bijvoorbeeld inge.</p>	
<p>Voeg een filter toe om het aantal gevonden items te reduceren.</p> <p>Typ bijvoorbeeld Tes om een naamfilter toe te voegen. Het filter werkt als een EN met het bestaande filter Status:uitgeschakeld om alleen de gedeactiveerde pijplijnen met Tes in de naam weer te geven.</p> <p>Wanneer u een ander filter toevoegt, worden de overige opties weergegeven: Naam, Beschrijving, Tags, Link, Project en Bijgewerkt door.</p>	

Methoden die u kunt gebruiken om te zoeken**Hoe u dit invoert**

Als u het aantal weergegeven items wilt verminderen, klikt u op het filterpictogram bij eigenschappen van een pijplijn of een pijplijnuitvoering.

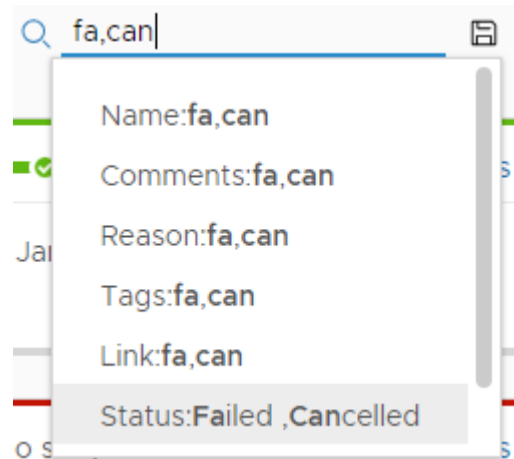
- Voor pijplijnen hebben **Status**, **Tags**, **Project** en **Bijgewerkt door** elk een filterpictogram.
- Voor uitvoeringen hebben **Tags**, **Uitgevoerd door** en **Statusbericht** elk een filterpictogram.

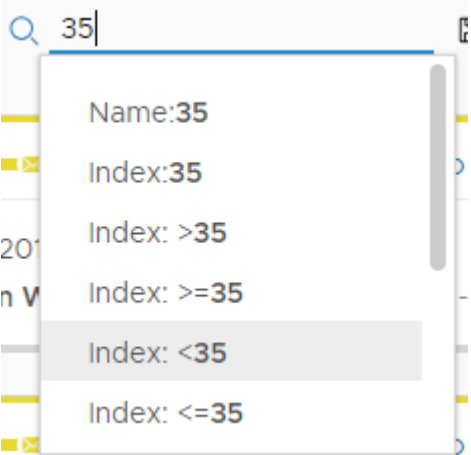
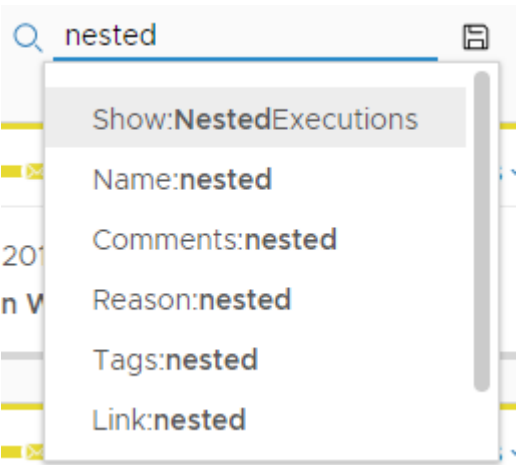
Klik bijvoorbeeld op het pictogram op de pijplijnkaart om het filter voor het **SmartTemplate**-label toe te voegen aan de bestaande filters voor: **Status: ingeschakeld**, **Project: test**, **Bijgewerkt door: gebruiker** en **Tags: Canary**.



Gebruik een komma als scheidingsteken om alle items in twee uitvoeringstoestanden op te nemen.

Typ bijvoorbeeld **mis1, geann** om een statusfilter te maken dat werkt als een **OF** om alle mislukte of geannuleerde uitvoeringen weer te geven.

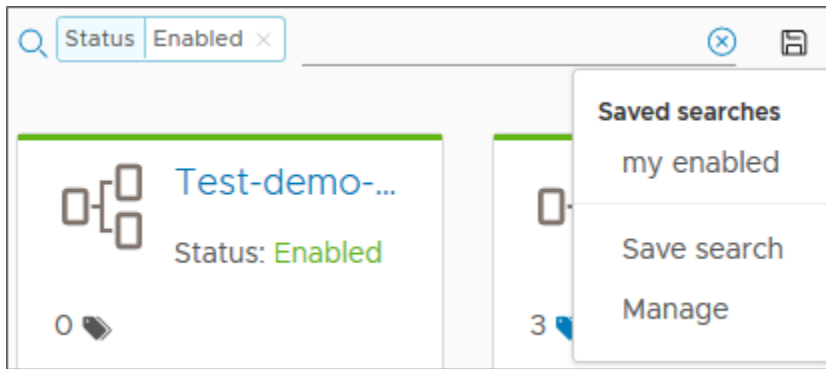


Methoden die u kunt gebruiken om te zoeken	Hoe u dit invoert
<p>Typ een nummer om alle items in een indexbereik op te nemen.</p> <p>Typ bijvoorbeeld 35 en selecteer < om alle uitvoeringen met een indexnummer lager dan 35 weer te geven.</p>	
<p>Pijplijnen die als taken zijn gemodelleerd, worden geneste uitvoeringen en worden standaard niet weergegeven met alle uitvoeringen.</p> <p>Als u geneste uitvoeringen wilt weergeven, typt u genest en selecteert u het filter Weergeven.</p>	

Hoe sla ik een favoriete zoekopdracht op?

U kunt favoriete zoekopdrachten opslaan om op elke pagina te gebruiken door op het schijfpictogram naast het zoekgebied te klikken.

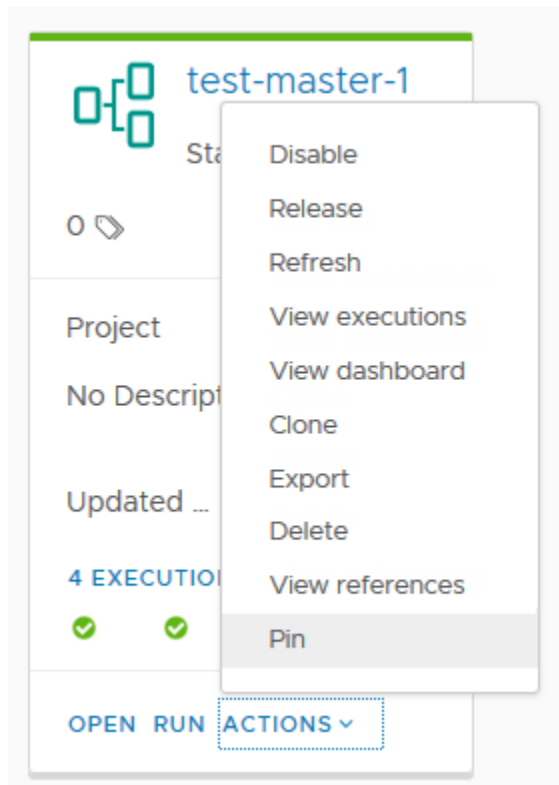
- U slaat een zoekopdracht op door de parameters voor de zoekopdracht in te typen en op het pictogram te klikken om de zoekopdracht een naam te geven zoals **mijn ingeschakeld**.
- Nadat u een zoekopdracht heeft opgeslagen, klikt u op het pictogram om de zoekopdracht te openen. U kunt ook **Beheren** selecteren om de zoekopdracht te hernoemen, verwijderen of verplaatsen in de lijst met opgeslagen zoekopdrachten.



Zoekopdrachten zijn gekoppeld aan uw gebruikersnaam en worden alleen weergegeven op de pagina's waarop de zoekopdracht van toepassing is. Als u bijvoorbeeld een zoekopdracht heeft opgeslagen met de naam **mijn ingeschakeld** voor **status: ingeschakeld** op de pagina Pijplijnen, dan is de zoekopdracht **mijn ingeschakeld** niet beschikbaar op de pagina Gerrit-triggers, zelfs als **status: ingeschakeld** een geldige zoekopdracht voor een trigger is.

Kan ik een favoriete pijplijn opslaan

Als u een favoriete pijplijn of favoriet dashboard hebt, kunt u deze of dit vastmaken zodat de pijplijn of het dashboard altijd bovenaan uw pagina Pijplijnen of Dashboards wordt weergegeven. Op de pijplijnkaart klikt u op **Acties > Vastmaken**.



Meer resources voor Code Stream-beheerders en -ontwikkelaars

Als Code Stream-beheerder of -ontwikkelaar kunt u meer te weten komen over Code Stream.

Tabel 9-1. Meer resources voor beheerders

Voor meer informatie over...	Zie deze resources...
<p>Andere manieren waarop beheerders Code Stream kunnen gebruiken:</p> <ul style="list-style-type: none"> ■ Het configureren van pijplijnen om het testen en vrijgeven van systeemeigen applicaties in de cloud te automatiseren. ■ Het automatiseren en testen van broncode van ontwikkelaars, via testen, naar productie. ■ Configureer pijplijnen voor ontwikkelaars om wijzigingen te testen voordat ze deze doorvoeren naar de hoofdtekst. ■ Het volgen van belangrijke pijplijnmetriekeken. 	<p>Code Stream</p> <ul style="list-style-type: none"> ■ Documentatie voor vRealize Automation ■ Productwebsite voor vRealize Automation <p>VMware Hands On</p> <ul style="list-style-type: none"> ■ Gebruik de vRealize Automation-gemeenschap. ■ Gebruik de VMware Learning Zone. ■ Zoek in de VMware-blogs. ■ Probeer de VMware Hands On Labs.

Tabel 9-2. Meer resources voor ontwikkelaars

Voor meer informatie over...	Zie deze resources...
<p>Andere manieren waarop ontwikkelaars Code Stream kunnen gebruiken:</p> <ul style="list-style-type: none"> ■ Gebruik openbare en persoonlijke registerimages om omgevingen te bouwen voor nieuwe applicaties of services. ■ Stel ontwikkelingsomgevingen in zodat u takken vanaf de nieuwste stabiele build kunt maken. ■ Werk de ontwikkelingsomgevingen bij met de nieuwste codewijzigingen en artefacten. ■ Test niet-doorgevoerde codewijzigingen tegen de laatste stabiele builds van andere afhankelijke services. ■ Ontvang een melding wanneer een wijziging die is doorgevoerd in een primaire CI/CD-pijplijn andere services verstoort. 	<p>Code Stream</p> <ul style="list-style-type: none"> ■ Documentatie voor vRealize Automation ■ Productwebsite voor vRealize Automation <p>VMware Hands On</p> <ul style="list-style-type: none"> ■ Gebruik de vRealize Automation-gemeenschap. ■ Gebruik de VMware Learning Zone. ■ Zoek in de VMware-blogs. ■ Probeer de VMware Hands On Labs.