

Verwenden und Verwalten von vRealize Automation Code Stream

14. Dezember 2022

vRealize Automation 8.3

Die aktuellste technische Dokumentation finden Sie auf der VMware-Website unter:

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

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

VMware Global, Inc.
Zweigniederlassung Deutschland
Willy-Brandt-Platz 2
81829 München
Germany
Tel.: +49 (0) 89 3706 17 000
Fax: +49 (0) 89 3706 17 333
www.vmware.com/de

Copyright © 2022 VMware, Inc. Alle Rechte vorbehalten. [Urheberrechts- und Markenhinweise](#).

Inhalt

| | | |
|----------|--|------------|
| 1 | Definition und Funktionsweise von vRealize Automation Code Stream | 5 |
| 2 | Einrichten zum Modellieren des Versionsprozesses | 11 |
| | Vorgehensweise zum Hinzufügen eines Projekts | 15 |
| | Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen | 16 |
| | Definition von Benutzervorgängen und Genehmigungen | 25 |
| 3 | Erstellen und Verwenden von Pipelines | 28 |
| | Vorgehensweise zum Ausführen einer Pipeline und Anzeigen von Ergebnissen | 32 |
| | Verfügbare Aufgabentypen | 37 |
| | Vorgehensweise zum Verwenden von Variablenbindungen in Pipelines | 43 |
| | Vorgehensweise zum Verwenden von Variablenbindungen in einer Bedingungsangabe zum Ausführen oder Anhalten einer Pipeline | 53 |
| | Welche Variablen und Ausdrücke kann ich bei der Bindung von Pipelineaufgaben verwenden? | 56 |
| | Wie sende ich Benachrichtigungen über meine Pipeline? | 76 |
| | Vorgehensweise zum Erstellen eines JIRA-Tickets bei einer fehlgeschlagenen Pipeline-Aufgabe | 79 |
| | Vorgehensweise zum Rollback meiner Bereitstellung | 82 |
| 4 | Planen des nativen Aufbaus, der Integration und der Bereitstellung Ihres Codes | 89 |
| | Planen eines nativen CICD-Builds vor der Verwendung der intelligenten Pipeline-Vorlage | 89 |
| | Planen eines nativen CI-Builds vor der Verwendung der intelligenten Pipeline-Vorlage | 94 |
| | Planen eines nativen CD-Builds vor der Verwendung der intelligenten Pipeline-Vorlage | 95 |
| | Planen eines nativen CICD-Builds vor dem manuellen Hinzufügen von Aufgaben | 97 |
| | Planen für ein Rollback | 102 |
| 5 | Lernprogramme | 105 |
| | Vorgehensweise zur kontinuierlichen Integration von Code aus einem GitHub- oder GitLab-Repository in eine Pipeline | 106 |
| | Vorgehensweise zum Automatisieren der Version einer Anwendung, die von einer YAML-Cloud-Vorlage bereitgestellt wird | 112 |
| | Vorgehensweise zum Automatisieren der Freigabe einer Anwendung in einem Kubernetes-Cluster | 120 |
| | Vorgehensweise zum Bereitstellen meiner Anwendung für meine Blau/Grün-Bereitstellung | 128 |
| | Vorgehensweise zum Integrieren eigener Build-, Test- und Bereitstellungstools | 133 |
| | Vorgehensweise zum Verwenden der Ressourceneigenschaften einer Cloud-Vorlagenangabe in der nächsten Aufgabe | 144 |

Vorgehensweise zum Verwenden einer REST API für die Integration in andere Anwendungen
149

6 Verbinden mit Endpoints 154

Definition von Endpoints 154

Vorgehensweise zum Integrieren in Jenkins 156

Vorgehensweise zum Integrieren in Git 161

Vorgehensweise zum Integrieren in Gerrit 165

Vorgehensweise zum Integrieren in vRealize Orchestrator 169

7 Auslösen von Pipelines 175

Vorgehensweise zum Verwenden des Docker-Auslösers zur Ausführung einer kontinuierlichen
Bereitstellungs-Pipeline 175

Vorgehensweise zum Verwenden des Git-Auslösers zum Ausführen einer Pipeline 184

Vorgehensweise zum Verwenden des Gerrit-Auslösers zum Ausführen einer Pipeline 192

8 Überwachen von Pipelines 200

Überblick über das Pipeline-Dashboard 200

Vorgehensweise zum Verwenden benutzerdefinierter Dashboards zum Verfolgen von wichtigen
Leistungsindikatoren 204

9 Weitere Informationen 207

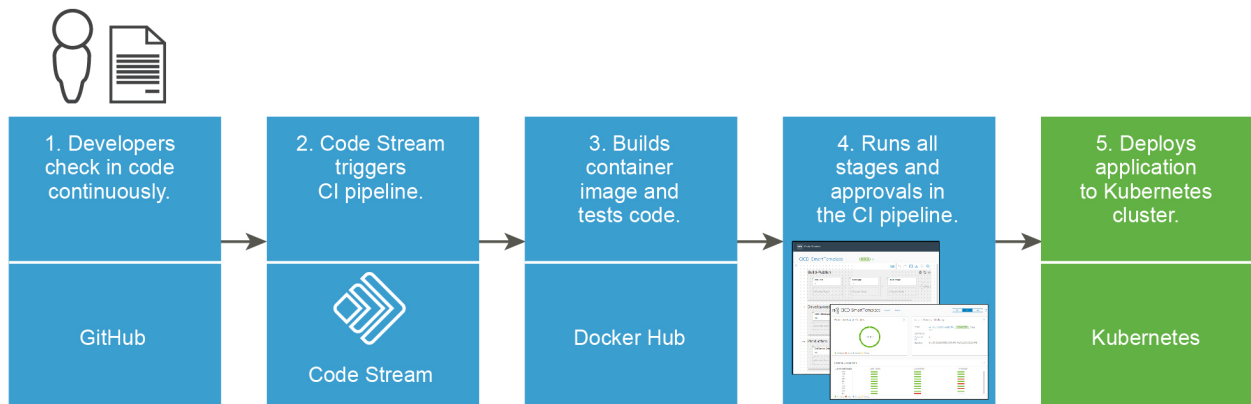
Definition der Suchfunktion 207

Weitere Ressourcen für Administratoren und Entwickler 212

Definition und Funktionsweise von vRealize Automation Code Stream

1

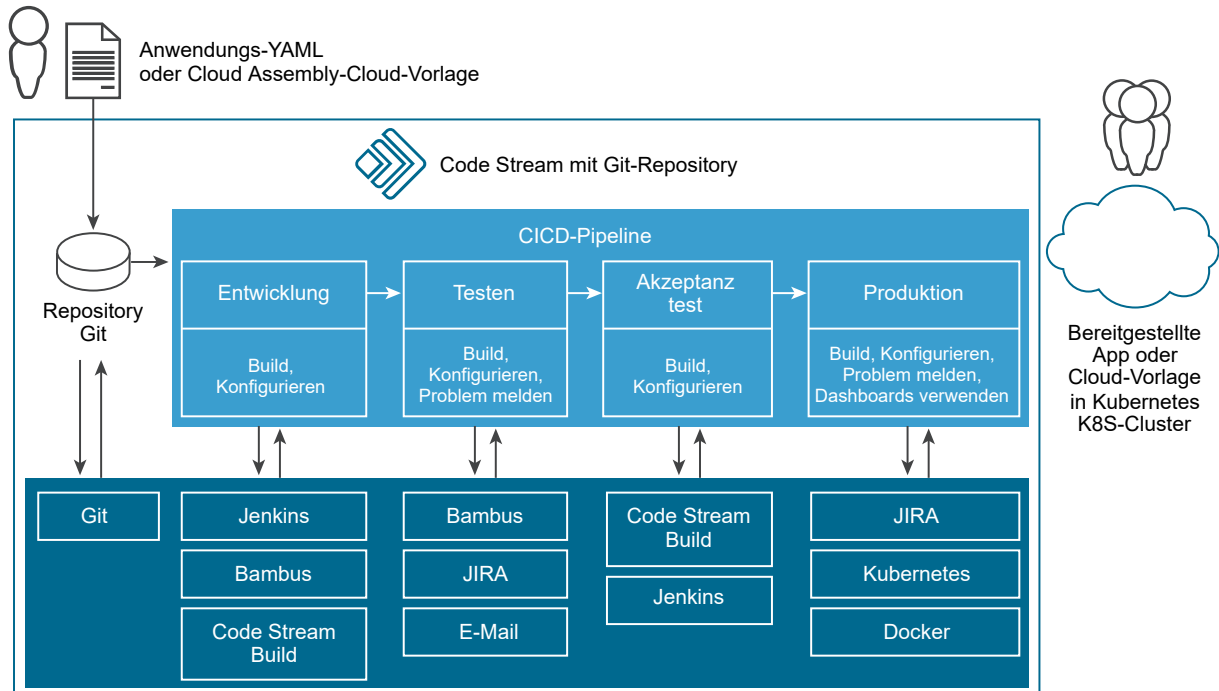
vRealize Automation Code Stream™ ist ein Tool für die kontinuierliche Integration und die kontinuierliche Bereitstellung (Continuous Integration and Continuous Delivery, CICD). Durch das Erstellen von Pipelines, die den Softwarefreigabeprozess in Ihrem DevOps-Lebenszyklus modellieren, erstellen Sie die Codeinfrastruktur, mit der Ihre Software schnell und kontinuierlich bereitgestellt wird.



Wenn Sie mit vRealize Automation Code Stream Ihre Software bereitstellen, integrieren Sie zwei der wichtigsten Teile Ihres DevOps-Lebenszyklus: den Versionsprozess und die Entwicklertools. Nach der anfänglichen Einrichtung, die vRealize Automation Code Stream in Ihre vorhandenen Entwicklungstools integriert, automatisieren die Pipelines Ihren gesamten DevOps-Lebenszyklus.

Ab vRealize Automation 8.2 werden Blueprints als VMware Cloud Templates bezeichnet.

Sie erstellen eine Pipeline, die Ihre Software erstellt, testet und freigibt. vRealize Automation Code Stream verwendet diese Pipeline, um Ihre Software vom Quellcode-Repository über Tests bis hin zur Produktion weiter zu entwickeln.



Weitere Informationen zum Planen der Pipelines für kontinuierliche Integration und kontinuierliche Bereitstellung finden Sie unter [Kapitel 4 Planen eines nativen Builds, der Integration und Bereitstellung von Code in vRealize Automation Code Stream](#).

Wie vRealize Automation Code Stream-Administratoren vRealize Automation Code Stream verwenden

Als Administrator erstellen Sie Endpoints und stellen sicher, dass funktionierende Instanzen für Entwickler verfügbar sind. Sie können Pipelines erstellen, auslösen und verwalten und vieles mehr. Sie verfügen über die Rolle `Administrator`, die in [Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream](#) beschrieben wird.

Tabelle 1-1. Wie vRealize Automation Code Stream-Administratoren Entwickler unterstützen

| Zur Unterstützung der Entwickler ... | Können Sie Folgendes tun ... |
|---|--|
| Bereitstellen und Verwalten von Umgebungen. | <p>Erstellen Sie Umgebungen, in denen Entwickler ihren Code testen und bereitstellen können.</p> <ul style="list-style-type: none"> ■ Verfolgen Sie den Status und senden Sie E-Mail-Benachrichtigungen. ■ Sorgen Sie dafür, dass Ihre Entwickler durchgängig produktiv sein können. Stellen Sie dazu sicher, dass ihre Umgebungen kontinuierlich funktionieren. <p>Weitere Informationen finden Sie unter Weitere Ressourcen für vRealize Automation Code Stream-Administratoren und -Entwickler.</p> <p>Siehe auch Kapitel 5 Lernprogramme für die Verwendung von vRealize Automation Code Stream.</p> |
| Stellen Sie Endpoints bereit. | Stellen Sie sicher, dass die Entwickler über funktionierende Endpoint-Instanzen verfügen, um die Verbindung zu ihren Pipelines herstellen zu können. |

Tabelle 1-1. Wie vRealize Automation Code Stream-Administratoren Entwickler unterstützen (Fortsetzung)

| Zur Unterstützung der Entwickler ... | Können Sie Folgendes tun ... |
|---|---|
| Bereitstellen von Integrationen mit anderen Diensten. | Stellen Sie sicher, dass die Integration in anderen Dienste funktioniert. Weitere Informationen finden Sie in der Dokumentation zu vRealize Automation . |
| Erstellen von Pipelines. | Erstellen Sie Pipelines, die Versionsprozesse modellieren. Weitere Informationen finden Sie unter Kapitel 3 Erstellen und Verwenden von Pipelines in vRealize Automation Code Stream . |
| Auslösen von Pipelines. | <p>Stellen Sie sicher, dass Pipelines ausgeführt werden, wenn Ereignisse auftreten.</p> <ul style="list-style-type: none"> ■ Verwenden Sie den Docker-Auslöser, um eine eigenständige Pipeline für die kontinuierliche Bereitstellung (Continuous Delivery, CD) auszulösen, wenn ein Build-Artefakt erstellt oder aktualisiert wird. ■ Um eine Pipeline auszulösen, wenn ein Entwickler Änderungen an seinem Code vornimmt, verwenden Sie den Git-Auslöser. ■ Um eine Pipeline auszulösen, wenn Entwickler Code überprüfen, Elemente zusammenführen und weitere Aktivitäten ausführen, verwenden Sie den Gerrit-Auslöser. ■ Verwenden Sie den Docker-Auslöser, um eine eigenständige Pipeline für die kontinuierliche Bereitstellung (Continuous Delivery, CD) auszuführen, wenn ein Build-Artefakt erstellt oder aktualisiert wird. <p>Weitere Informationen finden Sie unter Kapitel 7 Auslösen von Pipelines in vRealize Automation Code Stream.</p> |
| Verwalten von Pipelines und Genehmigungen. | <p>Seien Sie immer aktuell über Ihre Pipelines informiert.</p> <ul style="list-style-type: none"> ■ Zeigen Sie den Pipelinestatus an und sehen Sie, wer die Pipelines ausgeführt hat. ■ Zeigen Sie Genehmigungen für Pipeline-Ausführungen an und verwalten Sie Genehmigungen für aktive und inaktive Pipeline-Ausführungen. <p>Weitere Informationen finden Sie unter Definition der Benutzervorgänge und Genehmigungen in vRealize Automation Code Stream.</p> <p>Siehe auch Vorgehensweise zum Verwenden benutzerdefinierter Dashboards zum Verfolgen von wichtigen Leistungsindikatoren für meine Pipeline in vRealize Automation Code Stream.</p> |

Tabelle 1-1. Wie vRealize Automation Code Stream-Administratoren Entwickler unterstützen (Fortsetzung)

| Zur Unterstützung der Entwickler ... | Können Sie Folgendes tun ... |
|--------------------------------------|--|
| Überwachen von Entwicklerumgebungen. | <p>Erstellen Sie benutzerdefinierte Dashboards, die den Pipelinestatus, Trends, Metriken und wichtige Indikatoren überwachen. Verwenden Sie die benutzerdefinierten Dashboards, um Pipelines zu überwachen, die in Entwicklerumgebungen erfolgreich ausgeführt werden oder fehlschlagen. Sie können ebenfalls nicht ausreichend genutzte Ressourcen identifizieren und melden und Ressourcen freigeben.</p> <p>Außerdem können Sie sehen:</p> <ul style="list-style-type: none"> ■ Wie lange eine Pipeline ausgeführt wurde, bevor sie erfolgreich war. ■ Wie lange eine Pipeline auf die Genehmigung gewartet hat und ob der Benutzer benachrichtigt wurde, der sie genehmigen muss. ■ Phasen und Aufgaben, die am häufigsten fehlschlagen. ■ Phasen und Aufgaben, die die meiste Zeit für die Ausführung benötigen. ■ Versionen, an denen Entwicklungsteams arbeiten. ■ Anwendungen, die erfolgreich bereitgestellt und freigegeben wurden. <p>Weitere Informationen finden Sie unter Kapitel 8 Überwachen von Pipelines in vRealize Automation Code Stream.</p> |
| Beheben von Problemen. | <p>Suche und Behebung von Pipeline-Fehlern in Entwicklungsumgebungen.</p> <ul style="list-style-type: none"> ■ Identifizieren und beheben Sie Probleme in Umgebungen mit kontinuierlicher Integration und Bereitstellung (Continuous Integration and Continuous Delivery, CICD). ■ Verwenden Sie die Pipeline-Dashboards und erstellen Sie benutzerdefinierte Dashboards, um weitere Informationen anzuzeigen. Weitere Informationen hierzu finden Sie unter Kapitel 8 Überwachen von Pipelines in vRealize Automation Code Stream. <p>Siehe auch Kapitel 2 Einrichten von vRealize Automation Code Stream zum Modellieren des Freigabeprozesses.</p> |

vRealize Automation Code Stream ist Teil von vRealize Automation. vRealize Automation Code Stream lässt sich integrieren in:

- Verwenden Sie vRealize Automation Cloud Assembly zum Bereitstellen von Cloud-Vorlagen.
- Verwenden Sie vRealize Automation Service Broker zum Abrufen von Cloud-Vorlagen aus dem Katalog.

Weitere Informationen finden Sie in der [Dokumentation zu VMware vRealize Automation](#).

Vorgehensweise bei der Verwendung von vRealize Automation Code Stream durch Entwickler

Als Entwickler verwenden Sie vRealize Automation Code Stream zum Erstellen und Ausführen von Pipelines und zum Überwachen der Pipeline-Aktivitäten auf den Dashboards. Sie verfügen über die Rolle `User`, die in [Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream](#) beschrieben wird.

Nachdem Sie eine Pipeline ausgeführt haben, möchten Sie Folgendes wissen:

- Hat mein Code alle Phasen der Pipeline erfolgreich durchlaufen? Sehen Sie sich die Ergebnisse in **Ausführungen** an.
- Was soll ich tun, wenn die Pipeline fehlgeschlagen ist, und wie stelle ich fest, was den Fehler verursacht hat? In **Dashboards** können Sie sich die Fehler ansehen, die am häufigsten aufgetreten sind.

Tabelle 1-2. Entwickler, die vRealize Automation Code Stream verwenden

| Integrieren und Freigeben von Code... | gehen Sie wie folgt vor ... |
|---|---|
| Erstellen von Pipelines. | Testen Sie Ihren Code und stellen Sie ihn bereit. Aktualisieren Sie Ihren Code, wenn eine Pipeline fehlschlägt. |
| Verbinden Ihrer Pipeline mit Endpoints. | Verbinden Sie die Aufgaben in Ihrer Pipeline mit Endpoints, zum Beispiel einem GitHub-Repository. |
| Ausführen von Pipelines. | Fügen Sie eine Genehmigungsaufgabe für den Benutzervorgang hinzu, damit ein anderer Benutzer Ihre Pipeline an bestimmten Punkten genehmigen kann. |
| Anzeigen von Dashboards. | Zeigen Sie die Ergebnisse im Dashboard „Pipeline“ an. Sie können Trends, Verlauf, Fehler und vieles mehr anzeigen. |

Weitere Informationen zu den ersten Schritten finden Sie unter [Erste Schritte mit VMware Code Stream](#).

Weitere Dokumentation finden Sie im produktinternen Hilfe-Bereich

Wenn Sie die hier benötigten Informationen nicht finden, erhalten Sie weitere Hilfe im Produkt.



- Klicken Sie auf die Wegweiser und QuickInfos in der Benutzeroberfläche, um die kontextspezifischen Informationen, die Sie benötigen, am geeigneten Ort und zur geeigneten Zeit zu erhalten.
- Öffnen Sie den Bereich „Produktinterner Support“ und lesen Sie die Themen, die für die aktive Seite der Benutzeroberfläche angezeigt werden. Sie können auch im Bereich suchen, um Antworten auf Fragen zu erhalten.

Weitere Informationen zu Webhooks

Sie können mehrere Webhooks für verschiedene Zweige erstellen, indem Sie denselben Git-Endpoint verwenden und verschiedene Werte für den Namen des Zweigs auf der Konfigurationsseite des Webhooks bereitstellen. Zum Erstellen eines weiteren Webhooks für einen anderen Zweig im selben Git-Repository muss der Git-Endpoint nicht mehrmals für mehrere

Zweige geklont werden. Stattdessen geben Sie den Namen des Zweigs im Webhook an, wodurch Sie den Git-Endpoint wiederverwenden können. Wenn der Zweig im Git-Webhook mit dem Zweig im Endpoint übereinstimmt, müssen Sie den Namen des Zweigs nicht auf der Seite des Git-Webhooks angeben.

Einrichten von vRealize Automation Code Stream zum Modellieren des Freigabeprozesses

2

Zum Modellieren des Freigabeprozesses erstellen Sie eine Pipeline mit den Phasen, Aufgaben und Genehmigungen, die in der Regel zum Freigeben der Software verwendet werden. vRealize Automation Code Stream automatisiert dann den Prozess, mit dem der Code erstellt, getestet, genehmigt und bereitgestellt wird.

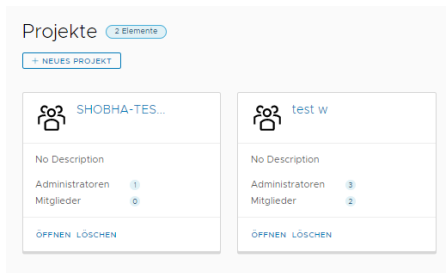
Da nun alles für die Modellierung des Softwarefreigabeprozesses vorbereitet ist, finden Sie hier Informationen zur Vorgehensweise in vRealize Automation Code Stream.

Voraussetzungen

- Prüfen Sie, ob bereits Endpoints verfügbar sind. In vRealize Automation Code Stream klicken Sie auf **Endpoints**.
- Informationen über native Möglichkeiten zum Erstellen und Bereitstellen Ihres Codes. Weitere Informationen hierzu finden Sie unter [Kapitel 4 Planen eines nativen Builds, der Integration und Bereitstellung von Code in vRealize Automation Code Stream](#).
- Legen Sie fest, ob bestimmte in der Pipeline zu verwendende Ressourcen als eingeschränkt markiert werden müssen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream](#).
- Wenn Sie anstelle der Benutzerrolle über die Entwickler- oder Betrachter-Rolle verfügen, geben Sie den Administrator für die vRealize Automation Code Stream-Instanz an.

Verfahren

- 1 Schauen Sie sich die verfügbaren Projekte in vRealize Automation Code Stream an und wählen Sie ein für Ihre Zwecke geeignetes Projekt aus.
 - Wenn keine Projekte aufgeführt sind, bitten Sie einen vRealize Automation Code Stream-Administrator, ein Projekt zu erstellen und Sie zum Mitglied des Projekts zu machen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).
 - Falls Sie kein Mitglied von einem der aufgeführten Projekte sind, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen.



- 2 Fügen Sie alle neuen Endpoints hinzu, die Sie für Ihre Pipeline brauchen.

Sie benötigen beispielsweise Git, Jenkins, Code Stream Build, Kubernetes und Jira.

- 3 Erstellen Sie Variablen, um Werte in Ihren Pipeline-Aufgaben wiederverwenden zu können.

Nutzen Sie eingeschränkte Variablen, um die in Ihren Pipelines verwendeten Ressourcen, z. B. eine Hostmaschine, einzuschränken. Sie können die Ausführung der Pipeline so lange einschränken, bis sie von einem anderen Benutzer ausdrücklich genehmigt wird.

Administratoren können geheime Variablen und eingeschränkte Variablen erstellen. Benutzer können geheime Variablen erstellen.

Sie können eine Variable beliebig oft über mehrere Pipelines hinweg wiederverwenden.

Eine Variable, die eine Hostmaschine angibt, kann beispielsweise als `HostIPAddress` definiert werden. Zur Verwendung der Variable in einer Pipeline-Aufgabe geben Sie dann `${var.HostIPAddress}` ein.

| Projekt | Name | Typ | Wert |
|--|---------------|------------|-------|
| 0709-AWS-w2-Test-Instanz 中CE6關停B酒UBau'h | 2000000000000 | SECRET | ***** |
| 0709-AWS-w2-Test-Instanz 中CE6關停B酒UBau'h | 20 | RESTRICTED | ***** |
| 0709-AWS-w2-Test-Instanz 中CE6關停B酒UBau'h | 2 | SECRET | ***** |

- 4 Kennzeichnen Sie als Administrator alle Endpoints und Variablen, die für Ihr Unternehmen unerlässlich sind, als eingeschränkte Ressourcen.

Wenn ein Benutzer, der kein Administrator ist, versucht, eine Pipeline auszuführen, die eine eingeschränkte Ressource enthält, stoppt die Pipeline bei der Aufgabe, die die eingeschränkte Ressource verwendet. Anschließend muss ein Administrator die Ausführung der Pipeline fortsetzen.

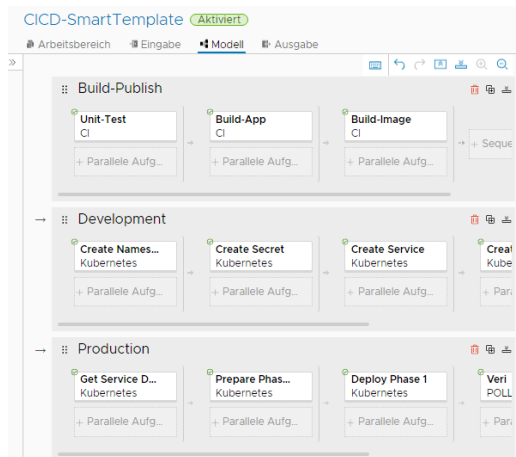
5 Planen Sie die Erstellungsstrategie für die native CICD-, CI- oder CD-Pipeline.

Planen Sie Ihre Erstellungsstrategie vor dem Einrichten einer Pipeline, die Ihren Code kontinuierlich integriert (CI, Kontinuierliche Integration) und kontinuierlich bereitstellt (CD, Kontinuierliche Bereitstellung). Mit dem Build-Plan können Sie ermitteln, was vRealize Automation Code Stream benötigt, damit es Ihren Code nativ erstellen, integrieren, testen und bereitstellen kann.

| Erstellung eines nativen vRealize Automation Code Stream-Builds | Folgende Erstellungsstrategie wird benötigt |
|---|---|
| Verwenden Sie eine der intelligenten Pipeline-Vorlagen. | <ul style="list-style-type: none"> ■ Erstellt alle Phasen und Aufgaben für Sie. ■ Klont das Quell-Repository. ■ Erstellt und testet den Code. ■ Fasst den Code zur Bereitstellung in Containern zusammen. ■ Befüllt die Schritte der Pipeline-Aufgabe auf Basis Ihrer Auswahl. |
| Fügen Sie Phasen und Aufgaben manuell hinzu. | Sie fügen Phasen und Aufgaben hinzu und geben die Informationen ein, mit denen sie befüllt werden. |

6 Erstellen Sie Ihre Pipeline mithilfe einer intelligenten Pipeline-Vorlage oder fügen Sie der Pipeline manuell Phasen und Aufgaben hinzu.

Markieren Sie anschließend alle Ressourcen als eingeschränkt. Fügen Sie gegebenenfalls Genehmigungen hinzu. Wenden Sie alle regulären, eingeschränkten oder geheimen Variablen an. Fügen Sie alle Bindungen zwischen Aufgaben hinzu.



7 Validieren, aktivieren und führen Sie die Pipeline aus.

8 Zeigen Sie die Pipeline-Ausführungen an.

The screenshot shows the 'Ausführungen' (Executions) page with 468 elements. It lists several pipeline executions:

- Demo-Jenkins-K... #93**: COMPLETED, Phasen: 4/4, Von sestervil am 19. Jan. 2020, 16:17:14. Eingabe: 6d82d079a8b8921a...
- Demo-Jenkins-K... #13**: COMPLETED, Phasen: 4/4, Von sestervil am 19. Jan. 2020, 16:14:35. Eingabe: 6d82d079a8b8921a9...
- Demo-Jenkins-K... #92**: COMPLETED, Phasen: 4/4, Von sestervil am 19. Jan. 2020, 16:11:06. Eingabe: 8b3a29fdf...
- Demo-CICD-Simp#48**: FAILED, Phasen: 4/4, Von sestervil am 19. Jan. 2020, 16:09:20. Production.Deploy Phase 1: Failed script execution: Failed to execu...

9 Verwenden Sie zum Verfolgen des Status und der KPIs die Pipeline-Dashboards und erstellen Sie benutzerdefinierte Dashboards.

The screenshot shows the 'CICD-SmartTemplate' dashboard with the following components:

- Letzte erfolgreiche Ausführung**: Demo-Jenkins-K8s #109 COMPLETED vor einer Stunde. Ausgeführt von smasaru, Dauer 28 Minuten. Eingabe: 8b3a29fdf...
- Zahlen für Ausführ...**: Donut chart showing 'Gesamt: 106' with a legend for COMPLETED (green), FAILED (red), and WAITING (yellow).
- Ausführungsstatistik**: Metrics for the last 14 days:
 - MTTD: 1 Minute
 - MTTF: 3 Tage
 - MTBD: 47 Minuten
 - MTTR: -
- Zuletzt erfolgte Ausführungen**: Table of recent successful executions with columns for Ausführung #/Phas..., Dev, and status icons.

Ergebnisse

Sie haben eine Pipeline erstellt, die Sie im ausgewählten Projekt verwenden können.

Sie können auch die Pipeline-YAML für den Import und die Wiederverwendung in anderen Projekten exportieren.

Nächste Schritte

Informationen zu Anwendungsfällen, die Sie gegebenenfalls in Ihrer Umgebung anwenden möchten. Weitere Informationen hierzu finden Sie unter [Kapitel 5 Lernprogramme für die Verwendung von vRealize Automation Code Stream](#).

Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream

Sie erstellen ein Projekt und fügen ihm Administratoren und Mitglieder hinzu. Projektmitglieder können Funktionen wie das Erstellen einer Pipeline und das Hinzufügen eines Endpoints verwenden. Um ein Projekt für ein Entwicklungsteam zu erstellen, zu löschen oder zu aktualisieren, müssen Sie ein vRealize Automation Code Stream-Administrator sein.

Ein Projekt muss vorhanden sein. Erst dann können Sie eine Pipeline erstellen. Wenn Sie eine Pipeline erstellen, wählen Sie ein Projekt aus, das alle Ihre Pipeline-Informationen zusammenfasst. Definitionen für Endpoints und Variablen sind ebenfalls von einem bestehenden Projekt abhängig.

Voraussetzungen

- Stellen Sie sicher, dass Sie über die Benutzerrolle des vRealize Automation Code Stream-Administrators verfügen. Weitere Informationen hierzu finden Sie unter [Definition von Rollen in vRealize Automation Code Stream](#).

Wenn Sie nicht über die Rolle des vRealize Automation Code Stream-Administrators verfügen, aber in vRealize Automation Cloud Assembly ein Administrator sind, können Sie die vRealize Automation Cloud Assembly-Benutzeroberfläche verwenden, um Projekte zu erstellen, zu aktualisieren oder zu löschen. Siehe [Vorgehensweise zum Hinzufügen eines Projekts für das vRealize Automation Cloud Assembly-Entwicklungsteam](#)

- Wenn Sie Projekten Active Directory-Gruppen hinzufügen, müssen Sie Active Directory-Gruppen für Ihre Organisation konfiguriert haben. Weitere Informationen finden Sie unter [Vorgehensweise zum Aktivieren von Active Directory-Gruppen in vRealize Automation für Projekte](#). Wenn Sie versuchen, nicht synchronisierte Gruppen zu einem Projekt hinzuzufügen, sind sie nicht verfügbar.

Verfahren

- 1 Klicken Sie auf **Projekte** und dann auf **Neues Projekt**.
- 2 Geben Sie den Projektnamen ein.
- 3 Klicken Sie auf **Erstellen**.
- 4 Wählen Sie die Karte für das neu erstellte Projekt aus und klicken Sie auf **Öffnen**.
- 5 Klicken Sie auf die Registerkarte **Benutzer** und fügen Sie Benutzer hinzu und weisen Sie Rollen zu.
 - Der Projektadministrator kann Mitglieder hinzufügen.
 - Das Projektmitglied mit einer Dienstrolle kann Dienste verwenden.
 - Der Projekt-Viewer kann Projekte anzeigen, sie aber weder erstellen, aktualisieren noch löschen.

Weitere Informationen zu Projektrollen finden Sie unter [Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream](#).

6 Klicken Sie auf **Speichern**.

Nächste Schritte

Fügen Sie Endpoints und Pipelines hinzu, die das Projekt verwenden. Siehe [Kapitel 6 Verbinden von vRealize Automation Code Stream mit Endpoints](#) und [Kapitel 3 Erstellen und Verwenden von Pipelines in vRealize Automation Code Stream](#).

Nachdem Sie eine Pipeline erstellt haben, wird der Name des Projekts, in dem alle Ihre Pipeline-Informationen zusammengefasst sind, auf Pipeline-Karten und Pipeline-Ausführungskarten angezeigt.

Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream

vRealize Automation Code Stream bietet mehrere Möglichkeiten, um sicherzustellen, dass Benutzer über die entsprechende Autorisierung und Zustimmung verfügen, um mit Pipelines zu arbeiten, die Ihre Softwareanwendungen freigeben.

Jedes Mitglied in einem Team hat eine zugewiesene Rolle, die bestimmte Berechtigungen für Pipelines, Endpoints und Dashboards erteilt und die Möglichkeit bietet, Ressourcen als eingeschränkt zu markieren.

Mithilfe von Benutzervorgängen und Genehmigungen können Sie festlegen, wann eine Pipeline ausgeführt wird und für eine Genehmigung angehalten werden muss. Ihre Rolle legt fest, ob Sie eine Pipeline fortsetzen und Pipelines ausführen können, die eingeschränkte Endpoints oder Variablen enthalten.

Verwenden Sie geheime Variablen, um vertrauliche Informationen auszublenden und zu verschlüsseln. Verwenden Sie eingeschränkte Variablen für Zeichenfolgen, Kennwörter und URLs, die ausgeblendet und verschlüsselt sein müssen, sowie zur Einschränkung ihrer Nutzung in Ausführungen. Verwenden Sie beispielsweise eine geheime Variable für ein Kennwort oder eine URL. Sie können geheime und eingeschränkte Variablen in jeder Art von Aufgabe in Ihrer Pipeline verwenden.

Definition von Rollen in vRealize Automation Code Stream

Je nach Ihrer Rolle in vRealize Automation Code Stream können Sie bestimmte Aktionen ausführen und auf bestimmte Bereiche zugreifen. Beispiel: Sie können mit Ihrer Rolle unter Umständen Pipelines erstellen, aktualisieren und ausführen. Oder Sie verfügen nur über die Berechtigung zum Anzeigen von Pipelines.

Alle, ausgenommen eingeschränkter Aktionen bedeutet, dass dieser Rolle die Berechtigung zum Durchführen, Erstellen, Lesen, Aktualisieren und Löschen von Aktionen in Entitäten, ausgenommen eingeschränkter Variablen und Endpoints, erteilt wurde.

Tabelle 2-1. Zugriffsberechtigungen auf Dienst- und Projektebene in vRealize Automation Code Stream

| vRealize Automation Code Stream-Rollen | | | | | |
|---|---------------------------|--|--|--|--|
| Zugriffsebenen | Code Stream-Administrator | Code Stream-Entwickler | Code Stream-Executor | Code Stream-Betrachter | Code Stream-Benutzer |
| Zugriff auf Ebene des vRealize Automation Code Stream-Diensts | Alle Aktionen | Alle, ausgenommen eingeschränkter Aktionen | Ausführungsaktionen | Schreibgeschützt | Keine |
| Zugriff auf Projektebene: Projektadministrator | Alle Aktionen | Alle Aktionen | Alle Aktionen | Alle Aktionen | Alle Aktionen |
| Zugriff auf Projektebene: Projektmitglied | Alle Aktionen | Alle, ausgenommen eingeschränkter Aktionen | Alle, ausgenommen eingeschränkter Aktionen | Alle, ausgenommen eingeschränkter Aktionen | Alle, ausgenommen eingeschränkter Aktionen |
| Zugriff auf Projektebene: Projekt-Viewer | Alle Aktionen | Alle, ausgenommen eingeschränkter Aktionen | Ausführungsaktionen | Schreibgeschützt | Schreibgeschützt |

Benutzer mit der Rolle „Projektadministrator“ können alle Aktionen für Projekte ausführen, bei denen sie als Projektadministratoren fungieren.

Ein Projektadministrator kann Pipelines, Variablen, Endpoints, Dashboards und Auslöser erstellen, lesen, aktualisieren und löschen. Weiterhin kann er eine Pipeline mit eingeschränkten Endpoints oder Variablen starten, wenn sich die Ressourcen in dem Projekt befinden, in dem der Benutzer als Projektadministrator fungiert.

Benutzer, die über die Rolle „Dienst-Viewer“ verfügen, können alle für den Administrator verfügbaren Informationen anzeigen. Sie können keine Aktionen ausführen, es sei denn, der Administrator macht sie zu einem Projektadministrator oder Projektmitglied. Wenn der Benutzer zu einem Projekt gehört, verfügt er über die Berechtigungen, die mit dieser Rolle verbunden sind. Der Projekt-Viewer erweitert seine Berechtigungen nicht in der Weise, wie dies bei den Rollen „Projektadministrator“ oder „Projektmitglied“ der Fall ist. Diese Rolle verfügt für alle Projekte nur über Leserechte.

Wenn Sie über Leseberechtigungen in einem Projekt verfügen, können Sie weiterhin eingeschränkte Ressourcen anzeigen.

- Zum Anzeigen eingeschränkter Endpoints, für die auf der Endpoint-Karte ein Sperrsymbol angezeigt wird, klicken Sie auf **Konfigurieren > Endpoints**.
- Zum Anzeigen eingeschränkter oder geheimer Variablen, für die EINGESCHRÄNKT oder GEHEIM in der Spalte **Typ** angezeigt wird, klicken Sie auf **Konfigurieren > Variablen**.

Tabelle 2-2. Funktionen der vRealize Automation Code Stream-Dienstrolle

| UI-Kontext | Funktionen | Code Stream-Administratorrolle | Code Stream-Entwicklerrolle | Code Stream-Executor-Rolle | Code Stream-Viewer-Rolle | Code Stream-Benutzerrolle |
|---|--|--------------------------------|-----------------------------|----------------------------|--------------------------|---------------------------|
| Pipelines | | | | | | |
| | Pipelines anzeigen | Ja | Ja | Ja | Ja | |
| | Pipelines erstellen | Ja | Ja | | | |
| | Pipelines ausführen | Ja | Ja | Ja | | |
| | Pipelines ausführen, die eingeschränkte Endpoints oder Variablen enthalten | Ja | | | | |
| | Pipelines aktualisieren | Ja | Ja | | | |
| | Pipelines löschen | Ja | Ja | | | |
| Pipeline-Ausführungen | | | | | | |
| | Pipeline-Ausführungen anzeigen | Ja | Ja | Ja | Ja | |
| | Pipeline-Ausführungen fortsetzen, anhalten und abbrechen | Ja | Ja | Ja | | |
| | Pipelines fortsetzen, die für die Genehmigung auf eingeschränkten Ressourcen angehalten werden | Ja | | | | |
| Benutzerdefinierte Integrationen | | | | | | |
| | Benutzerdefinierte Integrationen erstellen | Ja | Ja | | | |
| | Benutzerdefinierte Integrationen lesen | Ja | Ja | Ja | Ja | |
| | Benutzerdefinierte Integration aktualisieren | Ja | Ja | | | |
| Endpoints | | | | | | |
| | Ausführungen anzeigen | Ja | Ja | Ja | Ja | |
| | Ausführungen erstellen | Ja | Ja | | | |
| | Ausführungen aktualisieren | Ja | Ja | | | |

Tabelle 2-2. Funktionen der vRealize Automation Code Stream-Dienstrolle (Fortsetzung)

| UI-Kontext | Funktionen | Code Stream-Administratorrolle | Code Stream-Entwicklerrolle | Code Stream-Executor-Rolle | Code Stream-Viewer-Rolle | Code Stream-Benutzerrolle |
|---|---|--------------------------------|-----------------------------|----------------------------|--------------------------|---------------------------|
| | Ausführungen löschen | Ja | Ja | | | |
| Ressourcen als eingeschränkt kennzeichnen | | | | | | |
| | Endpoint oder Variable als eingeschränkt kennzeichnen | Ja | | | | |
| Dashboards | | | | | | |
| | Dashboards anzeigen | Ja | Ja | Ja | Ja | |
| | Dashboards erstellen | Ja | Ja | | | |
| | Dashboards aktualisieren | Ja | Ja | | | |
| | Dashboards löschen | Ja | Ja | | | |

Benutzerdefinierte Rollen und Berechtigungen in vRealize Automation Code Stream

Sie können in vRealize Automation Cloud Assembly benutzerdefinierte Rollen erstellen, mit denen Berechtigungen auf Benutzer erweitert werden, die mit Pipelines arbeiten. Wenn Sie eine benutzerdefinierte Rolle für vRealize Automation Code Stream-Pipelines erstellen, wählen Sie eine oder mehrere **Pipeline**-Berechtigungen aus.

Wählen Sie die geringste Anzahl an **Pipeline**-Berechtigungen aus, die von Benutzern benötigt wird, denen diese benutzerdefinierte Rolle zugewiesen wird.

Wenn ein Benutzer einem Projekt zugewiesen wird und eine Rolle in diesem Projekt erhält und diesem Benutzer dann eine benutzerdefinierte Rolle zugeteilt wird, die eine oder mehrere **Pipeline**-Berechtigungen enthält, kann der Benutzer alle durch die Berechtigungen zugelassenen Aktionen durchführen. Der Benutzer kann beispielsweise eingeschränkte Variablen erstellen, eingeschränkte Pipelines verwalten, benutzerdefinierte Integrationen erstellen und verwalten usw.

Tabelle 2-3. Pipeline-Berechtigungen, die benutzerdefinierten Rollen zugewiesen werden können

| Pipeline-Berechtigung | Code Stream-Administrator | Code Stream-Entwickler | Code Stream-Executor | Code Stream-Betrachter | Code Stream-Benutzer | Projektadministrator | Projektmitglied | Projekt-Viewer |
|---|---------------------------|------------------------|----------------------|------------------------|----------------------|----------------------|-----------------|----------------|
| Pipelines verwalten | Ja | Ja | | | | Ja | Ja | |
| Eingeschränkte Pipelines verwalten | Ja | | | | | Ja | | |
| Benutzerdefinierte Integrationen verwalten | Ja | Ja | | | | | | |
| Pipelines ausführen | Ja | Ja | Ja | | | Ja | Ja | |
| Eingeschränkte Pipelines ausführen | Ja | | | | | Ja | | |
| Ausführungen verwalten | Ja | | | | | Ja | | |
| Lesen. Diese Berechtigung wird nicht angezeigt. | Ja | Ja | Ja | Ja | | Ja | Ja | Ja |

Tabelle 2-4. Vorgehensweise zum Verwenden von Pipeline-Berechtigungen mit benutzerdefinierten Rollen

| Berechtigung | Aktion |
|--|--|
| Pipelines verwalten | <ul style="list-style-type: none"> ■ Pipelines erstellen, aktualisieren, löschen klonen ■ Pipelines für VMware Service Broker freigeben und deren Freigabe aufheben. ■ Endpoints erstellen, aktualisieren und löschen. ■ Reguläre und geheime Variablen erstellen, aktualisieren und löschen. ■ Gerrit-Listener erstellen, klonen, aktualisieren und löschen. ■ Gerrit-Listener verbinden und deren Verbindung aufheben. ■ Gerrit-Auslöser erstellen, klonen, aktualisieren, löschen. ■ Git-Webhook erstellen, aktualisieren und löschen. ■ Docker-Webhook erstellen, aktualisieren und löschen. ■ Intelligente Pipeline-Vorlagen zum Erstellen von Vorlagen verwenden. ■ Pipelines aus YAML importieren und nach YAML exportieren. ■ Benutzerdefinierte Dashboards erstellen, aktualisieren und löschen. ■ Alle benutzerdefinierten Integrationen lesen. ■ Alle eingeschränkten Endpoints und Variablen lesen, zugehörige Werte können jedoch nicht angezeigt werden. |
| Eingeschränkte Pipelines verwalten | <ul style="list-style-type: none"> ■ Endpoints erstellen, aktualisieren und löschen. ■ Endpoints als eingeschränkt kennzeichnen, eingeschränkte Endpoints aktualisieren und Endpoints löschen. ■ Reguläre und geheime Variablen erstellen, aktualisieren und löschen. ■ Eingeschränkte Variablen erstellen, aktualisieren und löschen. ■ Alle Berechtigungen, die mit „Pipelines verwalten“ zur Verfügung stehen. |
| Benutzerdefinierte Integrationen verwalten | <ul style="list-style-type: none"> ■ Benutzerdefinierte Integrationen erstellen und aktualisieren. ■ Benutzerdefinierte Integrationen versionieren und freigeben. ■ Benutzerdefinierte Integrationsversionen löschen und als veraltet kennzeichnen. ■ Benutzerdefinierte Integrationen löschen. |
| Pipelines ausführen | <ul style="list-style-type: none"> ■ Pipelines ausführen. ■ Pipeline-Ausführungen anhalten, fortsetzen und abbrechen. ■ Pipeline-Ausführungen erneut ausführen. ■ Gerrit-Auslöserereignis fortsetzen, erneut ausführen und manuell auslösen. ■ Benutzervorgang genehmigen und Batchgenehmigungen von Benutzervorgängen durchführen. |

Tabelle 2-4. Vorgehensweise zum Verwenden von Pipeline-Berechtigungen mit benutzerdefinierten Rollen (Fortsetzung)

| Berechtigung | Aktion |
|---------------------------------------|--|
| Eingeschränkte Pipelines ausführen | <ul style="list-style-type: none"> ■ Pipelines ausführen. ■ Pipeline-Ausführungen anhalten, fortsetzen, abbrechen und löschen. ■ Pipeline-Ausführungen erneut ausführen. ■ Laufende Pipeline-Ausführung synchronisieren. ■ Löschen einer laufenden Pipeline-Ausführung erzwingen. ■ Gerrit-Auslöserereignis fortsetzen, erneut ausführen, löschen und manuell auslösen. ■ Eingeschränkte Elemente auflösen und die Pipeline-Ausführung fortsetzen. ■ Benutzerkontext wechseln und Pipeline-Ausführung nach der Genehmigung einer Benutzervorgangsaufgabe fortsetzen. ■ Alle Berechtigungen, die mit „Pipelines ausführen“ zur Verfügung stehen. |
| Ausführungen verwalten | <ul style="list-style-type: none"> ■ Pipelines ausführen. ■ Pipeline-Ausführungen anhalten, fortsetzen, abbrechen und löschen. ■ Pipeline-Ausführungen erneut ausführen. ■ Gerrit-Auslöserereignis fortsetzen, erneut ausführen, löschen und manuell auslösen. ■ Alle Berechtigungen, die mit „Pipelines ausführen“ zur Verfügung stehen. |

Benutzerdefinierte Rollen können Kombinationen von Berechtigungen enthalten. Diese Berechtigungen sind in Gruppen von Funktionen zusammengefasst, mit denen Benutzer Pipelines mit und ohne eingeschränkte Ressourcen verwalten oder ausführen können. Diese Berechtigungen repräsentieren alle Funktionen, die von jeder Rolle in vRealize Automation Code Stream ausgeführt werden können.

Wenn Sie beispielsweise eine benutzerdefinierte Rolle erstellen und die Berechtigung **Eingeschränkte Pipelines verwalten** einbeziehen, können Benutzer mit der Rolle „vRealize Automation Code Stream-Entwickler“ folgende Aktionen durchführen:

- Endpoints erstellen, aktualisieren und löschen.
- Endpoints als eingeschränkt kennzeichnen, eingeschränkte Endpoints aktualisieren und Endpoints löschen.
- Reguläre und geheime Variablen erstellen, aktualisieren und löschen.
- Eingeschränkte Variablen erstellen, aktualisieren und löschen.

Tabelle 2-5. Beispielskombinationen für Pipeline-Berechtigungen in benutzerdefinierten Rollen

| Anzahl der der benutzerdefinier ten Rolle zugewiesenen Berechtigungen | Beispiele für kombinierte Berechtigungen | Vorgehensweise zum Verwenden dieser Kombination |
|---|---|--|
| Eine Berechtigung | Pipelines ausführen | |
| Zwei Berechtigungen | Pipelines verwalten und Pipelines ausführen | |

Tabelle 2-5. Beispielskombinationen für Pipeline-Berechtigungen in benutzerdefinierten Rollen (Fortsetzung)

| Anzahl der der benutzerdefinierten Rolle zugewiesenen Berechtigungen | Beispiele für kombinierte Berechtigungen | Vorgehensweise zum Verwenden dieser Kombination |
|--|---|---|
| Drei Berechtigungen | Pipelines verwalten und Pipelines ausführen und Eingeschränkte Pipelines ausführen | |
| | Pipelines verwalten und Benutzerdefinierte Integrationen verwalten und Eingeschränkte Pipelines ausführen | Diese Kombination gilt unter Umständen für eine Rolle vom Typ „vRealize Automation Code Stream-Entwickler“, ist aber auf die Projekte beschränkt, zu denen der Benutzer gehört. |
| | Pipelines verwalten und Benutzerdefinierte Integrationen verwalten und Ausführungen verwalten | Diese Kombination gilt unter Umständen für einen vRealize Automation Code Stream-Administrator, ist aber auf die Projekte beschränkt, zu denen der Benutzer gehört. |
| | Pipelines verwalten, Eingeschränkte Pipelines verwalten und Benutzerdefinierte Integrationen verwalten | Mit dieser Kombination verfügt ein Benutzer über uneingeschränkte Berechtigungen und kann in vRealize Automation Code Stream alle Aktionen erstellen und löschen. |

Rolle „Administrator“

Als Administrator können Sie benutzerdefinierte Integrationen, Endpoints, Variablen, Auslöser, Pipelines und Dashboards erstellen.

Projekte ermöglichen Pipelines den Zugriff auf Infrastrukturressourcen. Administratoren erstellen Projekte, damit Benutzer Pipelines, Endpoints und Dashboards gruppieren können. Benutzer wählen dann das Projekt in ihren Pipelines aus. Jedes Projekt umfasst einen Administrator sowie Benutzer mit zugewiesenen Rollen.

Mit der Administratorrolle können Sie Endpoints und Variablen als eingeschränkte Ressourcen markieren sowie Pipelines ausführen, die eingeschränkte Ressourcen verwenden. Wenn ein Benutzer ohne Administratorrechte die Pipeline ausführt, die einen eingeschränkten Endpoint oder eine eingeschränkte Variable enthält, wird die Pipeline bei der Aufgabe angehalten, in der die eingeschränkte Variable verwendet wird. Die Pipeline muss dann von einem Administrator fortgesetzt werden.

Als Administrator können Sie auch verlangen, dass Pipelines in vRealize Automation Service Broker veröffentlicht werden.

Rolle „Entwickler“

Sie können mit Pipelines wie ein Administrator arbeiten, mit der Ausnahme, dass Sie nicht mit eingeschränkten Endpoints oder Variablen arbeiten können.

Wenn Sie eine Pipeline ausführen, die eingeschränkte Endpoints oder Variablen verwendet, wird die Pipeline nur bis zu der Aufgabe ausgeführt, die die eingeschränkte Ressource verwendet. Dann wird sie angehalten und ein vRealize Automation Code Stream- oder Projektadministrator muss die Pipeline fortsetzen.

Rolle „Benutzer“

Sie können auf vRealize Automation Code Stream zugreifen, haben aber keine Berechtigungen wie die anderen Rollen.

Rolle „Viewer“

Sie können dieselben Ressourcen anzeigen wie ein Administrator, beispielsweise Pipelines, Endpoints, Pipeline-Ausführungen, Dashboards, benutzerdefinierte Integrationen und Auslöser. Sie können diese Ressourcen jedoch weder erstellen, aktualisieren noch löschen. Zum Durchführen von Aktionen muss der Viewer-Rolle ebenfalls die Rolle „Projektadministrator“ oder „Projektmitglied“ zugewiesen werden.

Benutzer mit der Viewer-Rolle können Projekte anzeigen. Sie können auch eingeschränkte Endpoints und Variablen anzeigen. Entsprechende Detailinformationen stehen jedoch nicht zur Verfügung.

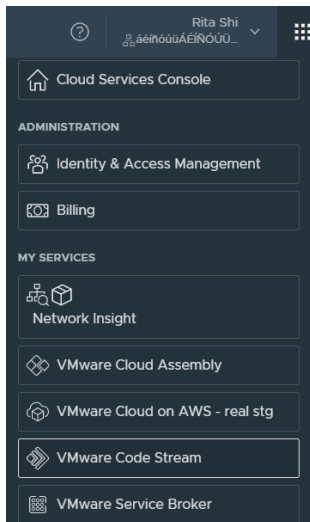
Rolle „Executor“

Sie können Pipelines ausführen und Aktionen in Verbindung mit Benutzervorgangsaufgaben durchführen. Sie können darüber hinaus Pipeline-Ausführungen fortsetzen, anhalten und abbrechen. Sie können Pipelines jedoch nicht ändern.

Vorgehensweise beim Zuweisen und Aktualisieren von Rollen

Sie müssen Administrator sein, um Rollen für andere Benutzer zuzuweisen und zu aktualisieren.

- 1 Wenn Sie die aktiven Benutzer und deren Rollen anzeigen möchten, klicken Sie in vRealize Automation oben rechts auf das Symbol mit den neun Punkten.
- 2 Klicken Sie auf **Identitäts- und Zugriffsverwaltung**.



- 3 Um Benutzernamen und Rollen anzuzeigen, klicken Sie auf **Aktive Benutzer**.



- 4 Um Rollen für einen Benutzer hinzuzufügen oder seine Rollen zu ändern, aktivieren Sie das Kontrollkästchen neben dem Benutzernamen und klicken Sie auf **Rollen bearbeiten**.
- 5 Wenn Sie Benutzerrollen hinzufügen oder ändern, können Sie auch den Zugriff auf Dienste hinzufügen.
- 6 Um Ihre Änderungen zu speichern, klicken Sie auf **Speichern**.

Definition der Benutzervorgänge und Genehmigungen in vRealize Automation Code Stream

Im Bereich „Benutzervorgänge“ werden Pipeline-Ausführungen angezeigt, die genehmigt werden müssen. Der erforderliche Genehmiger kann die Pipeline-Ausführung entweder genehmigen oder ablehnen.

Beim Erstellen einer Pipeline müssen Sie der Pipeline unter Umständen eine Genehmigung hinzufügen, wenn Folgendes zutrifft:

- Ein Teammitglied muss Ihren Code überprüfen.
- Ein anderer Benutzer muss ein Build-Artefakt bestätigen.
- Sie müssen sicherstellen, dass alle Tests abgeschlossen sind.
- Eine Aufgabe verwendet eine von einem Administrator als eingeschränkt gekennzeichnete Ressource, und die Aufgabe muss genehmigt werden.
- Die Pipeline gibt Software für die Produktion frei.

Um zu bestimmen, ob eine Pipeline-Aufgabe genehmigt werden soll, muss der erforderliche Genehmiger über die entsprechende Berechtigung und das nötige Wissen verfügen.

Wenn Sie eine Benutzervorgangsaufgabe hinzufügen, können Sie das Ablaufzeitlimit in Tagen, Stunden oder Minuten angeben. Beispiel: Sie benötigen unter Umständen den erforderlichen Benutzer, um die Pipeline innerhalb von 30 Minuten zu genehmigen. Wenn die Pipeline nicht innerhalb von 30 Minuten genehmigt wird, schlägt die Pipeline erwartungsgemäß fehl.

Wenn Sie das Senden von E-Mail-Benachrichtigungen aktivieren, sendet die Aufgabe „Benutzervorgang“ Benachrichtigungen nur an Genehmiger, die über eine vollständige E-Mail-Adresse verfügen, und nicht an Genehmigernamen, die nicht in einem E-Mail-Format vorliegen.

Nach der Genehmigung der Aufgabe durch den erforderlichen Benutzer:

- Die Ausführung der ausstehenden Pipeline kann fortgesetzt werden.
- Wenn die Pipeline fortgesetzt wird, werden alle vorherigen ausstehenden Anforderungen zur Genehmigung dieser Benutzervorgangsaufgabe abgebrochen.

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 |

☒

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

Im Bereich „Benutzervorgänge“ werden Elemente zur Genehmigung oder Ablehnung als aktive oder inaktive Elemente angezeigt. Jedes Element wird einer Benutzervorgangsaufgabe in einer Pipeline zugeordnet.

- **Aktive Elemente** warten auf den Genehmiger, der die Aufgabe überprüfen und genehmigen oder ablehnen muss. Wenn Sie als Benutzer in der Genehmigungsliste enthalten sind, können Sie die Benutzervorgangszeile erweitern und auf **Akzeptieren** oder **Ablehnen** klicken.
- **Inaktive Elemente** wurden genehmigt oder abgelehnt. Wenn ein Benutzer den Benutzervorgang abgelehnt hat oder die Genehmigung für die Aufgabe abgelaufen ist, kann sie nicht mehr genehmigt werden.

Der Index ist eine eindeutige sechsstellige alphanumerische Zeichenfolge, die Sie als Filter für die Suche nach einer bestimmten Genehmigung verwenden können.

Pipeline-Genehmigungen werden auch im Bereich **Ausführungen** angezeigt.

- Der Status von Pipelines, die auf ihre Genehmigung warten, wird als „Warten“ angezeigt.
- Zu den weiteren Statusangaben gehören „In Warteschlange“, „Abgeschlossen“ und „Fehlgeschlagen“.
- Befindet sich die Pipeline im Wartezustand, muss der erforderliche Genehmiger Ihre Pipeline-Aufgabe genehmigen.

Erstellen und Verwenden von Pipelines in vRealize Automation Code Stream

3

Sie können vRealize Automation Code Stream verwenden, um den Erstellungs-, Test- und Bereitstellungsprozess zu entwickeln. Mit vRealize Automation Code Stream richten Sie die Infrastruktur ein, die Ihren Versionszyklus unterstützt, und erstellen Pipelines, die Ihre Softwareversionsaktivitäten entwickeln. vRealize Automation Code Stream stellt Ihre Software aus Entwicklungscode basierend auf Tests in Ihren Produktionsinstanzen bereit.

Jede Pipeline umfasst Phasen und Aufgaben. Phasen stellen Ihre Entwicklungsphasen dar, und Aufgaben führen die Aktionen aus, die für die Bereitstellung Ihrer Softwareanwendung über die Phasen erforderlich sind.

Pipelines in vRealize Automation Code Stream

Eine Pipeline ist ein Modell zur kontinuierlichen Integration und kontinuierlichen Bereitstellung Ihres Softwareveröffentlichungsprozesses. Sie gibt Ihre Software vom Quellcode über Tests bis zur Produktion frei. Sie enthält eine Abfolge von Phasen, die Aufgaben enthalten, die die Aktivitäten in Ihrem Softwareversionszyklus darstellen. Über die Pipeline geht Ihre Softwareanwendung von einer Phase zur nächsten über.

Sie fügen Endpoints hinzu, sodass die Aufgaben in Ihrer Pipeline mit Datenquellen, Repositorys oder Benachrichtigungssystemen verbunden werden können.

Erstellen von Pipelines

Sie können eine Pipeline erstellen, indem Sie mit einer leeren Arbeitsfläche beginnen, eine intelligente Pipeline-Vorlage verwenden oder YAML-Code importieren.

- Verwenden Sie die leere Arbeitsfläche. Ein Beispiel finden Sie unter [Planen eines nativen CICD-Builds in vRealize Automation Code Stream vor dem manuellen Hinzufügen von Aufgaben](#).
- Verwenden Sie eine intelligente Pipeline-Vorlage. Ein Beispiel finden Sie unter [Kapitel 4 Planen eines nativen Builds, der Integration und Bereitstellung von Code in vRealize Automation Code Stream](#).
- Importieren Sie YAML-Code. Klicken Sie auf **Pipelines > Importieren**. Wählen Sie im Dialogfeld **Importieren** die YAML-Datei aus oder geben Sie den YAML-Code ein und klicken Sie auf **Importieren**.

Wenn Sie die leere Arbeitsfläche zum Erstellen einer Pipeline verwenden, fügen Sie Phasen, Aufgaben und Genehmigungen hinzu. Die Pipeline automatisiert den Vorgang, der Ihre Anwendung erstellt, testet, bereitstellt und freigibt. Die Aufgaben in jeder Phase führen Aktionen aus, die Ihren Code in jeder Phase erstellen, testen und freigeben.

Tabelle 3-1. Beispiel für Pipeline-Phasen und Verwendungen

| Beispielphase | Beispiele für Anwendungsmöglichkeiten |
|---------------|---|
| Entwicklung | <p>In einer Entwicklungsphase können Sie unter anderem eine Maschine bereitstellen, ein Artefakt abrufen oder eine Build-Aufgabe hinzufügen, die einen Docker-Host für die kontinuierliche Integration Ihres Codes erstellt.</p> <p>Beispiel:</p> <ul style="list-style-type: none"> ■ Informationen zum Planen und Erstellen eines CI-Builds (Continuous Integration, kontinuierliche Integration), der Ihren Code mithilfe der nativen Build-Funktion in vRealize Automation Code Stream bereitstellt, finden Sie unter Planen eines nativen CI-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage. |
| Testen | <p>In einer Testphase können Sie unter anderem eine Jenkins-Aufgabe zum Testen Ihrer Softwareanwendung hinzufügen und Testtools wie JUnit, JaCoCo und andere integrieren.</p> <p>Beispiel:</p> <ul style="list-style-type: none"> ■ Integrieren Sie vRealize Automation Code Stream in Jenkins und führen Sie einen Jenkins-Auftrag in Ihrer Pipeline aus, der Ihren Quellcode erstellt und testet. Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Integrieren von vRealize Automation Code Stream in Jenkins. ■ Erstellen Sie benutzerdefinierter Skripts, die die Funktion von vRealize Automation Code Stream erweitern, sodass Sie sie in Ihre eigenen Build-, Test- und Bereitstellungstools integrieren können. Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Integrieren von eigenen Build-, Test- und Bereitstellungstools mit vRealize Automation Code Stream. ■ Verfolgen Sie Trends bei der Nachbearbeitung für eine CI-Pipeline (Continuous Integration, kontinuierliche Integration). Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Verwenden benutzerdefinierter Dashboards zum Verfolgen von wichtigen Leistungsindikatoren für meine Pipeline in vRealize Automation Code Stream. |
| Produktion | <p>In einer Produktionsphase können Sie eine Cloud-Vorlage in vRealize Automation Cloud Assembly integrieren, die Ihre Infrastruktur sowie Software auf einem Kubernetes-Cluster bereitstellt usw.</p> <p>Beispiel:</p> <ul style="list-style-type: none"> ■ Beispiele für Entwicklungs- und Produktionsphasen, die Ihre Softwareanwendung in Ihrem eigenen Blau/Grün-Bereitstellungsmodell bereitstellen können, finden Sie unter Vorgehensweise zum Bereitstellen meiner Anwendung in vRealize Automation Code Stream für meine Blau/Grün-Bereitstellung. ■ Informationen zum Integrieren einer Cloud-Vorlage in Ihre Pipeline finden Sie unter Vorgehensweise zum Automatisieren der Version einer Anwendung, die von einer YAML-Cloud-Vorlage in vRealize Automation Code Stream bereitgestellt wird. Sie können auch eine Bereitstellungsaufgabe hinzufügen, die ein Skript ausführt, um die Anwendung zur Freigabe bereitzustellen. ■ Informationen zur Automatisierung der Bereitstellung Ihrer Softwareanwendungen auf einem Kubernetes-Cluster finden Sie unter Vorgehensweise zum Automatisieren der Freigabe einer Anwendung in vRealize Automation Code Stream in einem Kubernetes-Cluster. ■ Informationen zum Integrieren von Code in Ihre Pipeline und zur Bereitstellung Ihres Build-Images finden Sie unter Vorgehensweise zur kontinuierlichen Integration von Code aus einem GitHub- oder GitLab-Repository in eine Pipeline in vRealize Automation Code Stream. |

Sie können Ihre Pipeline auch als YAML-Datei exportieren. Klicken Sie auf **Pipelines**, dann auf eine Pipeline-Karte und anschließend auf **Aktionen > Exportieren**.

Genehmigen von Pipelines

An bestimmten Punkten in Ihrer Pipeline können Sie eine Genehmigung von einem anderen Teammitglied einholen.

- Informationen zum Anfordern einer Genehmigung für eine Pipeline durch Einschließen einer Benutzervorgangsaufgabe in eine Pipeline finden Sie unter [Vorgehensweise zum Ausführen einer Pipeline und Anzeigen von Ergebnissen](#). Diese Aufgabe sendet eine E-Mail-Benachrichtigung an den Benutzer, der sie überprüfen muss. Der Prüfer muss die Genehmigung entweder genehmigen oder ablehnen, bevor die Pipeline weiter ausgeführt werden kann. Wenn das Ablaufzeitlimit in der Benutzervorgangsaufgabe auf Tage, Stunden oder Minuten festgelegt ist, muss der erforderliche Benutzer die Pipeline vor Ablauf der Aufgabe genehmigen. Ansonsten schlägt die Pipeline erwartungsgemäß fehl.
- Wenn eine Aufgabe oder Phase in irgendeiner Phase einer Pipeline fehlschlägt, können Sie von vRealize Automation Code Stream ein Jira-Ticket erstellen lassen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Erstellen eines JIRA-Tickets in vRealize Automation Code Stream bei einer fehlgeschlagenen Pipeline-Aufgabe](#).

Auslösen von Pipelines

Pipelines können ausgelöst werden, wenn Entwickler ihren Code in das Repository einchecken, Code überprüfen oder wenn ein neues oder aktualisiertes Build-Artefakt erkannt wird.

- Um vRealize Automation Code Stream in den Git-Lebenszyklus zu integrieren und eine Pipeline auszulösen, wenn Entwickler ihren Code aktualisieren, verwenden Sie den Git-Auslöser. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwenden des Git-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#).
- Um vRealize Automation Code Stream in den Lebenszyklus für Gerrit-Codeprüfungen zu integrieren und eine Pipeline für Codeprüfungen auszulösen, verwenden Sie den Gerrit-Auslöser. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwenden des Gerrit-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#).
- Mit dem Docker-Auslöser können Sie eine Pipeline auslösen, wenn ein Docker-Build-Artefakt erstellt oder aktualisiert wird. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwenden des Docker-Auslösers in vRealize Automation Code Stream zum Ausführen einer kontinuierlichen Bereitstellungs-Pipeline](#).

Weitere Informationen zu den Auslösern, die vRealize Automation Code Stream unterstützt, finden Sie unter [Kapitel 7 Auslösen von Pipelines in vRealize Automation Code Stream](#).

Dieses Kapitel enthält die folgenden Themen:

- [Vorgehensweise zum Ausführen einer Pipeline und Anzeigen von Ergebnissen](#)
- [In vRealize Automation Code Stream verfügbare Aufgabentypen](#)
- [Vorgehensweise zum Verwenden von Variablenbindungen in vRealize Automation Code Stream-Pipelines](#)

- Vorgehensweise zum Verwenden von Variablenbindungen in einer Bedingungsangabe zum Ausführen oder Anhalten einer Pipeline in vRealize Automation Code Stream
- Welche Variablen und Ausdrücke kann ich beim Binden von Pipelineaufgaben in vRealize Automation Code Stream verwenden?
- Vorgehensweise zum Senden von Benachrichtigungen über meine Pipeline in vRealize Automation Code Stream
- Vorgehensweise zum Erstellen eines JIRA-Tickets in vRealize Automation Code Stream bei einer fehlgeschlagenen Pipeline-Aufgabe
- Vorgehensweise zum Rollback meiner Bereitstellung in vRealize Automation Code Stream

Vorgehensweise zum Ausführen einer Pipeline und Anzeigen von Ergebnissen

Sie können eine Pipeline von der Pipeline-Karte, im Bearbeitungsmodus der Pipeline und von der Pipeline-Ausführung aus ausführen. Mit den verfügbaren Auslösern können Sie auch festlegen, dass vRealize Automation Code Stream eine Pipeline ausführt, wenn bestimmte Ereignisse eintreten.

Wenn alle Phasen und Aufgaben in Ihrer Pipeline gültig sind, kann die Pipeline freigegeben, ausgeführt oder ausgelöst werden.

Zum Ausführen oder Auslösen der Pipeline mithilfe von vRealize Automation Code Stream können Sie die Pipeline über die Pipeline-Karte oder in der Pipeline selbst aktivieren und ausführen. Anschließend können Sie die Pipeline-Ausführung anzeigen, um zu überprüfen, ob Ihr Code in der Pipeline erstellt, getestet und bereitgestellt wurde.

Während der Ausführung einer Pipeline können Sie die Ausführung löschen, wenn Sie als Administrator oder Benutzer ohne Administratorrechte fungieren.

- Administrator: Zum Löschen einer laufenden Pipeline-Ausführung klicken Sie auf **Ausführungen**. Klicken Sie für die zu löschende Ausführung auf **Aktionen > Löschen**.
- Benutzer ohne Administratorrechte: Zum Löschen einer laufenden Pipeline-Ausführung klicken Sie auf **Ausführungen** und dann auf **Alt Shift d**.

Wenn eine Pipeline-Ausführung nicht mehr reagiert, kann ein Administrator die Ausführung über die Seite „Ausführungen“ oder die Seite „Ausführungsdetails“ aktualisieren.

- Seite „Ausführungen“: Klicken Sie auf **Ausführungen**. Klicken Sie für die zu aktualisierende Ausführung auf **Aktionen > Synchronisieren**.
- Seite „Ausführungsdetails“: Klicken Sie auf **Ausführungen**, dann auf den Link zu den Ausführungsdetails und schließlich auf **Aktionen > Synchronisieren**.

Verwenden Sie die Auslöser zum Ausführen einer Pipeline bei bestimmten Ereignissen.

- Der Git-Auslöser kann eine Pipeline ausführen, wenn Entwickler Code aktualisieren.
- Der Gerrit-Auslöser kann eine Pipeline ausführen, wenn Codeprüfungen stattfinden.

- Der Docker-Auslöser kann eine Pipeline ausführen, wenn ein Artefakt in einer Docker-Registrierung erstellt wird.
- Mit dem Befehl `curl` kann Jenkins zur Ausführung einer Pipeline veranlasst werden, wenn ein Jenkins-Build beendet wird.

Weitere Informationen zur Verwendung der Auslöser finden Sie unter [Kapitel 7 Auslösen von Pipelines in vRealize Automation Code Stream](#).

In folgendem Verfahren wird die Ausführung einer Pipeline über die Pipeline-Karte, die Anzeige von Ausführungen und Ausführungsdetails sowie die Verwendung der Aktionen dargestellt. Darüber hinaus erhalten Sie Informationen zum Freigeben einer Pipeline, die dann zu vRealize Automation Service Broker hinzugefügt werden kann.

Voraussetzungen

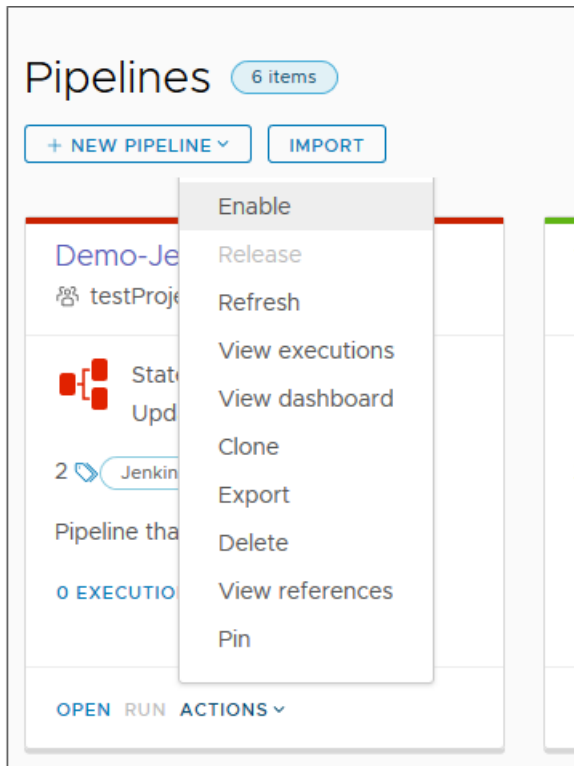
- Stellen Sie sicher, dass mindestens eine Pipeline erstellt wird. Weitere Informationen finden Sie in den Beispielen unter [Kapitel 5 Lernprogramme für die Verwendung von vRealize Automation Code Stream](#).

Verfahren

1 Aktivieren Sie die Pipeline.

Zum Ausführen oder Freigeben einer Pipeline müssen Sie diese erst aktivieren.

- a Klicken Sie auf **Pipelines**.
- b Klicken Sie auf der Pipeline-Karte auf **Aktionen > Aktivieren**.



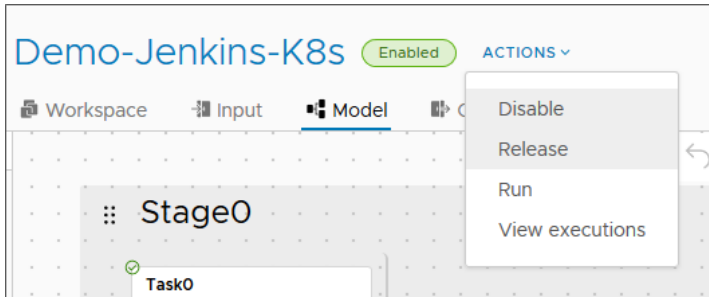
Sie können die Pipeline auch aktivieren, während Sie sich in der Pipeline befinden. Wenn die Pipeline bereits aktiviert ist, ist **Ausführen** aktiv, und im Menü **Aktionen** wird **Deaktivieren** angezeigt.

2 (Optional) Geben Sie die Pipeline frei.

Wenn Sie Ihre Pipeline als Katalogelement in vRealize Automation Service Broker zur Verfügung stellen möchten, müssen Sie sie in vRealize Automation Code Stream freigeben.

- a Klicken Sie auf **Pipelines**.
- b Klicken Sie auf der Pipeline-Karte auf **Aktionen > Freigeben**.

Sie können die Pipeline auch freigeben, während Sie sich in der Pipeline befinden.



Nach der Freigabe der Pipeline öffnen Sie vRealize Automation Service Broker, um die Pipeline als Katalogelement hinzuzufügen und auszuführen. Weitere Informationen finden Sie unter [Hinzufügen von vRealize Automation Code Stream-Pipelines zum vRealize Automation Service Broker-Katalog](#).

Hinweis Wenn die Pipeline mehr als 120 Minuten ausgeführt werden muss, geben Sie eine geschätzte Ausführungszeit als Wert für die Zeitüberschreitung bei Anforderung an. Um die Zeitüberschreitung bei Anforderung für ein Projekt festzulegen oder zu überprüfen, öffnen Sie vRealize Automation Service Broker als Administrator und wählen **Infrastruktur > Projekte** aus. Klicken Sie auf den Namen des Projekts und dann auf **Bereitstellen**.

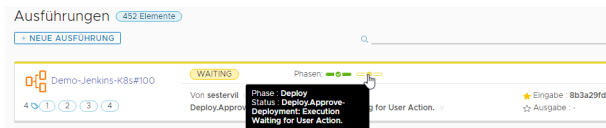
Wenn der Wert für die Zeitüberschreitung bei Anforderung nicht festgelegt ist, wird eine mehr als 120 Minuten umfassende Ausführung mit dem Fehler „Zeitüberschreitung bei Rückrufanforderung“ angezeigt. Die Ausführung der Pipeline ist jedoch nicht betroffen.

- 3 Klicken Sie auf der Pipeline-Karte auf **Ausführen**.
- 4 Um die Pipeline während der Ausführung anzuzeigen, klicken Sie auf **Ausführungen**.

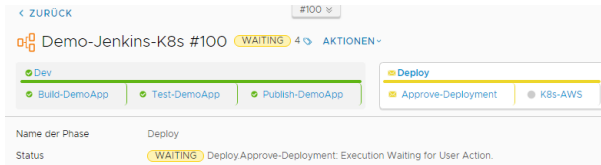
Die Pipeline führt die einzelnen Phasen der Reihe nach aus, und die Pipeline-Ausführung zeigt für jede Phase ein Statussymbol an. Wenn die Pipeline eine Benutzervorgangsaufgabe enthält, muss ein Benutzer die Aufgabe genehmigen, damit die Ausführung der Pipeline fortgesetzt wird. Wenn eine Benutzervorgangsaufgabe verwendet wird, wird die Ausführung der Pipeline so lange angehalten, bis der erforderliche Benutzer die Aufgabe genehmigt hat.

Sie können beispielsweise die Aufgabe „Benutzervorgang“ verwenden, um die Bereitstellung von Code in einer Produktionsumgebung zu genehmigen.

Wenn das Ablaufzeitlimit in der Benutzervorgangsaufgabe auf Tage, Stunden oder Minuten festgelegt ist, muss der erforderliche Benutzer die Pipeline vor Ablauf der Aufgabe genehmigen. Ansonsten schlägt die Pipeline erwartungsgemäß fehl.



- 5 Mit einem Klick auf das Statussymbol für die Phase können Sie die Pipeline-Phase anzeigen, die auf die Benutzergenehmigung wartet.



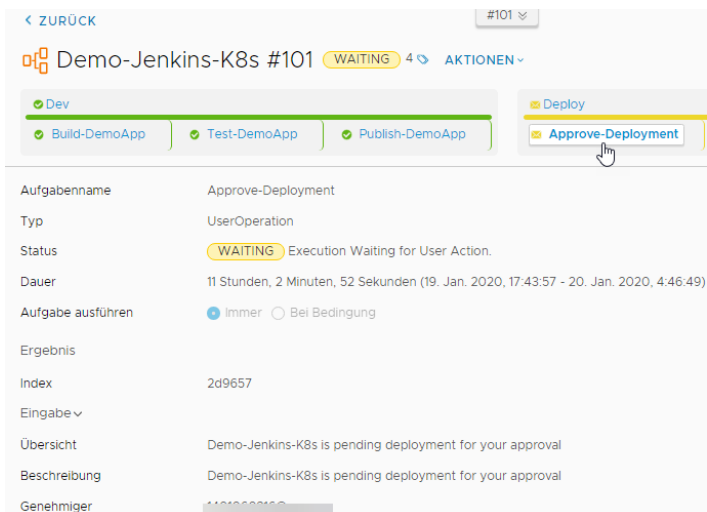
- 6 Mit einem Klick auf die Aufgabe können Sie Details für die Aufgabe anzeigen.

Nachdem der erforderliche Benutzer die Aufgabe genehmigt hat, muss ein Benutzer mit der entsprechenden Rolle die Pipeline fortsetzen. Weitere Informationen zu den erforderlichen Rollen finden Sie unter [Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream](#).

Wenn eine Ausführung fehlschlägt, müssen Sie die Fehlerursache sichten und beheben.

Navigieren Sie dann zur Ausführung und klicken Sie auf **Aktionen > Erneut ausführen**.

Sie können primäre Pipeline-Ausführungen und verschachtelte Ausführungen fortsetzen.



- 7 Aus der Pipeline-Ausführung können Sie auf **Aktionen** klicken, um die Pipeline anzuzeigen und eine Aktion auszuwählen, wie z. B. **Anhalten**, **Abbrechen** usw. Während einer Pipeline-Ausführung können Sie diese als Administrator löschen oder synchronisieren. Als Benutzer ohne Administratorrechte können Sie eine laufende Pipeline löschen.

- 8 Um ganz einfach zwischen Ausführungen zu navigieren und die Details für eine Aufgabe anzuzeigen, klicken Sie auf **Ausführungen** und dann auf eine Pipeline-Ausführung. Klicken Sie anschließend oben auf die Registerkarte und wählen Sie die Pipeline-Ausführung aus.



Ergebnisse

Herzlichen Glückwunsch! Sie haben eine Pipeline ausgeführt, die Pipeline-Ausführung untersucht und eine Benutzervorgangsaufgabe angezeigt, für die eine Genehmigung erforderlich war, um die Pipeline-Ausführung fortzusetzen. Außerdem sind Sie über das Menü **Aktionen** in der Pipeline-Ausführung zum Pipeline-Modell zurückgekehrt, sodass Sie alle erforderlichen Änderungen vornehmen können.

Nächste Schritte

Weitere Informationen zur Verwendung von vRealize Automation Code Stream zum Automatisieren des Softwarefreigabezyklus finden Sie unter [Kapitel 5 Lernprogramme für die Verwendung von vRealize Automation Code Stream](#).

In vRealize Automation Code Stream verfügbare Aufgabentypen

Wenn Sie Ihre Pipeline konfigurieren, fügen Sie bestimmte Aufgabentypen hinzu, die die Pipeline für die von Ihnen benötigten Aktionen ausführt. Jeder Aufgabentyp lässt sich in eine andere Anwendung integrieren und aktiviert Ihre Pipeline, während sie Ihre Anwendungen erstellt, testet und bereitstellt.

Egal, ob Sie zur Bereitstellung Artefakte aus einem Repository abrufen, ein Remote-Skript ausführen müssen oder von einem Teammitglied eine Genehmigung für einen Benutzervorgang benötigen, hat vRealize Automation Code Stream den geeigneten Aufgabentyp, damit Sie Ihre Pipeline ausführen können!

Bevor Sie eine Aufgabe in Ihrer Pipeline verwenden, stellen Sie sicher, dass der entsprechende Endpoint verfügbar ist.

Tabelle 3-2. Anfordern einer Genehmigung oder Festlegen eines Entscheidungspunkts

| Aufgabentyp | Funktionsweise | Beispiele und Details |
|------------------------|---|---|
| Benutzervorgang | Eine Aufgabe des Typs „Benutzervorgang“ ermöglicht eine erforderliche Genehmigung, die steuert, wann eine Pipeline ausgeführt wird und für eine Genehmigung angehalten werden muss. | Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Ausführen einer Pipeline und Anzeigen von Ergebnissen . und Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream . |
| Bedingung | Fügt einen Entscheidungspunkt hinzu, der bestimmt, ob die Pipeline auf der Grundlage von Bedingungsausdrücken weiterhin ausgeführt oder angehalten wird. Wenn die Bedingung „true“ lautet, führt die Pipeline nachfolgende Aufgaben aus. Wenn sie „false“ lautet, wird die Pipeline angehalten. | Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Verwenden von Variablenbindungen in einer Bedingungs Aufgabe zum Ausführen oder Anhalten einer Pipeline in vRealize Automation Code Stream . |

Tabelle 3-3. Automatisieren der kontinuierlichen Integration und Bereitstellung

| Aufgabentyp | Funktionsweise | Beispiele und Details |
|-------------------|---|--|
| Cloud-Vorlage | Stellt eine Automatisierungs-Cloud-Vorlage aus GitHub sowie eine Anwendung bereit und automatisiert die kontinuierliche Integration und kontinuierliche Bereitstellung (CICD) dieser Cloud-Vorlage für Ihre Bereitstellung. | <p>Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Automatisieren der Version einer Anwendung, die von einer YAML-Cloud-Vorlage in vRealize Automation Code Stream bereitgestellt wird.</p> <p>Die Cloud-Vorlagenparameter werden angezeigt, nachdem Sie zuerst Erstellen oder Aktualisieren und dann Cloud-Vorlage und Version ausgewählt haben. Sie können die folgenden Elemente, die Variablenbindungen ermöglichen, zu den Eingabetextbereichen in der Cloud-Vorlagenaufgabe hinzufügen:</p> <ul style="list-style-type: none"> ■ Ganzzahl ■ Aufzählungszeichenfolge ■ Boolesch ■ Array-Variable <p>Wenn Sie Variablenbindung in der Eingabe verwenden, beachten Sie die folgenden Ausnahmen. Für Aufzählungen müssen Sie einen Aufzählungswert aus einem festen Satz auswählen. Für boolesche Werte müssen Sie den Wert in den Texteingabebereich eingeben.</p> <p>Der Cloud-Vorlagenparameter wird in der Cloud-Vorlagenaufgabe angezeigt, wenn eine Cloud-Vorlage in vRealize Automation Cloud Assembly Eingabevariablen enthält. Wenn eine Cloud-Vorlage beispielsweise den Eingabetyp <code>Integer</code> aufweist, können Sie die Ganzzahl direkt oder als Variable mithilfe der Variablenbindung eingeben.</p> |
| CI | Die CI-Aufgabe ermöglicht die kontinuierliche Integration Ihres Codes in Ihre Pipeline, indem ein Docker-Build-Image aus einem Registrierungs-Endpoint abgerufen und in einem Kubernetes-Cluster bereitgestellt wird. | Weitere Informationen hierzu finden Sie unter Planen eines nativen CICD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage . |
| Benutzerdefiniert | Die Aufgabe „Benutzerdefiniert“ integriert vRealize Automation Code Stream in Ihre eigenen Build-, Test- und Bereitstellungstools. | Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Integrieren von eigenen Build-, Test- und Bereitstellungstools mit vRealize Automation Code Stream . |

Tabelle 3-3. Automatisieren der kontinuierlichen Integration und Bereitstellung (Fortsetzung)

| Aufgabentyp | Funktionsweise | Beispiele und Details |
|-------------|--|---|
| Kubernetes | Automatisiert die Bereitstellung Ihrer Softwareanwendungen für Kubernetes-Cluster auf AWS. | Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Automatisieren der Freigabe einer Anwendung in vRealize Automation Code Stream in einem Kubernetes-Cluster . |
| Pipeline | <p>Verschachtelt eine Pipeline in einer primären Pipeline. Wenn eine Pipeline verschachtelt ist, verhält sie sich in der primären Pipeline wie eine Aufgabe.</p> <p>Auf der Registerkarte „Aufgabe“ der primären Pipeline können Sie problemlos zur verschachtelten Pipeline navigieren, indem Sie auf den dorthin führenden Link klicken. Die verschachtelte Pipeline wird in einer neuen Browser-Registerkarte geöffnet.</p> | Um verschachtelte Pipelines in Ausführungen zu finden, geben Sie Verschachtelt in den Suchbereich ein. |

Tabelle 3-4. Integrieren von Entwicklungs-, Test- und Bereitstellungsanwendungen

| Aufgabentyp | Funktion | Beispiele und Details |
|-------------|---|---|
| Bamboo | Interagiert mit einem Bamboo-CI-Server, der die Software bei der Vorbereitung für die Bereitstellung kontinuierlich erstellt, testet und integriert sowie Code-Builds auslöst, wenn Entwickler Änderungen übergeben. Macht die vom Bamboo-Build erzeugten Artefakt-Speicherorte verfügbar, sodass die Aufgabe die Parameter für andere Aufgaben ausgeben kann, die für den Build und die Bereitstellung verwendet werden. | Stellen Sie eine Verbindung mit einem Bamboo-Server-Endpoint her und starten Sie einen Bamboo-Build-Plan aus Ihrer Pipeline. |
| Jenkins | Löst Jenkins-Jobs aus, die Ihren Quellcode erstellen und testen, Testfälle ausführen und benutzerdefinierte Skripts verwenden können. | Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Integrieren von vRealize Automation Code Stream in Jenkins . |
| TFS | Ermöglicht Ihnen, Ihre Pipeline mit Team Foundation Server zu verbinden, um Build-Projekte zu verwalten und aufzurufen, einschließlich konfigurierter Aufträge zum Erstellen und Testen Ihres Codes. | vRealize Automation Code Stream unterstützt Team Foundation Server 2013 und 2015. |
| vRO | Erweitert den Funktionsumfang von vRealize Automation Code Stream, indem vordefinierte oder benutzerdefinierte Workflows in vRealize Orchestrator ausgeführt werden. | Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Integrieren von vRealize Automation Code Stream in vRealize Orchestrator . |

Tabelle 3-5. Integrieren anderer Anwendungen über eine API

| Aufgabentyp | Funktion | Beispiele und Details |
|-------------|--|---|
| REST | Integriert vRealize Automation Code Stream mit anderen Anwendungen, die eine REST API verwenden, sodass Sie kontinuierlich miteinander interagierende Softwareanwendungen entwickeln und bereitstellen können. | Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Verwenden einer REST API für die Integration von vRealize Automation Code Stream in andere Anwendungen . |
| Abfrage | Ruft eine REST API auf und fragt sie ab, bis die Pipeline-Aufgabe die Beendigungskriterien erfüllt und abgeschlossen ist. | Weitere Informationen hierzu finden Sie unter Vorgehensweise zum Verwenden einer REST API für die Integration von vRealize Automation Code Stream in andere Anwendungen . |

Tabelle 3-6. Ausführen von Remote-Skripts und benutzerdefinierten Skripten

| Aufgabentyp | Funktionsweise | Beispiele und Details |
|-------------|---|--|
| PowerShell | <p>Mit der Aufgabe „PowerShell“ kann vRealize Automation Code Stream Skriptbefehle auf einem Remote-Host ausführen. Ein Skript kann beispielsweise Testaufgaben automatisieren und administrative Befehlstypen ausführen.</p> <p>Das Skript kann remote oder benutzerdefiniert sein. Es kann über HTTP oder HTTPS eine Verbindung herstellen und TLS verwenden.</p> <p>Auf dem Windows-Host muss der winrm-Dienst konfiguriert sein und für winrm müssen MaxShellsPerUser und MaxMemoryPerShellMB konfiguriert sein.</p> <p>Um eine PowerShell-Aufgabe ausführen zu können, müssen Sie über eine aktive Sitzung mit dem Remote-Windows-Host verfügen.</p> <p>Länge der PowerShell-Befehlszeile</p> <p>Wenn Sie einen base64-PowerShell-Befehl eingeben, beachten Sie, dass Sie die Gesamtlänge des Befehls berechnen müssen.</p> <p>Die vRealize Automation Code Stream-Pipeline codiert und umhüllt einen base64-PowerShell-Befehl in einem anderen Befehl, wodurch die Gesamtlänge des Befehls erhöht wird.</p> <p>Die maximal zulässige Länge eines winrm-PowerShell-Befehls beträgt 8192 Byte. Der Grenzwert für die Befehlslänge ist für die PowerShell-Aufgabe niedriger, wenn sie verschlüsselt und umhüllt ist. Folglich müssen Sie die Befehlslänge berechnen, bevor Sie den PowerShell-Befehl eingeben.</p> <p>Der Grenzwert für die Befehlslänge für die vRealize Automation Code Stream-PowerShell-Aufgabe richtet sich nach der base64-codierten Länge des ursprünglichen Befehls. Die Länge des Befehls wird wie folgt berechnet.</p> $3 * (\text{length of original command} / 4) - (\text{numberOfPaddingCharacters}) + 77 (\text{Length of Write-output command})$ <p>Die Länge des Befehls für vRealize Automation Code Stream muss unter dem maximalen Grenzwert von 8192 liegen.</p> | <p>Wenn Sie MaxShellsPerUser und MaxMemoryPerShellMB konfigurieren, gilt Folgendes:</p> <ul style="list-style-type: none"> Der zulässige Wert für MaxShellsPerUser ist 500 für 50 gleichzeitige Pipelines mit fünf PowerShell-Aufgaben für jede Pipeline. Um den Wert festzulegen, führen Sie Folgendes aus: winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'. Der zulässige Arbeitsspeicherwert für MaxMemoryPerShellMB ist 2048. Um den Wert festzulegen, führen Sie Folgendes aus: winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'. <p>Das Skript schreibt die Ausgabe in eine Antwortdatei, die von einer anderen Pipeline verwendet werden kann.</p> |
| SSH | <p>Die Aufgabe „SSH“ ermöglicht, dass die Aufgabe „Bash-Shell-Skript“ Skriptbefehle auf einem Remote-Host ausführt. Ein Skript kann beispielsweise Testaufgaben automatisieren und administrative Befehlstypen ausführen.</p> | <p>Das Skript kann remote oder benutzerdefiniert sein. Ein Skript kann beispielsweise folgendermaßen aussehen:</p> <pre>message="Hello World" echo \$message</pre> |

Tabelle 3-6. Ausführen von Remote-Skripts und benutzerdefinierten Skripts (Fortsetzung)

| Aufgabentyp | Funktionsweise | Beispiele und Details |
|-------------|--|--|
| | <p>Das Skript kann remote oder benutzerdefiniert sein. Es kann über HTTP oder HTTPS eine Verbindung herstellen und erfordert einen privaten Schlüssel oder ein Kennwort.</p> <p>Der SSH-Dienst muss auf dem Linux-Host konfiguriert werden, und die SSHD-Konfiguration <code>MaxSessions</code> muss auf 50 festgelegt werden.</p> | <p>Das Skript schreibt die Ausgabe in eine Antwortdatei, die von einer anderen Pipeline verwendet werden kann.</p> |

Vorgehensweise zum Verwenden von Variablenbindungen in vRealize Automation Code Stream-Pipelines

Beim Binden einer Pipeline-Aufgabe erstellen Sie eine Abhängigkeit für die Aufgabe, wenn die Pipeline ausgeführt wird. Zum Erstellen einer Bindung für eine Pipeline-Aufgabe stehen mehrere Möglichkeiten zur Verfügung. Sie können eine Aufgabe an eine andere Aufgabe, an eine Variable, einen Ausdruck oder an eine Bedingung binden.

Vorgehensweise zum Anwenden von Dollarbindungen auf Cloud-Vorlagenvariablen in einer Cloud-Vorlagenaufgabe

Sie können Dollarbindungen auf Cloud-Vorlagenvariablen in einer Cloud-Vorlagenaufgabe der vRealize Automation Code Stream-Pipeline anwenden. Die Art und Weise, wie Sie die Variablen in vRealize Automation Code Stream ändern, richtet sich nach der Kodierung der Variableneigenschaften in der Cloud-Vorlage.

Wenn Sie Dollarbindungen in einer Cloud-Vorlagenaufgabe verwenden müssen, die aktuelle Version der in der Cloud-Vorlagenaufgabe verwendeten Cloud-Vorlage dies aber nicht zulässt, ändern Sie die Cloud-Vorlage in vRealize Automation Cloud Assembly und stellen Sie eine neue Version bereit. Verwenden Sie anschließend die neue Cloud-Vorlagenversion in Ihrer Cloud-Vorlagenaufgabe und fügen Sie die Dollarbindungen bei Bedarf hinzu.

Zum Anwenden von Dollarbindungen auf die von der vRealize Automation Cloud Assembly-Cloud-Vorlage bereitgestellten Eigenschaftstypen benötigen Sie die entsprechenden Berechtigungen.

- Sie müssen dieselbe Rolle wie die Person aufweisen, die die Bereitstellung der Cloud-Vorlage in vRealize Automation Cloud Assembly erstellt hat.
- Die Person, die die Pipeline modelliert, und die Person, die die Pipeline ausführt, können zwei verschiedene Benutzer mit unterschiedlichen Rollen sein.
- Wenn ein Entwickler mit der Rolle „vRealize Automation Code Stream-Executor“ die Pipeline modelliert, muss der Entwickler auch dieselbe vRealize Automation Cloud Assembly-Rolle wie die Person aufweisen, die die Cloud-Vorlage bereitgestellt hat. Die notwendige Rolle kann beispielsweise „vRealize Automation Cloud Assembly-Administrator“ lauten.

- Nur die Person, die die Pipeline modelliert, kann die Pipeline und die Bereitstellung erstellen, da sie über die Berechtigung verfügt.

So verwenden Sie ein API-Token in der Cloud-Vorlagenaufgabe:

- Die Person, die die Pipeline modelliert, kann ein API-Token an einen anderen Benutzer mit der Rolle „vRealize Automation Code Stream-Executor“ übergeben. Wenn der Executor die Pipeline dann ausführt, verwendet er das API-Token und die Anmeldedaten, die vom API-Token erstellt werden.
- Wenn ein Benutzer das API-Token in der Cloud-Vorlagenaufgabe eingibt, werden die Anmeldedaten erstellt, die für die Pipeline notwendig sind.
- Klicken Sie zum Verschlüsseln des API-Tokenwerts auf **Variable erstellen**.
- Wenn Sie keine Variable für das API-Token erstellen und es in der Cloud-Vorlagenaufgabe verwenden, wird der Wert des API-Tokens im Klartext angezeigt.

Führen Sie die folgenden Schritte aus, um Dollarbindungen auf Cloud-Vorlagenvariablen in einer Cloud-Vorlagenaufgabe anzuwenden.

Sie beginnen mit einer Cloud-Vorlage, für die Eigenschaften der Eingabevariablen definiert sind, wie z. B. `integerVar`, `stringVar`, `flavorVar`, `BooleanVar`, `objectVar` und `arrayVar`. Sie finden die definierten Image-Eigenschaften im Abschnitt `resources`. Die Eigenschaften im Code der Cloud-Vorlage ähneln unter Umständen Folgendem:

```
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}'

```

Sie können die Dollarzeichenvariablen (\$) für `image` und `flavor` verwenden. Beispiel:

```

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

```

Zum Verwenden einer Cloud-Vorlage in einer vRealize Automation Code Stream-Pipeline und Hinzufügen von Dollarbindungen gehen Sie nach diesen Schritten vor.

- 1 Klicken Sie in vRealize Automation Code Stream auf **Pipelines > Leere Arbeitsfläche**.
- 2 Fügen Sie der Pipeline eine Aufgabe vom Typ **Cloud-Vorlage** hinzu.
- 3 Wählen Sie in der Cloud-Vorlagenaufgabe für **Quelle der Cloud-Vorlage** die Option **Cloud Assembly-Cloud-Vorlagen** aus, geben Sie den Namen der Cloud-Vorlage ein und wählen Sie die Version der Cloud-Vorlage aus.
- 4 Beachten Sie, dass Sie ein API-Token eingeben können, das Anmeldedaten für die Pipeline liefert. Um eine Variable zu erstellen, die das API-Token in der Cloud-Vorlagenaufgabe verschlüsselt, klicken Sie auf **Variable erstellen**.
- 5 Beachten Sie in der angezeigten Tabelle **Parameter und Wert** die Parameterwerte. Der Standardwert für `flavor` ist `small` und der Standardwert für `image` ist `ubuntu`.
- 6 Angenommen, Sie müssen die Cloud-Vorlage in vRealize Automation Cloud Assembly ändern. Beispiel:
 - a Legen Sie den `flavor` so fest, dass er eine Eigenschaft vom Typ `array` verwendet. vRealize Automation Cloud Assembly lässt kommagetrennte Werte für `flavor` zu, wenn **array** als Typ verwendet wird.
 - b Klicken Sie auf **Bereitstellen**.
 - c Geben Sie auf der Seite „Bereitstellungstyp“ einen Namen für die Bereitstellung ein und wählen Sie die Version der Cloud-Vorlage aus.
 - d Auf der Seite „Bereitstellungseingaben“ können Sie einen oder mehrere Werte für Typ definieren.

- e Beachten Sie, dass die Bereitstellungseingaben alle im Code der Cloud-Vorlage definierten Variablen enthalten und im Code der Cloud-Vorlage definitionsgemäß angezeigt werden.
Beispiel: Integer Var, String Var, Flavor Var, Boolean Var, Object Var und Array Var. String Var und Flavor Var sind Zeichenfolgenwerte. Boolean Var ist ein Kontrollkästchen.
 - f Klicken Sie auf **Bereitstellen**.
- 7 Wählen Sie in vRealize Automation Code Stream die neue Version der Cloud-Vorlage aus und geben Sie Werte in der Tabelle **Parameter und Werte** ein. Cloud-Vorlagen unterstützen die folgenden Parametertypen, die vRealize Automation Code Stream-Bindungen mithilfe von Dollarzeichenvariablen aktivieren. Es bestehen geringfügige Unterschiede zwischen der Benutzeroberfläche der vRealize Automation Code Stream-Cloud-Vorlagenaufgabe und der Benutzeroberfläche der vRealize Automation Cloud Assembly-Cloud-Vorlage. Je nach Codierung einer Cloud-Vorlage in vRealize Automation Cloud Assembly ist die Eingabe von Werten in der Cloud-Vorlagenaufgabe in vRealize Automation Code Stream nicht zulässig.
- a Wenn in der Cloud-Vorlage der Typ als Zeichenfolge oder Array für **flavorVar** definiert wurde, geben Sie eine Zeichenfolge oder ein kommagetrenntes Wert-Array ein. Ein Beispiel-Array ähnelt **test, test**.
 - b Wählen Sie für **BooleanVar** im Dropdown-Menü die Option **true** oder **false** aus. Geben Sie alternativ zur Verwendung einer Variablenbindung den Wert **\$** ein und wählen Sie eine Variablenbindung aus der Liste

| Parameter | Value |
|------------|---|
| stringVar | raj |
| integerVar | 1 |
| flavorVar | medium |
| BooleanVar | \$ |
| objectVar | var |
| arrayVar | input comments requestBy executionIndex executionId executionUrl |

Output Parameter

name
description
Stage0
status
deploymentCriteria
deploymentId
deploymentName

aus.

- c Geben Sie für **objectVar** den Wert in geschweiften Klammern und Anführungszeichen in folgendem Format ein: **{ "bkix": "bkix": }**.
- d Der Wert **objectVar** wird an die Cloud-Vorlage übergeben und kann je nach Cloud-Vorlage auf verschiedene Arten verwendet werden. Dieser Wert ermöglicht ein Zeichenfolgenformat für ein JSON-Objekt, und Sie können Schlüssel-Wert-Paare in der Schlüssel-Wert-Tabelle als kommagetrennte Werte hinzufügen. Sie können Klartext für ein JSON-Objekt oder ein Schlüssel-Wert-Paar als normales Zeichenfolgenformat für JSON eingeben.

- e Geben Sie für **arrayVar** den kommagetrennten Eingabewert als Array in folgendem Format ein: `["1", "2"]`.
- 8 In der Pipeline können Sie einen Eingabeparameter an ein Array binden.
- a Klicken Sie auf die Registerkarte **Eingabe**.
 - b Geben Sie einen Namen für die Eingabe ein. Beispielsweise **arrayInput**.
 - c Klicken Sie in der Tabelle **Parameter und Wert** auf **arrayVar** und geben Sie `$ {input.arrayInput}` ein.
 - d Nachdem Sie die Pipeline gespeichert und aktiviert haben, müssen Sie bei Ausführung der Pipeline einen Eingabewert für das Array angeben. Geben Sie z. B. `["1", "2"]` ein und klicken Sie auf **Ausführen**.

Jetzt haben Sie sich in einer Cloud-Vorlagenaufgabe der vRealize Automation Code Stream-Pipeline mit der Verwendung von `$`-Variablenbindungen in einer Cloud-Vorlage vertraut gemacht.

Vorgehensweise zum Übergeben eines Parameters an eine Pipeline, während diese ausgeführt wird

Sie können der Pipeline Eingabeparameter hinzufügen, damit sie von vRealize Automation Code Stream an die Pipeline übergeben werden. Wenn die Pipeline dann ausgeführt wird, muss ein Benutzer den Wert für den Eingabeparameter eingeben. Wenn Sie Ausgabeparameter zur Pipeline hinzufügen, können die Pipeline-Aufgaben den Ausgabewert aus einer Aufgabe verwenden. vRealize Automation Code Stream bietet vielfältige Unterstützung für die Verwendung von Parametern, die sich nach Ihren Anforderungen an Pipelines richtet.

Wenn Sie einen Benutzer während der Ausführung einer Pipeline mit einer REST-Aufgabe zur Eingabe der URL des Git-Servers auffordern, können Sie die REST-Aufgabe an eine Git-Server-URL binden.

Zum Erstellen der Variablenbindung fügen Sie der REST-Aufgabe eine URL-Bindungsvariable hinzu. Wenn die Pipeline ausgeführt wird und die REST-Aufgabe erreicht, muss ein Benutzer die URL des Git-Servers eingeben. So erstellen Sie die Bindung:

- 1 Klicken Sie in Ihrer Pipeline auf die Registerkarte **Eingabe**.
- 2 Klicken Sie zum Einrichten des Parameters für **Parameter automatisch einfügen** auf **Git**.
Die Liste der Git-Parameter wird angezeigt und enthält **GIT_SERVER_URL**. Wenn Sie einen Standardwert für die Git-Server-URL verwenden müssen, bearbeiten Sie diesen Parameter.
- 3 Klicken Sie auf **Modell** und dann auf die REST-Aufgabe.
- 4 Geben Sie auf der Registerkarte **Aufgabe** im Bereich **URL** den Wert `$` ein und wählen Sie dann **Eingabe** und **GIT_SERVER_URL** aus.

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|

Agent endpoint

Headers

Output Parameters

status responseHeaders responseBody responseJson responseCode

Der Eintrag ähnelt folgendem Eintrag: `$(input.GIT_SERVER_URL)`

- Um die Integrität der Variablenbindung für die Aufgabe zu überprüfen, klicken Sie auf **Aufgabe validieren**.

vRealize Automation Code Stream gibt an, dass die Aufgabe erfolgreich validiert wurde.

- Wenn die REST-Aufgabe von der Pipeline ausgeführt wird, muss ein Benutzer die URL des Git-Servers eingeben. Andernfalls wird die Aufgabe immer weiter ausgeführt.

Vorgehensweise zum Binden zweier Pipeline-Aufgaben durch Erstellen von Eingabe- und Ausgabeparametern

Beim Binden von Aufgaben fügen Sie der Eingabekonfiguration der empfangenden Aufgabe eine Bindungsvariable hinzu. Bei der Ausführung der Pipeline ersetzt ein Benutzer die Bindungsvariable dann durch die erforderliche Eingabe.

Um Pipeline-Aufgaben miteinander zu verbinden, verwenden Sie die Dollarzeichen-Variable (\$) in den Eingabe- und Ausgabeparametern. Die entsprechende Vorgehensweise finden Sie in diesem Beispiel.

Angenommen, Sie benötigen Ihre Pipeline zum Aufrufen einer URL in einer REST-Aufgabe und Ausgeben einer Antwort. Zum Aufrufen der URL und zur Ausgabe der Antwort fügen Sie sowohl Eingabe- als auch Ausgabeparameter in Ihre REST-Aufgabe ein. Weiterhin benötigen Sie einen Benutzer, der die Aufgabe genehmigen kann, und schließen eine Aufgabe vom Typ

„Benutzervorgänge“ für einen anderen Benutzer ein, der die Aufgabe während der Ausführung der Pipeline genehmigen kann. Dieses Beispiel zeigt, wie Sie Ausdrücke in den Eingabe- und Ausgabeparametern verwenden und die Pipeline dazu veranlassen, auf die Genehmigung der Aufgabe zu warten.

- 1 Klicken Sie in Ihrer Pipeline auf die Registerkarte **Eingabe**.

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

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

- 2 Belassen Sie die Einstellung **Parameter automatisch einfügen** bei **Keine**.
- 3 Klicken Sie auf **Hinzufügen** und geben Sie den Parameternamen, den Wert und die Beschreibung ein. Klicken Sie anschließend auf **OK**. Beispiel:
 - a Geben Sie einen URL-Namen ein.
 - b Geben Sie den Wert ein: {Stage0.Task3.input.http://www.docs.vmware.com}
 - c Geben Sie eine Beschreibung ein.
- 4 Klicken Sie auf die Registerkarte **Ausgabe** und dann auf **Hinzufügen**. Geben Sie anschließend den Namen und die Zuordnung des Ausgabeparameters ein.

Add Pipeline Output Parameter

Name *

Reference \$ *

responseHeaders
 responseBody
 responseJson
 responseCode

- a Geben Sie einen eindeutigen Namen für den Ausgabeparameter ein.
- b Klicken Sie auf den Bereich **Referenz** und geben Sie \$ ein.
- c Geben Sie die Ausgabezuordnung der Aufgabe ein, indem Sie die verfügbaren Optionen auswählen. Wählen Sie **Stage0**, **Task3**, **output** und **responseCode** aus. Klicken Sie dann auf **OK**.

rest-ix-1
Enabled
ACTIONS

Workspace
Input
Model
Output

Output Parameters ⓘ

ADD

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

- 5 Speichern Sie die Pipeline.
- 6 Klicken Sie im Menü **Aktionen** auf **Ausführen**.
- 7 Klicken Sie auf **Aktionen > Ausführungen anzeigen**.
- 8 Klicken Sie auf die Pipeline-Ausführung und untersuchen Sie die von Ihnen definierten Eingabe- und Ausgabeparameter.

The screenshot displays the vRealize Automation interface for a pipeline execution. At the top, the pipeline is named 'rest-ix-1 #2' and is in a 'WAITING' state. Below this, a progress bar shows 'Stage0' as the active stage, with 'Task2' and 'Task3' listed below it. The main section provides detailed information about the execution:

- Project:** chim
- Execution:** rest-ix-1 #2
- Status:** WAITING (Stage0.Task2: Execution Waiting for User Action.)
- Updated By:**
- Executed By:** kerm@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 Klicken Sie zur Genehmigung der Pipeline auf **Benutzervorgänge** und zeigen Sie die Liste der Genehmigungen auf der Registerkarte **Aktive Elemente** an. Oder bleiben Sie unter „Ausführungen“ und klicken Sie auf die Aufgabe und dann auf **Genehmigen**.
- 10 Zum Aktivieren der Schaltflächen **Genehmigen** und **Ablehnen** klicken Sie auf das Kontrollkästchen neben der Ausführung.
- 11 Erweitern Sie zur Anzeige der Details das Dropdown-Menü.
- 12 Klicken Sie zum Genehmigen der Aufgabe auf **GENEHMIGEN**, geben Sie einen Grund ein und klicken Sie auf **OK**.

User Operations GUIDED SETUP

Active Items Inactive Items

✓ APPROVE ✗ REJECT

☐ Index# Execution

☑ User Operation #f0d252

Request Details

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

APPROVE REJECT VIEW DASHBOARD

- 13 Klicken Sie auf **Ausführungen** und verfolgen Sie die weitere Ausführung der Pipeline.

Executions 3,347 items GUIDED SETUP

+ NEW EXECUTION

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

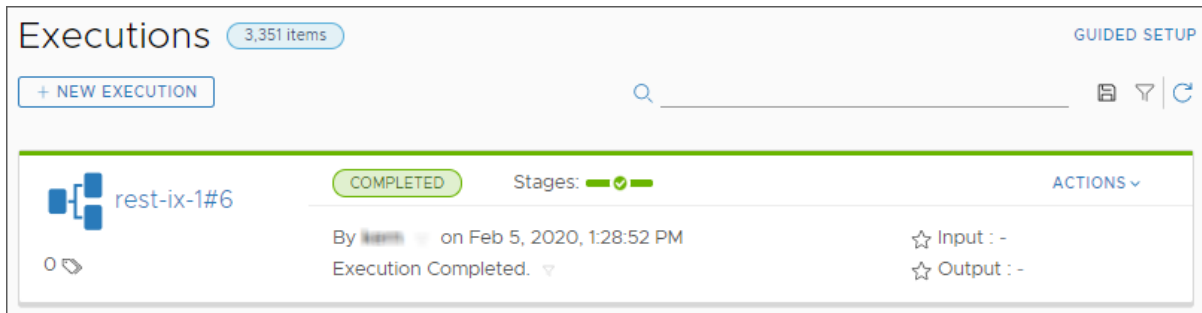
By kern on Feb 4, 2020, 3:41:05 PM

☆ Input : -

☆ Output : -

Comments: Testing

- 14 Beheben Sie bei einem Ausfall der Pipeline alle Fehler, speichern Sie die Pipeline und führen Sie sie erneut aus.



Vorgehensweise zum Abrufen von Informationen zu Variablen und Ausdrücken

Weitere Informationen zur Verwendung von Variablen und Ausdrücken beim Binden von Pipeline-Aufgaben finden Sie unter [Welche Variablen und Ausdrücke kann ich beim Binden von Pipelineaufgaben in vRealize Automation Code Stream verwenden?](#).

Weitere Informationen zur Verwendung der Ausgabe einer Pipeline-Aufgabe mit einer Variablenbindung für Bedingungen finden Sie unter [Vorgehensweise zum Verwenden von Variablenbindungen in einer Bedingungs Aufgabe zum Ausführen oder Anhalten einer Pipeline in vRealize Automation Code Stream](#).

Vorgehensweise zum Verwenden von Variablenbindungen in einer Bedingungs Aufgabe zum Ausführen oder Anhalten einer Pipeline in vRealize Automation Code Stream

Sie können festlegen, dass die Ausgabe einer Aufgabe in Ihrer Pipeline bestimmt, ob die Pipeline basierend auf einer von Ihnen bereitgestellten Bedingung ausgeführt oder angehalten wird. Wenn die Ausführung der Pipeline basierend auf der Aufgabenausgabe erfolgreich sein oder fehlschlagen soll, verwenden Sie die Aufgabe „Bedingung“.

Sie verwenden die Aufgabe **Bedingung** als Entscheidungspunkt in Ihrer Pipeline. Sie können alle Eigenschaften in der Pipeline, in den Phasen und in den Aufgaben auswerten, indem Sie die Aufgabe „Bedingung“ mit einem von Ihnen bereitgestellten Bedingungs Ausdruck kombinieren.

Das Ergebnis der Aufgabe „Bedingung“ bestimmt, ob die nächste Aufgabe in der Pipeline ausgeführt wird.

- Wenn die Bedingung „true“ ergibt, kann die Pipeline-Ausführung fortgesetzt werden.
- Ergibt die Bedingung „false“, wird die Pipeline angehalten.

Der Ausgabewert einer Aufgabe kann als Eingabe für die nächste Aufgabe verwendet werden. Dazu werden die Aufgaben mit einer Aufgabe „Bedingung“ aneinander gebunden. Beispiele dazu, wie dies funktioniert, finden Sie unter [Vorgehensweise zum Verwenden von Variablenbindungen in vRealize Automation Code Stream-Pipelines](#).

Tabelle 3-7. Beziehung zwischen der Aufgabe „Bedingung“ sowie ihrem Bedingungsausdruck und der Pipeline

| Aufgabe „Bedingung“ | Hat Auswirkung auf | Funktionsweise |
|---------------------|---------------------------------|--|
| Aufgabe „Bedingung“ | Pipeline | Die Aufgabe Bedingung bestimmt, ob die Pipeline an diesem Punkt ausgeführt oder angehalten wird – je nachdem, ob die Aufgabenausgabe „true“ oder „false“ lautet. |
| Bedingungsausdruck | Ausgabe der Aufgabe „Bedingung“ | <p>Wenn die Pipeline ausgeführt wird, erzeugt der Bedingungsausdruck, den Sie in die Aufgabe Bedingung aufnehmen, den Ausgabestatus „true“ oder „false“. Beispiel: Ein Bedingungsausdruck kann erfordern, dass der Ausgabestatus der Bedingung Abgeschlossen lautet oder dass die Build-Nummer 74 verwendet wird.</p> <p>Der Bedingungsausdruck wird auf der Registerkarte „Aufgabe“ in der Aufgabe „Bedingung“ angezeigt.</p> |

Aufgabe :Task2 Benachrichtigungen Rollback

Aufgabenname * Task2

Typ * Condition

Aufgabe „Bedingung“

Bedingung \$ Wenn die Dollar-Bindung eine Zeichenfolge ergibt, mit ' ' oder " " einschließen

Ausgabe

Bedingter Ausdruck

Ein Ausdruck, dessen Auswertung **wahr** oder **falsch** zurückgibt

Beispiel:

```

${stage1.task1.output.status} == "COMPLETED"
|| ${input.buildNumber} == 74

```

Unterstützte Syntax:

Wenn die Dollar-Bindung eine Zeichenfolge ergibt, mit ' ' oder " " einschließen

| Typ | Beispiel |
|--------------------------|--|
| Pipeline-Variablen | \$(input.changeSetNumber) (numerische Bindung) oder \$(input.changeSetOwner) (Zeichenfolgenbindung) |
| Aufgabenausgabevariablen | \$(stage1.task1.output.responseCode) (numerische Bindung) oder \$(stage1.task1.output.status) |

Die Aufgabe **Bedingung** unterscheidet sich in der Funktion und im Verhalten von der Einstellung **Bei Bedingung** in anderen Aufgabentypen.

Aufgabe :Deploy Phase 1 Benachrichtigungen Rollback **AUFGABE VALIDIEREN**

Aufgabenname * Deploy Phase 1

Typ * Kubernetes

Bei Fehler fortfahren ☐

Aufgabe ausführen ☐ Immer ☒ Bei Bedingung

Bedingung \$

In anderen Aufgabentypen legt **Bei Bedingung** fest, ob die aktuelle Aufgabe anstelle nachfolgender Aufgaben ausgeführt wird. Das Ergebnis basiert darauf, ob ihr Vorbedingungsausdruck mit „true“ oder „false“ ausgewertet wird. Wenn die Pipeline ausgeführt wird, erzeugt der Bedingungsausdruck für die Einstellung **Bei Bedingung** den Ausgabestatus „true“ oder „false“ für die aktuelle Aufgabe. Die Einstellung **Bei Bedingung** wird auf der Registerkarte „Aufgabe“ mit einem eigenen Bedingungsausdruck angezeigt.

In diesem Beispiel wird die Aufgabe „Bedingung“ verwendet.

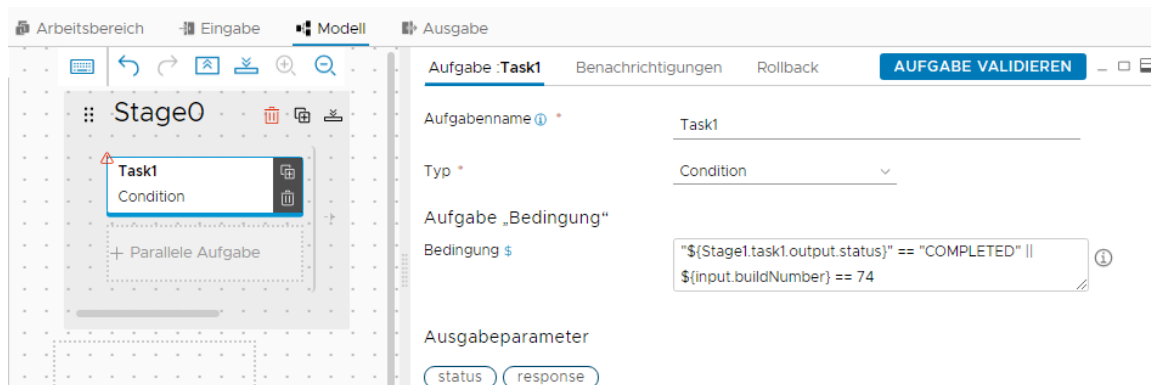
Voraussetzungen

- Stellen Sie sicher, dass eine Pipeline vorhanden ist und Phasen und Aufgaben umfasst.

Verfahren

- 1 Bestimmen Sie in Ihrer Pipeline den Entscheidungspunkt, an dem die Aufgabe „Bedingung“ angezeigt werden muss.
- 2 Fügen Sie die Aufgabe „Bedingung“ vor der Aufgabe hinzu, die vom Status – erfolgreich oder fehlgeschlagen – abhängt.
- 3 Fügen Sie einen Bedingungsausdruck zur Aufgabe „Bedingung“ hinzu.

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



- 4 Validieren Sie die Aufgabe.
- 5 Speichern Sie die Pipeline, aktivieren Sie sie und führen Sie sie aus.

Ergebnisse

Überwachen Sie die Pipeline-Ausführungen und beachten Sie, ob die Pipeline weiterhin ausgeführt oder bei der Aufgabe „Bedingung“ angehalten wird.

Nächste Schritte

Wenn Sie ein Rollback einer Pipeline-Bereitstellung vornehmen, können Sie auch die Aufgabe „Bedingung“ verwenden. In einer Rollback-Pipeline hilft die Aufgabe „Bedingung“ beispielsweise vRealize Automation Code Stream, basierend auf dem Bedingungsausdruck einen Pipeline-Fehler zu markieren. Darüber hinaus kann sie einen einzelnen Rollback-Ablauf für verschiedene Fehlertypen auslösen.

Informationen zum Rollback einer Bereitstellung finden Sie unter [Vorgehensweise zum Rollback meiner Bereitstellung in vRealize Automation Code Stream](#).

Welche Variablen und Ausdrücke kann ich beim Binden von Pipelineaufgaben in vRealize Automation Code Stream verwenden?

Mit Variablen und Ausdrücken können Sie Eingabe- und Ausgabeparameter in Ihren Pipeline-Aufgaben verwenden. Die von Ihnen eingegebenen Parameter binden die Pipeline-Aufgabe an eine oder mehrere Variablen, Ausdrücke oder Bedingungen und bestimmen das Verhalten der Pipeline, wenn sie ausgeführt wird.

Pipelines können einfache oder komplexe Softwarebereitstellungslösungen ausführen

Bei der Bindung von Pipeline-Aufgaben können Sie Standardausdrücke und komplexe Ausdrücke aufnehmen. Dies führt dazu, dass Ihre Pipeline einfache oder komplexe Softwarebereitstellungslösungen ausführen kann.

Um die Parameter in Ihrer Pipeline zu erstellen, klicken Sie auf die Registerkarte **Eingabe** oder **Ausgabe** und fügen Sie eine Variable hinzu, indem Sie das Dollarzeichen \$ und einen Ausdruck eingeben. Beispielsweise wird dieser Parameter als Aufgabeneingabe verwendet, die eine URL aufruft: `${Stage0.Task3.input.URL}`.

Das Format für Variablenbindungen verwendet Syntaxkomponenten, die als „scopes“ und „keys“ bezeichnet werden. `SCOPE` definiert den Kontext als Eingabe oder Ausgabe, und `KEY` definiert die Details. Im Parameterbeispiel `${Stage0.Task3.input.URL}` ist `input` der `SCOPE` und der `KEY` die `URL`.

Die Ausgabeigenschaften einer beliebigen Aufgabe können zu einer beliebigen Anzahl an geschachtelten Ebenen mit variabler Bindung aufgelöst werden.

Weitere Informationen zur Verwendung von Variablenbindungen in Pipelines finden Sie unter [Vorgehensweise zum Verwenden von Variablenbindungen in vRealize Automation Code Stream-Pipelines](#).

Verwenden von Dollarausdrücken mit Geltungsbereichen und Schlüsseln zum Binden von Pipeline-Aufgaben

Sie können Pipeline-Aufgaben aneinander binden, indem Sie Ausdrücke in Variablen mit Dollarzeichen verwenden. Sie geben Ausdrücke als `${SCOPE.KEY.<PATH>}` ein.

Zur Bestimmung des Verhaltens einer Pipeline-Aufgabe zu bestimmen, wird in jedem Ausdruck `SCOPE` als der von vRealize Automation Code Stream verwendete Kontext verwendet. `SCOPE` sucht nach einem `KEY`, der die Einzelheiten der von der Aufgabe durchgeführten Aktion definiert. Wenn der Wert für `KEY` ein verschachteltes Objekt ist, können Sie einen optionalen `PATH` bereitstellen.

In diesen Beispielen werden `SCOPE` und `KEY` beschrieben und es wird gezeigt, wie Sie sie in Ihrer Pipeline verwenden.

Tabelle 3-8. Verwenden von SCOPE und KEY

| SCOPE | Zweck des Ausdrucks und Beispiel | KEY | Verwenden von SCOPE und KEY in Ihrer Pipeline |
|--------|---|-----------------------------|--|
| input | Eingabeeigenschaften einer Pipeline: <code>\${input.input1}</code> | Name der Eingabeeigenschaft | <p>Um auf die Eingabeeigenschaft einer Pipeline in einer Aufgabe zu verweisen, verwenden Sie folgendes Format:</p> <pre>tasks: mytask: type: REST input: url: \${input.url} action: get</pre> <pre>input: url: https://www.vmware.com</pre> |
| output | Ausgabeeigenschaften einer Pipeline: <code>\${output.output1}</code> | Name der Ausgabeeigenschaft | <p>Um auf eine Ausgabeeigenschaft zum Senden einer Benachrichtigung zu verweisen, verwenden Sie das folgende Format:</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> |

Tabelle 3-8. Verwenden von SCOPE und KEY (Fortsetzung)

| SCOPE | Zweck des Ausdrucks und Beispiel | KEY | Verwenden von SCOPE und KEY in Ihrer Pipeline |
|------------|--|---|---|
| task input | <p>Eingabe für eine Aufgabe:</p> <pre>\$ {MY_STAGE.MY_TASK.input. SOMETHING}</pre> | Gibt die Eingabe einer Aufgabe in einer Benachrichtigung an | <p>Wenn ein Jenkins-Auftrag gestartet wird, kann er auf den Namen des Auftrags verweisen, der von der Aufgabeneingabe ausgelöst wurde. Senden Sie in diesem Fall eine Benachrichtigung in folgendem Format:</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> |

Tabelle 3-8. Verwenden von SCOPE und KEY (Fortsetzung)

| SCOPE | Zweck des Ausdrucks und Beispiel | KEY | Verwenden von SCOPE und KEY in Ihrer Pipeline |
|--------------------|---|--|--|
| task output | Ausgabe einer Aufgabe: \$ {MY_STAGE.MY_TASK.output .SOMETHING} | Gibt die Ausgabe einer Aufgabe in einer nachfolgenden Aufgabe an | Um auf die Ausgabe der Pipeline-Aufgabe 1 in Aufgabe 2 zu verweisen, verwenden Sie folgendes Format: <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> |
| var | Variable: \${var.myVariable} | Auf Variable in einem Endpoint verweisen | Verwenden Sie dieses Format, um auf eine geheime Variable in einem Endpoint für ein Kennwort zu verweisen: <pre>--- project: MyProject kind: ENDPOINT name: MyJenkinsServer type: jenkins properties: url: https:// jenkins.example.com username: jenkinsUser password: \$ {var.jenkinsPasswor d}</pre> |

Tabelle 3-8. Verwenden von SCOPE und KEY (Fortsetzung)

| SCOPE | Zweck des Ausdrucks und Beispiel | KEY | Verwenden von SCOPE und KEY in Ihrer Pipeline |
|---------------------|--|---|---|
| var | Variable: <code>\${var.myVariable}</code> | Auf eine Variable in einer Pipeline verweisen | Verwenden Sie dieses Format, um auf eine Variable in einer Pipeline-URL zu verweisen: <pre>tasks: task1: type: REST input: action: get url: \$ {var.MY_SERVER_URL}</pre> |
| task status | Status einer Aufgabe: <pre>\$ {MY_STAGE.MY_TASK.status } \$ {MY_STAGE.MY_TASK.status Message}</pre> | | |
| stage status | Status einer Phase: <pre>\${MY_STAGE.status} \$ {MY_STAGE.statusMessage}</pre> | | |

Standardausdrücke

Sie können Variablen mit Ausdrücken in Ihrer Pipeline verwenden. Diese Übersicht enthält die Standardausdrücke, die Sie verwenden können.

| Ausdruck | Beschreibung |
|-------------------------------|--|
| <code>\${comments}</code> | Kommentare, die bei der Pipeline-Ausführungsanforderung bereitgestellt werden. |
| <code>\${duration}</code> | Dauer der Pipeline-Ausführung. |
| <code>\${endTime}</code> | Endzeit der Pipeline-Ausführung in UTC, sofern abgeschlossen. |
| <code>\${executedOn}</code> | Entspricht der Startzeit – Startzeit der Pipeline-Ausführung in UTC. |
| <code>\${executionId}</code> | ID der Pipeline-Ausführung. |
| <code>\${executionUrl}</code> | URL, die zur Pipeline-Ausführung in der Benutzeroberfläche navigiert. |
| <code>\${name}</code> | Name der Pipeline. |
| <code>\${requestBy}</code> | Name des Benutzers, der die Ausführung angefordert hat. |
| <code>\${stageName}</code> | Name der aktuellen Phase, wenn sie im „scope“ einer Phase verwendet wird. |

| Ausdruck | Beschreibung |
|--------------------------------|--|
| <code>\${startTime}</code> | Startzeit der Pipeline-Ausführung in UTC. |
| <code>\${status}</code> | Status der Ausführung. |
| <code>\${statusMessage}</code> | Statusmeldung der Pipeline-Ausführung. |
| <code>\${taskName}</code> | Name der aktuellen Aufgabe, wenn sie bei der einer Aufgabeneingabe oder Benachrichtigung verwendet wird. |

Verwenden von SCOPE und KEY in den Pipeline-Aufgaben

Sie können Ausdrücke mit allen unterstützten Pipeline-Aufgaben verwenden. In diesen Beispielen wird gezeigt, wie Sie `SCOPE` und `KEY` definieren und die Syntax bestätigen. In den Codebeispielen werden `MY_STAGE` und `MY_TASK` als Bereitstellungs- und Aufgabennamen der Pipeline verwendet.

Weitere Informationen zu den verfügbaren Aufgaben finden Sie unter [In vRealize Automation Code Stream verfügbare Aufgabentypen](#).

Tabelle 3-9. Weitergeben von Aufgaben

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|------------------------|-------|--|--|
| Benutzervorgang | | | |
| | Input | <p><code>summary</code>: Zusammenfassung der Anforderung für den Benutzervorgang</p> <p><code>description</code>: Beschreibung der Anforderung für den Benutzervorgang</p> <p><code>approvers</code>: Liste der Genehmiger-E-Mail-Adressen, wobei jeder Eintrag eine mit Komma getrennte Variable sein kann (bzw. durch Semikolon getrennte E-Mail-Adressen)</p> <p><code>approverGroups</code>: Liste der Genehmiger-Gruppenadressen für die Plattform und Identität</p> <p><code>sendemail</code>: Sendet auf Anfrage oder Antwort optional eine E-Mail-Benachrichtigung, wenn auf „true“ festgelegt</p> <p><code>expirationInDays</code>: Anzahl der Tage, die die Ablaufzeit der Anforderung darstellt</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> |

Tabelle 3-9. Weitergeben von Aufgaben (Fortsetzung)

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|------------------|--------|--|---|
| | Output | index: sechsstellige hexadezimale Zeichenfolge, die die Anforderung darstellt respondedBy: Kontoname der Person, die den Benutzervorgang genehmigt bzw. abgelehnt hat respondedByEmail: E-Mail-Adresse der Person, die geantwortet hat comments: Während der Antwort bereitgestellte Kommentare | <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> |
| Bedingung | | | |
| | Input | condition: Auszuwertende Bedingung. Wenn die Bedingung „true“ zurückgibt, wird die Aufgabe als abgeschlossen markiert, während die Aufgabe bei anderen Antworten fehlschlägt | <pre>\${MY_STAGE.MY_TASK.input.condition}</pre> |
| | Output | result: Ergebnis bei Auswertung | <pre>\${MY_STAGE.MY_TASK.output.response}</pre> |

Tabelle 3-10. Pipeline-Aufgaben

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|-----------------|--------|--|--|
| Pipeline | | | |
| | Input | name: Name der auszuführenden Pipeline inputProperties: Eingabeeigenschaften, die an die verschachtelte Pipeline-Ausführung übergeben werden | <pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.inputProperties} # Siehe alle Eigenschaften \$ {MY_STAGE.MY_TASK.input.inputProperties.input1} # Siehe Wert von input1 </pre> |
| | Output | executionStatus: Status der Pipeline-Ausführung executionIndex: Index der Pipeline-Ausführung outputProperties: Ausgabeeigenschaften der Pipeline-Ausführung | <pre> \${MY_STAGE.MY_TASK.output.executionStatus} \${MY_STAGE.MY_TASK.output.executionIndex} \${MY_STAGE.MY_TASK.output.outputProperties} # Siehe alle Eigenschaften \$ {MY_STAGE.MY_TASK.output.outputProperties.output1} # Siehe Wert von output1 </pre> |

Tabelle 3-11. Automatisieren fortlaufender Integrationsaufgaben

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|-------------------|--------|---|--|
| CI | | | |
| | Input | <p>steps: Eine Reihe von Zeichenfolgen, die auszuführende Befehle darstellen</p> <p>export: Beizubehaltende Umgebungsvariablen nach dem Ausführen der Schritte</p> <p>artifacts: Pfade von Artefakten, die im gemeinsam genutzten Pfad beibehalten werden sollen</p> <p>process: Reihe von Konfigurationselementen für die Verarbeitung mit JUnit, JaCoCo, Checkstyle, FindBugs</p> | <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 } # Siehe Pfad der ersten Konfiguration </pre> |
| | Output | <p>exports: Schlüssel-Wert-Paar, das die exportierten Umgebungsvariablen aus der Eingabe darstellt export</p> <p>artifacts: Pfad der erfolgreich beibehaltenen Artefakte</p> <p>processResponse: Satz verarbeiteter Ergebnisse für die Eingabe process</p> | <pre> \${MY_STAGE.MY_TASK.output.exports} # Siehe alle Exporte \$ {MY_STAGE.MY_TASK.output.exports.myvar} # Siehe Wert von myvar \${MY_STAGE.MY_TASK.output.artifacts} \$ {MY_STAGE.MY_TASK.output.processResponse} \$ {MY_STAGE.MY_TASK.output.processResponse[0].result} # Ergebnis der ersten Prozesskonfiguration </pre> |
| Benutzerdefiniert | | | |
| | Input | <p>name: Name der benutzerdefinierten Integration</p> <p>version: Eine Version der benutzerdefinierten Integration, freigegeben oder veraltet</p> <p>properties: An die benutzerdefinierte Integration zu sendende Eigenschaften</p> | <pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.version} \${MY_STAGE.MY_TASK.input.properties} # Siehe alle Eigenschaften \$ {MY_STAGE.MY_TASK.input.properties.property1} # Siehe Wert von property1 </pre> |
| | Output | <p>properties: Ausgabeeigenschaften aus der Antwort der benutzerdefinierten Integration</p> | <pre> \${MY_STAGE.MY_TASK.output.properties} # Siehe alle Eigenschaften \$ {MY_STAGE.MY_TASK.output.properties.property1} # Siehe Wert von property1 </pre> |

Tabelle 3-12. Automatisieren kontinuierlicher Bereitstellungsaufgaben: Cloud-Vorlagen

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|---------------|-------|--|---|
| Cloud-Vorlage | | | |
| | Input | <p>action: Hierzu gehören createDeployment, updateDeployment, deleteDeployment, rollbackDeployment</p> <p>blueprintInputParams: wird für die Aktionen create deployment und update deployment verwendet</p> <p>allowDestroy: Maschinen können bei der Bereitstellungsaktualisierung gelöscht werden.</p> <p>CREATE_DEPLOYMENT</p> <ul style="list-style-type: none"> ■ blueprintName: Name der Cloud-Vorlage ■ blueprintVersion: Version der Cloud-Vorlage <p>ODER</p> <ul style="list-style-type: none"> ■ fileUrl: URL der Remote-Cloud-Vorlagen-YAML nach Auswahl eines GIT-Servers. <p>UPDATE_DEPLOYMENT</p> <p>Alle der folgenden Kombinationen:</p> <ul style="list-style-type: none"> ■ blueprintName: Name der Cloud-Vorlage ■ blueprintVersion: Version der Cloud-Vorlage <p>ODER</p> <ul style="list-style-type: none"> ■ fileUrl: URL der Remote-Cloud-Vorlagen-YAML nach Auswahl eines GIT-Servers. <p>-----</p> <ul style="list-style-type: none"> ■ deploymentId: ID der Bereitstellung <p>ODER</p> <ul style="list-style-type: none"> ■ deploymentName: Name der Bereitstellung <p>-----</p> | |

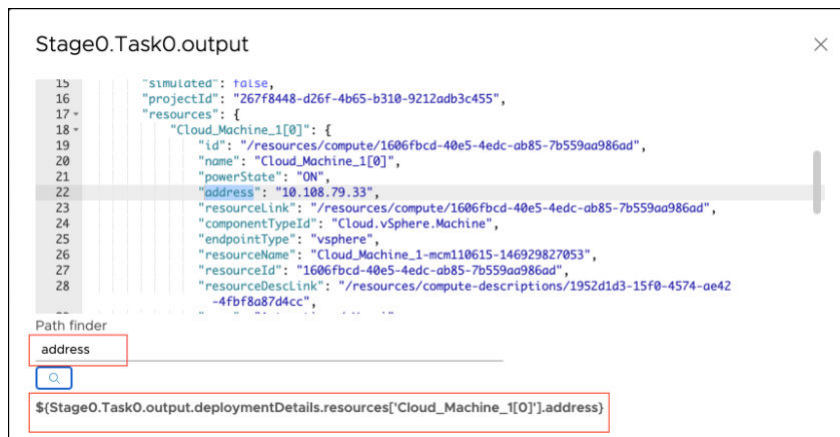
Tabelle 3-12. Automatisieren kontinuierlicher Bereitstellungsaufgaben: Cloud-Vorlagen (Fortsetzung)

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|---------|--------|--|--|
| | | <p>DELETE_DEPLOYMENT</p> <ul style="list-style-type: none"> ■ deploymentId: ID der Bereitstellung <p>ODER</p> <ul style="list-style-type: none"> ■ deploymentName: Name der Bereitstellung <p>ROLLBACK_DEPLOYMENT</p> <p>Alle der folgenden Kombinationen:</p> <ul style="list-style-type: none"> ■ deploymentId: ID der Bereitstellung <p>ODER</p> <ul style="list-style-type: none"> ■ deploymentName: Name der Bereitstellung <p>-----</p> <ul style="list-style-type: none"> ■ blueprintName: Name der Cloud-Vorlage ■ rollbackVersion: Version, auf die das Rollback durchgeführt werden soll | |
| | Output | | <p>Parameter, die an andere Aufgaben oder an die Ausgabe einer Pipeline gebunden werden können:</p> <ul style="list-style-type: none"> ■ Der Bereitstellungsname kann als <code>\${Stage0.Task0.output.deploymentName}</code> aufgerufen werden ■ Auf die Bereitstellungs-ID kann als <code>\${Stage0.Task0.output.deploymentId}</code> zugegriffen werden ■ Die Bereitstellungsdetails sind ein komplexes Objekt, und auf interne Details kann mithilfe der JSON-Ergebnisse zugegriffen werden. <p>Für den Zugriff auf eine beliebige Eigenschaft verwenden Sie den Punkt-Operator, um der JSON-Hierarchie zu folgen. Um beispielsweise auf die Adresse der Ressource <code>Cloud_Machine_1[0]</code> zuzugreifen, lautet die <code>\$</code>-Bindung:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}</pre> <p>Ähnlich gilt für die Konfiguration die <code>\$</code>-Bindung:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}</pre> |

Tabelle 3-12. Automatisieren kontinuierlicher Bereitstellungsaufgaben: Cloud-Vorlagen (Fortsetzung)

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|---------|-------|-----|---|
| | | | <p>In der vRealize Automation Code Stream-Benutzeroberfläche können Sie die \$-Bindungen für jede Eigenschaft abrufen.</p> <ol style="list-style-type: none"> 1 Klicken Sie im Eigenschaftsbereich der Aufgabenausgabe auf JSON-AUSGABE ANZEIGEN. 2 Um die \$-Bindung zu finden, geben Sie eine beliebige Eigenschaft ein. 3 Klicken Sie auf das Suchsymbol, das die entsprechende \$-Bindung anzeigt. |

JSON-Beispielausgabe:



Beispielobjekt für Bereitstellungsdetails:

```

{
  "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"
}

```

Tabelle 3-13. Automatisieren kontinuierlicher Bereitstellungsaufgaben: Kubernetes

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|------------|--------|---|---|
| Kubernetes | | | |
| | Input | <p>action: Hierzu gehören GET, CREATE, APPLY, DELETE, ROLLBACK</p> <ul style="list-style-type: none"> ■ timeout: Gesamtzahl der Zeitüberschreitungen für eine beliebige Aktion ■ filterByLabel: Zusätzliche Kennzeichnung zum Filtern für Aktion GET mithilfe von K8S labelSelector: <p>GET, CREATE, DELETE, APPLY</p> <ul style="list-style-type: none"> ■ yaml: Inline-YAML zum Verarbeiten und Senden an Kubernetes ■ parameters: KEY, VALUE-Paar - Ersetzen von \$\$KEY durch VALUE im Eingabebereich der Inline-YAML ■ filePath: Relativer Pfad des SCM Git-Endpoints (falls angegeben), aus dem die YAML abgerufen wird ■ scmConstants: KEY, VALUE-Paar - Ersetzen von \$\$ {KEY} durch VALUE in der über SCM abgerufenen YAML: ■ continueOnConflict: Wenn diese Option auf „True“ festgelegt ist, wird die Aufgabe bei einer bereits vorhandenen Ressource fortgesetzt. <p>ROLLBACK</p> <ul style="list-style-type: none"> ■ resourceType: Ressourcentyp für das Rollback ■ resourceName: Ressourcenname für das Rollback ■ namespace: Namespace, in dem das Rollback durchgeführt werden muss ■ revision: Revision, auf die das Rollback erfolgen soll | <pre> \${MY_STAGE.MY_TASK.input.action} #Bestimmt die durchzuführende Aktion. \${MY_STAGE.MY_TASK.input.timeout} \${MY_STAGE.MY_TASK.input.filterByLabel} \${MY_STAGE.MY_TASK.input.yaml} \${MY_STAGE.MY_TASK.input.parameters} \${MY_STAGE.MY_TASK.input.filePath} \${MY_STAGE.MY_TASK.input.scmConstants} \$ {MY_STAGE.MY_TASK.input.continueOnConflict} \${MY_STAGE.MY_TASK.input.resourceType} \${MY_STAGE.MY_TASK.input.resourceName} \${MY_STAGE.MY_TASK.input.namespace} \${MY_STAGE.MY_TASK.input.revision} </pre> |
| | Output | <p>response: Erfasst die vollständige Antwort</p> <p>response.<RESOURCE>: Ressource entspricht configMaps, Bereitstellungen, Endpoints, Dateieingängen, Jobs, Namespaces, Pods, replicaSets, replicationControllers, geheimen Schlüsseln, Diensten, statefulSets, Knoten, loadBalancers.</p> <p>response.<RESOURCE>.<KEY>: Schlüssel entspricht entweder apiVersion, Art, Metadaten oder Spezifikation</p> | <pre> \${MY_STAGE.MY_TASK.output.response} \${MY_STAGE.MY_TASK.output.response.} </pre> |

Tabelle 3-14. Integrieren von Entwicklungs-, Test- und Bereitstellungsanwendungen

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|----------------|--------|---|--|
| Bamboo | | | |
| | Input | plan: Name des Plans planKey: Planschlüssel variables: Variablen, die an den Plan übergeben werden parameters: Parameter, die an den Plan übergeben werden | <code>\${MY_STAGE.MY_TASK.input.plan}</code> <code>\${MY_STAGE.MY_TASK.input.planKey}</code> <code>\${MY_STAGE.MY_TASK.input.variables}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code> # Siehe alle Parameter <code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # Siehe Wert von param1 |
| | Output | resultUrl: URL des resultierenden Build buildResultKey: Schlüssel des resultierenden Build buildNumber: Build-Nummer buildTestSummary: Übersicht über die ausgeführten Tests successfulTestCount: Testergebnis übergeben failedTestCount: Testergebnis fehlgeschlagen skippedTestCount: Testergebnis übersprungen artifacts: Artefakte aus dem Build | <code>\${MY_STAGE.MY_TASK.output.resultUrl}</code> <code>\${MY_STAGE.MY_TASK.output.buildResultKey}</code> <code>\${MY_STAGE.MY_TASK.output.buildNumber}</code> <code>\${MY_STAGE.MY_TASK.output.buildTestSummary}</code> # Siehe alle Ergebnisse <code>\${MY_STAGE.MY_TASK.output.successfulTestCount}</code> # Siehe spezifische Testanzahl <code>\${MY_STAGE.MY_TASK.output.buildNumber}</code> |
| Jenkins | | | |
| | Input | job: Name des Jenkins-Auftrags parameters: Parameter, die an den Auftrag übergeben werden sollen | <code>\${MY_STAGE.MY_TASK.input.job}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code> # Siehe alle Parameter <code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # Siehe Wert eines Parameters |
| | Output | job: Name des Jenkins-Auftrags jobId: ID des resultierenden Auftrags, z. B. 1234 jobStatus: Status in Jenkins jobResults: Erfassung von Test-/Codeabdeckungsergebnissen jobUrl: URL der resultierenden Auftragsausführung | <code>\${MY_STAGE.MY_TASK.output.job}</code> <code>\${MY_STAGE.MY_TASK.output.jobId}</code> <code>\${MY_STAGE.MY_TASK.output.jobStatus}</code> <code>\${MY_STAGE.MY_TASK.output.jobResults}</code> # Siehe alle Ergebnisse <code>\${MY_STAGE.MY_TASK.output.jobResults.junitResponse}</code> # Siehe JUnit-Ergebnisse <code>\${MY_STAGE.MY_TASK.output.jobResults.jacocoResponse}</code> # Siehe JaCoCo-Ergebnisse <code>\${MY_STAGE.MY_TASK.output.jobUrl}</code> |
| TFS | | | |

Tabelle 3-14. Integrieren von Entwicklungs-, Test- und Bereitstellungsanwendungen (Fortsetzung)

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|------------|--------|--|---|
| | Input | projectCollection: Projektsammlung aus TFS teamProject: Ausgewähltes Projekt aus der verfügbaren Erfassung buildDefinitionId: Auszuführende Build- Definitions-ID | \${MY_STAGE.MY_TASK.input.projectCollection} \${MY_STAGE.MY_TASK.input.teamProject} \${MY_STAGE.MY_TASK.input.buildDefinitionId} |
| | Output | buildId: Resultierende Build-ID buildUrl: URL zur Build- Übersicht logUrl: URL zu Protokollen dropLocation: Ablagespeicherort der Artefakte, falls vorhanden | \${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 des auszuführenden Workflows parameters: Parameter, die an den Workflow übergeben werden sollen | \${MY_STAGE.MY_TASK.input.workflowId} \${MY_STAGE.MY_TASK.input.parameters} |
| | Output | workflowExecutionId: ID der Workflow-Ausführung properties: Ausgabeeigenschaften aus der Workflow-Ausführung | \${MY_STAGE.MY_TASK.output.workflowExecutionId} \${MY_STAGE.MY_TASK.output.properties} |

Tabelle 3-15. Integrieren anderer Anwendungen über eine API

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|----------------|--------|--|--|
| REST | | | |
| | Input | url: aufzurufende URL action: Zu verwendende HTTP-Methode headers: Zu übergebende HTTP-Header payload: Anforderungsnutzlast fingerprint: Abzugleichender Fingerabdruck, wenn die URL vom Typ „https“ ist allowAllCerts: Wenn „true“ festgelegt ist, kann es sich um ein beliebiges Zertifikat mit einer HTTPS-URL handeln | <pre> \${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} </pre> |
| | Output | responseCode: HTTP-Antwortcodes responseHeaders: HTTP-Antwort-Header responseBody: Zeichenfolgenformat der empfangenen Antwort responseJson: Praktikable Antwort, wenn der „content-type“ application/json lautet | <pre> \${MY_STAGE.MY_TASK.output.responseCode} \${MY_STAGE.MY_TASK.output.responseHeaders} \$ {MY_STAGE.MY_TASK.output.responseHeaders.header1} # Siehe Antwortkopfzeile „header1“ \${MY_STAGE.MY_TASK.output.responseBody} \${MY_STAGE.MY_TASK.output.responseJson} # Siehe Antwort als JSON \${MY_STAGE.MY_TASK.output.responseJson.a.b.c} # Siehe verschachteltes Objekt im a.b.c JSON-Pfad in der Antwort </pre> |
| Abfrage | | | |

Tabelle 3-15. Integrieren anderer Anwendungen über eine API (Fortsetzung)

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|---------|--------|---|---|
| | Input | url: aufzurufende URL headers: Zu übergebende HTTP-Header exitCriteria: Kriterien, die erfüllt werden müssen, damit die Aufgabe erfolgreich ist oder fehlschlägt. Ein Schlüssel-Wert-Paar aus „success“ → Ausdruck, „failure“ → Ausdruck: pollCount: Anzahl der auszuführenden Iterationen pollIntervalSeconds: Anzahl der zu wartenden Sekunden zwischen jeder Iteration ignoreFailure: Wenn auf „true“ festgelegt, werden Fehler der Zwischenantwort ignoriert fingerprint: Abzugleichender Fingerabdruck, wenn die URL vom Typ „https“ ist allowAllCerts: Wenn „true“ festgelegt ist, kann es sich um ein beliebiges Zertifikat mit einer HTTPS-URL handeln | \${MY_STAGE.MY_TASK.input.url} \${MY_STAGE.MY_TASK.input.headers} \${MY_STAGE.MY_TASK.input.exitCriteria} \${MY_STAGE.MY_TASK.input.pollCount} \${MY_STAGE.MY_TASK.input.pollIntervalSeconds} \${MY_STAGE.MY_TASK.input.ignoreFailure} \${MY_STAGE.MY_TASK.input.fingerprint} \${MY_STAGE.MY_TASK.input.allowAllCerts} |
| | Output | responseCode: HTTP-Antwortcodes responseBody: Zeichenfolgenformat der empfangenen Antwort responseJson: Praktikable Antwort, wenn der „content-type“ application/json lautet | \${MY_STAGE.MY_TASK.output.responseCode} \${MY_STAGE.MY_TASK.output.responseBody} \${MY_STAGE.MY_TASK.output.responseJson} # Refer to response as JSON |

Tabelle 3-16. Ausführen von Remote-Skripts und benutzerdefinierten Skripten

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|--|---------|---|---|
| PowerShell Um eine PowerShell-Aufgabe ausführen zu können, benötigen Sie Folgendes: <ul style="list-style-type: none"> ■ Eine aktive Sitzung mit einem Remote-Windows-Host. ■ Wenn Sie beabsichtigen, einen base64-PowerShell-Befehl einzugeben, berechnen Sie zuerst die Gesamtlänge des Befehls. Weitere Informationen finden Sie unter In vRealize Automation Code Stream verfügbare Aufgabentypen. | | | |
| | Eingabe | host: IP-Adresse oder Hostname des Computers username: Benutzername, der zur Verbindung verwendet werden soll password: Kennwort für die Verbindung useTLS: Versuch, eine HTTPS- Verbindung herzustellen trustCert: Wenn auf „true“ festgelegt, wird selbstsignierten Zertifikaten vertraut script: Auszuführendes Skript workingDirectory: Verzeichnispfad, in den vor Skriptausführung gewechselt wird environmentVariables: Ein Schlüssel-Wert-Paar der festzulegenden Umgebungsvariable arguments: Argumente, die an das Skript übergeben werden | <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> |
| | Ausgabe | response: Inhalt der Datei \$SCRIPT_RESPONSE_FILE responseFilePath: Wert von \$SCRIPT_RESPONSE_FILE exitCode: Exit-Code verarbeiten logFilePath: Pfad zur Datei mit stdout errorFilePath: Pfad zur Datei mit stderr | <pre> \${MY_STAGE.MY_TASK.output.response} \$ {MY_STAGE.MY_TASK.output.responseFilePath} \${MY_STAGE.MY_TASK.output.exitCode} \${MY_STAGE.MY_TASK.output.logFilePath} \${MY_STAGE.MY_TASK.output.errorFilePath} </pre> |
| SSH | | | |

Tabelle 3-16. Ausführen von Remote-Skripts und benutzerdefinierten Skripts (Fortsetzung)

| Aufgabe | Scope | Key | Vorgehensweise zum Verwenden von SCOPE und KEY in der Aufgabe |
|---------|--------|--|--|
| | Input | host: IP-Adresse oder Hostname des Computers username: Benutzername, der zur Verbindung verwendet werden soll password: Kennwort zur Verbindung (kann optional privateKey verwenden) privateKey: Für die Verbindung zu verwendender privateKey passphrase: Optionale Passphrase zum Entsperren von privateKey script: Auszuführendes Skript workingDirectory: Verzeichnispfad, in den vor Skriptausführung gewechselt wird environmentVariables: Schlüssel-Wert-Paar der festzulegenden Umgebungsvariable | <pre> \${MY_STAGE.MY_TASK.input.host} \${MY_STAGE.MY_TASK.input.username} \${MY_STAGE.MY_TASK.input.password} \${MY_STAGE.MY_TASK.input.privateKey} \${MY_STAGE.MY_TASK.input.passphrase} \${MY_STAGE.MY_TASK.input.script} \$ {MY_STAGE.MY_TASK.input.workingDirectory} } \$ {MY_STAGE.MY_TASK.input.environmentVariables} </pre> |
| | Output | response: Inhalt der Datei \$SCRIPT_RESPONSE_FILE responseFilePath: Wert von \$SCRIPT_RESPONSE_FILE exitCode: Exit-Code verarbeiten logFilePath: Pfad zur Datei mit stdout errorFilePath: Pfad zur Datei mit 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> |

Vorgehensweise beim Verwenden einer Variablenbindung zwischen Aufgaben

Dieses Beispiel zeigt, wie Sie Variablenbindungen in Ihren Pipeline-Aufgaben verwenden.

Tabelle 3-17. Syntax-Formatbeispiele

| Beispiel | Syntax |
|--|--|
| Zum Verwenden eines Aufgabenausgabewerts für Pipeline-Benachrichtigungen und Pipeline-Ausgabeeigenschaften | <code>\${<Stage Key>.<Task Key>.output.<Task output key>}</code> |
| Zum Verweisen auf den Wert der vorherigen Aufgabenausgabe als Eingabe für die aktuelle Aufgabe. | <code>\${<Previous/Current Stage key>.<Previous task key not in current Task group>.output.<task output key>}</code> |

Weitere Informationen

Weitere Informationen zum Binden von Variablen in Aufgaben finden Sie unter:

- [Vorgehensweise zum Verwenden von Variablenbindungen in vRealize Automation Code Stream-Pipelines](#)
- [Vorgehensweise zum Verwenden von Variablenbindungen in einer Bedingungs Aufgabe zum Ausführen oder Anhalten einer Pipeline in vRealize Automation Code Stream](#)
- [In vRealize Automation Code Stream verfügbare Aufgabentypen](#)

Vorgehensweise zum Senden von Benachrichtigungen über meine Pipeline in vRealize Automation Code Stream

Benachrichtigungen sind Möglichkeiten, mit Ihren Teams zu kommunizieren und sie über den Status Ihrer Pipelines in vRealize Automation Code Stream zu informieren.

Um Benachrichtigungen zu senden, wenn eine Pipeline ausgeführt wird, können Sie vRealize Automation Code Stream-Benachrichtigungen basierend auf dem Status der gesamten Pipeline, Phase oder Aufgabe konfigurieren.

- Eine E-Mail-Benachrichtigung sendet eine E-Mail bei:
 - Abschluss, Fehlschlagen, Abbruch oder Start der Pipeline, oder wenn sich die Pipeline im Wartezustand befindet.
 - Abschluss, Fehlschlagen oder Start der Phase.
 - Abschluss, Fehlschlagen oder Start der Aufgabe, oder wenn sich die Aufgabe im Wartezustand befindet.
- Eine Ticket-Benachrichtigung erstellt ein Ticket, die einem Teammitglied zugewiesen wird, bei:
 - Fehlschlagen oder Abschluss der Pipeline.
 - Fehlschlagen der Phase.
 - Fehlschlagen der Aufgabe.

- Eine Webhook-Benachrichtigung sendet eine Anforderung an eine andere Anwendung bei:
 - Fehlschlagen, Abschluss, Abbruch oder Start der Pipeline, oder wenn sich die Pipeline im Wartezustand befindet.
 - Fehlschlagen, Abschluss oder Start der Phase.
 - Fehlschlagen, Abschluss oder Start der Aufgabe, oder wenn sich die Aufgabe im Wartezustand befindet.

Sie können beispielsweise eine E-Mail-Benachrichtigung für eine Benutzervorgangsaufgabe konfigurieren, um an einem bestimmten Punkt in der Pipeline eine Genehmigung zu erhalten. Bei Ausführung der Pipeline sendet diese Aufgabe eine E-Mail an die Person, die die Aufgabe genehmigen muss. Wenn das Ablaufzeitlimit in der Benutzervorgangsaufgabe auf Tage, Stunden oder Minuten festgelegt ist, muss der erforderliche Benutzer die Pipeline vor Ablauf der Aufgabe genehmigen. Ansonsten schlägt die Pipeline erwartungsgemäß fehl.

Um ein Jira-Ticket zu erstellen, wenn eine Pipeline-Aufgabe fehlschlägt, können Sie eine Benachrichtigung konfigurieren. Alternativ können Sie eine Webhook-Benachrichtigung konfigurieren, um eine Anfrage zum Status einer Pipeline basierend auf dem Pipeline-Ereignis an einen Slack-Kanal zu senden.

Sie können Variablen in allen Benachrichtigungstypen verwenden. Sie können beispielsweise `{var}` in der URL einer Webhook-Benachrichtigung verwenden.

Voraussetzungen

- Überprüfen Sie, ob eine oder mehrere Pipelines erstellt wurden. Weitere Informationen finden Sie in den Anwendungsfällen in [Kapitel 5 Lernprogramme für die Verwendung von vRealize Automation Code Stream](#).
- Bestätigen Sie zum Senden von E-Mail-Benachrichtigungen, dass Sie auf einen funktionierenden E-Mail-Server zugreifen können. Weitere Informationen erhalten Sie bei Ihrem Administrator.
- Bestätigen Sie zum Erstellen von Tickets (z. B. ein Jira-Ticket), dass der Endpoint vorhanden ist. Weitere Informationen hierzu finden Sie unter [Definition von Endpoints in vRealize Automation Code Stream](#).
- Um eine Benachrichtigung basierend auf einer Integration zu senden, erstellen Sie eine Webhook-Benachrichtigung. Bestätigen Sie anschließend, dass der Webhook hinzugefügt wurde und funktioniert. Sie können Benachrichtigungen mit Anwendungen wie Slack, GitHub oder GitLab verwenden.

Verfahren

- 1 Öffnen Sie eine Pipeline.

- 2 So erstellen Sie eine Benachrichtigung zum Gesamt-Pipeline-Status oder den Status einer Phase oder Aufgabe:

| Um eine Benachrichtigung für Folgendes zu erstellen: | Gehen Sie wie folgt vor: |
|--|---|
| Pipeline-Status | Klicken Sie auf eine leere Stelle auf der Pipeline-Arbeitsfläche. |
| Status einer Phase | Klicken Sie auf einen leeren Bereich in einer Phase der Pipeline. |
| Status einer Aufgabe | Klicken Sie auf eine Aufgabe in einer Phase der Pipeline. |

- 3 Klicken Sie auf die Registerkarte **Benachrichtigungen**.
- 4 Klicken Sie auf **Hinzufügen**, wählen Sie den Typ der Benachrichtigung aus und konfigurieren Sie die Benachrichtigungsdetails.
- 5 Um eine Slack-Benachrichtigung zu erstellen, wenn eine Pipeline erfolgreich ist, erstellen Sie eine Webhook-Benachrichtigung.
- Wählen Sie **Webhook** aus.
 - Um die Slack-Benachrichtigung zu konfigurieren, geben Sie die Informationen ein.
 - Klicken Sie auf **Speichern**.
 - Wenn die Pipeline ausgeführt wird, erhält der Slack-Kanal die Benachrichtigung über den Pipeline-Status. Benutzern kann beispielsweise Folgendes im Slack-Kanal angezeigt werden:

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

- 6 Zum Erstellen eines Jira-Tickets konfigurieren Sie die Ticketinformationen.
- Wählen Sie **Ticket** aus.
 - Um die Jira-Benachrichtigung zu konfigurieren, geben Sie die Informationen ein.
 - Klicken Sie auf **Speichern**.

Notification

Send notification type ☐ Email ☒ Ticket ☐ Webhook

When pipeline ☒ Fails ☐ Completes

Jira endpoint

Create Ticket

Jira project

Issue type

Assignee

Summary

Description

Ergebnisse

Herzlichen Glückwunsch! Sie haben gelernt, dass Sie verschiedene Benachrichtigungstypen in mehreren Bereichen Ihrer Pipeline in vRealize Automation Code Stream erstellen können.

Nächste Schritte

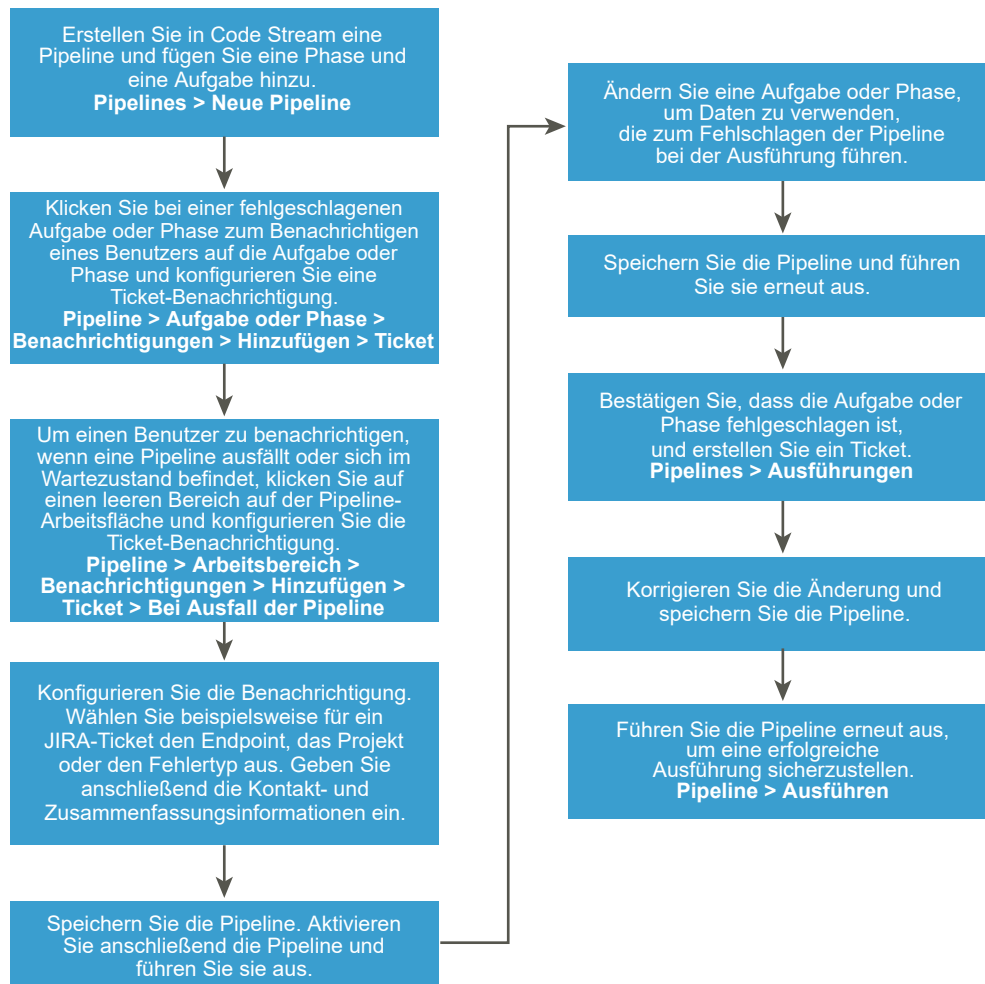
Ein detailliertes Beispiel zum Erstellen einer Benachrichtigung finden Sie unter [Vorgehensweise zum Erstellen eines JIRA-Tickets in vRealize Automation Code Stream bei einer fehlgeschlagenen Pipeline-Aufgabe](#).

Vorgehensweise zum Erstellen eines JIRA-Tickets in vRealize Automation Code Stream bei einer fehlgeschlagenen Pipeline-Aufgabe

Wenn eine Phase oder Aufgabe in Ihrer Pipeline fehlschlägt, können Sie über vRealize Automation Code Stream ein Jira-Ticket erstellen. Sie können das Ticket der Person zuweisen, die das Problem beheben muss. Sie können auch ein Ticket erstellen, wenn sich die Pipeline im Wartemodus befindet oder erfolgreich ausgeführt wird.

Sie können Benachrichtigungen für eine Aufgabe, eine Phase oder eine Pipeline hinzufügen und konfigurieren. vRealize Automation Code Stream erstellt das Ticket basierend auf dem Status der Aufgabe, der Phase oder Pipeline, der Sie die Benachrichtigung hinzufügen. Wenn ein Endpoint beispielsweise nicht verfügbar ist, können Sie mit vRealize Automation Code Stream ein JIRA-Ticket für die fehlgeschlagene Aufgabe erstellen, weil sie keine Verbindung zum Endpoint herstellen kann.

Sie können auch Benachrichtigungen erstellen, wenn die Pipeline erfolgreich ausgeführt wird. Sie können Ihr Qualitätssicherungsteam beispielsweise über erfolgreich ausgeführte Pipelines informieren, damit das Team den Build überprüfen und eine andere Test-Pipeline ausführen kann. Oder Sie benachrichtigen das Leistungsteam, damit es die Leistung der Pipeline messen und Vorbereitungen für ein Update der Bereitstellung oder Produktion treffen kann.



In diesem Beispiel wird ein JIRA-Ticket erstellt, wenn eine Pipeline-Aufgabe fehlschlägt.

Voraussetzungen

- Stellen Sie sicher, dass Sie über ein gültiges JIRA-Konto verfügen und sich bei Ihrer JIRA-Instanz anmelden können.
- Stellen Sie sicher, dass ein JIRA-Endpoint vorhanden ist und funktioniert.

Verfahren

- 1 Klicken Sie in der Pipeline auf eine Aufgabe.
- 2 Klicken Sie im Bereich „Aufgabenkonfiguration“ auf **Benachrichtigungen**.
- 3 Klicken Sie auf **Hinzufügen** und konfigurieren Sie die Ticketinformationen.
 - a Klicken Sie auf **Ticket**.
 - b Wählen Sie den JIRA-Endpoint aus.
 - c Geben Sie das Jira-Projekt und den Issue-Typ ein.
 - d Geben Sie die E-Mail-Adresse der Person ein, die das Ticket erhalten soll.
 - e Geben Sie eine Zusammenfassung und eine Beschreibung des Tickets ein und klicken Sie dann auf **Speichern**.

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 Speichern Sie die Pipeline, aktivieren Sie sie und führen Sie sie aus.
- 5 Testen Sie das Ticket.
 - a Ändern Sie die Aufgabeninformationen so, dass sie Daten enthalten, die das Fehlschlagen der Aufgabe verursachen.
 - b Speichern Sie die Pipeline und führen Sie sie erneut aus.
 - c Klicken Sie auf **Ausführungen** und bestätigen Sie, dass die Pipeline fehlgeschlagen ist.

- d Bestätigen Sie in der Ausführung, dass vRealize Automation Code Stream das Ticket erstellt und gesendet hat.
- e Korrigieren Sie die Aufgabeninformationen. Führen Sie die Pipeline dann erneut aus und stellen Sie sicher, dass sie erfolgreich ausgeführt wird.

Ergebnisse

Herzlichen Glückwunsch! Sie hatten vRealize Automation Code Stream veranlasst, bei einem Pipeline-Fehler ein JIRA-Ticket zu erstellen und es der Person zuzuweisen, die den Fehler beheben sollte.

Nächste Schritte

Fügen Sie weitere Benachrichtigungen hinzu, um Ihr Team über die Pipelines zu informieren.

Vorgehensweise zum Rollback meiner Bereitstellung in vRealize Automation Code Stream

Sie konfigurieren das Rollback als Pipeline mit Aufgaben, die Ihre Bereitstellung nach einem Fehler in einer Bereitstellungs-Pipeline auf einen früheren stabilen Zustand zurücksetzen. Um im Falle eines Fehlers ein Rollback durchzuführen, ordnen Sie die Rollback-Pipeline Aufgaben oder Phasen zu.

Je nach Ihrer Rolle können Ihre Gründe für ein Rollback variieren.

- Als Versionsentwickler möchten Sie, dass vRealize Automation Code Stream den Erfolg während einer Freigabe überprüft, damit Sie wissen, ob Sie mit der Freigabe fortfahren oder ein Rollback durchführen. Mögliche Fehler sind Aufgabenfehler, eine Ablehnung in UserOps oder eine Überschreitung des Metrikschwellenwerts.
- Als Besitzer einer Umgebung möchten Sie eine vorherige Version erneut bereitstellen, damit Sie eine Umgebung schnell wieder in einen zweifelsfrei funktionierenden Zustand versetzen können.
- Als Besitzer einer Umgebung möchten Sie das Rollback einer Blau/Grün-Bereitstellung unterstützen, um die Ausfallzeit bei fehlgeschlagenen Versionen minimieren zu können.

Wenn Sie eine intelligente Pipeline-Vorlage mit aktivierter Rollback-Option zum Erstellen einer CD-Pipeline verwenden, wird das Rollback automatisch den Aufgaben in der Pipeline hinzugefügt. In diesem Anwendungsfall verwenden Sie die intelligente Pipeline-Vorlage, um das Rollback für eine Anwendungsbereitstellung auf einem Kubernetes-Cluster mithilfe des Bereitstellungsmodells für das parallele Upgrade zu definieren. Die intelligente Pipeline-Vorlage erstellt eine Bereitstellungs-Pipeline und eine oder mehrere Rollback-Pipelines.

- In der Bereitstellungs-Pipeline ist ein Rollback erforderlich, wenn Aufgaben vom Typ „Bereitstellung aktualisieren“ oder „Bereitstellung überprüfen“ fehlschlagen.
- In der Rollback-Pipeline wird die Bereitstellung mit einem alten Image aktualisiert.

Sie können eine Rollback-Pipeline auch manuell mithilfe einer leeren Vorlage erstellen. Bevor Sie eine Rollback-Pipeline erstellen, sollten Sie den Rollback-Ablauf planen. Weitere Hintergrundinformationen zum Rollback finden Sie unter [Planen für ein Rollback in vRealize Automation Code Stream](#).

Voraussetzungen

- Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls nicht, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).
- Richten Sie die Kubernetes-Cluster ein, in denen Ihre Pipeline Ihre Anwendung bereitstellt. Richten Sie einen Entwicklungs-Cluster und einen Produktions-Cluster ein.
- Erstellen Sie die Kubernetes-Entwicklungs- und -Produktions-Endpoints, die Ihr Anwendungs-Image in den Kubernetes-Clustern bereitstellen.
- Stellen Sie sicher, dass Sie über die Einrichtung einer Docker-Registrierung verfügen.
- Stellen Sie sicher, dass Sie über eine Kubernetes-YAML-Datei verfügen, die auf die Bereitstellung angewendet werden soll.
- Machen Sie sich mit der intelligenten Pipeline-Vorlage für CD vertraut. Weitere Informationen hierzu finden Sie unter [Planen eines nativen CD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage](#).

Verfahren

- 1 Klicken Sie auf **Pipelines > Neue Pipeline > Intelligente Vorlage > Kontinuierliche Bereitstellung**.
- 2 Geben Sie die Informationen in die intelligente Pipeline-Vorlage ein.
 - a Wählen Sie ein Projekt aus.
 - b Geben Sie einen Pipeline-Namen ein, zum Beispiel **RollingUpgrade-Example**.
 - c Wählen Sie die Umgebungen für Ihre Anwendung aus. Wenn Sie ein Rollback zu Ihrer Bereitstellung hinzufügen möchten, müssen Sie **Prod** auswählen.
 - d Klicken Sie auf **Auswählen**, wählen Sie eine Kubernetes-YAML-Datei aus und klicken Sie auf **Prozess**.

Die intelligente Pipeline-Vorlage zeigt die verfügbaren Dienste und Bereitstellungsumgebungen an.

- e Wählen Sie den Dienst aus, den die Pipeline für die Bereitstellung verwenden soll.
- f Wählen Sie die Cluster-Endpoints für die Dev- und die Prod-Umgebung aus.
- g Wählen Sie als Image-Quelle **Pipeline-Laufzeiteingabe** aus.
- h Wählen Sie als Bereitstellungsmodell **Paralleles Upgrade** aus.

- i Klicken Sie auf **Rollback**.
- j Geben Sie die **URL für die Integritätsprüfung** an.

Intelligente Vorlage: Kontinuierliche Bereitst...

Endpoint-Voraussetzun... Kubernetes Docker-Registrierung

Projekt *

Name der Pipeline *

Umgebung Entwicklung Produktion

Kubernetes-YAML-Datei... AUSWÄHLEN VERARBEITEN
Verarbeitete Dateien: Kubernetesbgreen1.yaml

Dienst auswählen

| Name der Bereitstellung | Dienst | Namespace | Image |
|--|-----------------|-----------|--|
| <input checked="" type="radio"/> codestream-demo | codestream-demo | bgreen1 | symphony-tango-beta2.frog.io/codestream-demo |

1 Dienste

Bereitstellung

| Umgebung | Cluster-Endpoint | Namespace |
|-------------|-----------------------------|----------------|
| Entwicklung | Kubernetes-Endpoint-Staging | bgreen1-149157 |
| Produktion | Kubernetes-Endpoint-Staging | bgreen1 |

Image-Quelle * ☐ Docker-Auslöser ☒ Pipeline-Laufzeiteingabe

Bereitstellungsmodell * ☐ Canary ☒ Paralleles Upgrade ☐ Blau/Grün

Rollback ☒

Integritätsprüfungs-URL *

ERSTELLEN ABBRECHEN

- 3 Um die Pipeline mit dem Namen „RollbackUpgrade-Example“ zu erstellen, klicken Sie auf **Erstellen**.

Die Pipeline mit dem Namen „RollbackUpgrade-Example“ wird angezeigt und das Rollback-Symbol wird bei Aufgaben angezeigt, für die in der Entwicklungs- und Produktionsphase ein Rollback durchgeführt werden kann.

RollbackUpgrade-Example Deaktiviert

Arbeitsbereich | Eingabe | **Modell** | Ausgabe

Development

- Create Namesp... (Kubernetes)
- Create Secret (Kubernetes) **Ausgewählt**
- Create Sever (Kubernetes)

Production

- Create Se_trans... (Kubernetes)
- Update Deploy... (Kubernetes)
- Verify Deploy... (Kubernetes)

Aufgabe: Create Secret Benachrichtigungen Rollback **AUFGABE VALIDIEREN**

Aufgabenname: Create Secret

Typ: Kubernetes

Bei Fehler fortfahren: ☐

Aufgabe ausführen: ☒ Immer ☐ Bei Bedingung

Kubernetes-Aufgabeneigenschaften

Kubernetes-Cluster: Dev-VKE-Cluster

Zeitüberschreitung: 5

Aktion: ☐ Abrufen ☒ Erstellen ☐ Übernehmen ☐ Löschen ☐ Rollback

Bei Konflikt fortfahr...: ☒

Quellentyp: ☐ Quellcodeverwaltung ☒ Lokale Definition

Lokale YAML-Defini...: [AUS DATEI LESEN](#)

```
1 apiVersion: v1
2 data:
3   .dockercfg: eyJ2chVwa69ue510WV5nby1iZXh0Ml5sd1afjka12sdfxrg2hsch2hah
4   2fdsH5zxdg2dFh5s13h8dfs5453hfdsfh3as1Sghn1fs315h3f1ds5h5s3of15
5   h315sdf15h53108fds45h04f5d54h56h4in19
6 Kind: Secret
7 metadata:
8   name: jfrog-beta2
9   namespace: lgreen-549930
10 type: kubernetes.io/dockercfg
```

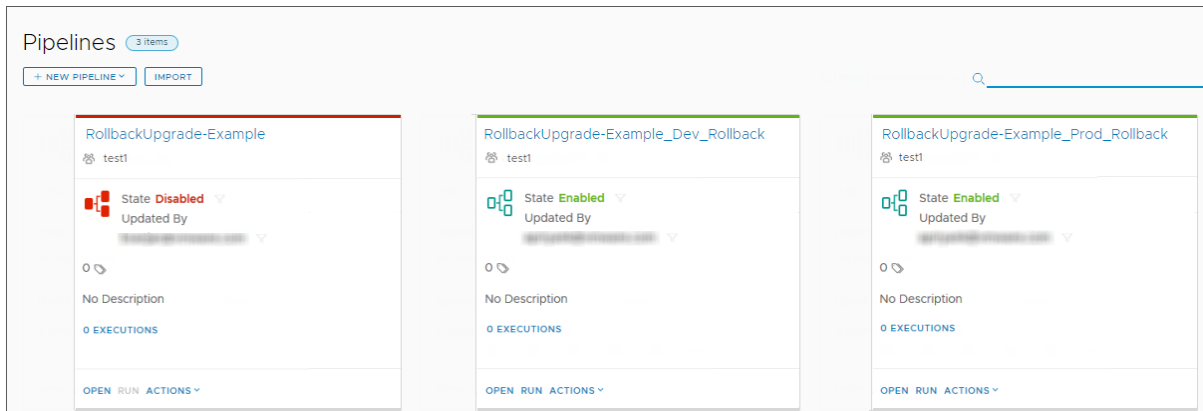
Ausgabeparameter: [status](#) [k8SRollbackTaskFields](#) [endpoint](#) [response](#) [yamlis](#) [operation](#) [config](#)

SPEICHERN **AUSFÜHREN** **SCHLIESSEN** Zuletzt gespeichert: vor 11 Minuten

4 Schließen Sie die Pipeline.

Auf der Seite „Pipelines“ werden die von Ihnen erstellte Pipeline und eine neue Pipeline für jede Phase in Ihrer Pipeline angezeigt.

- RollingUpgrade-Example. vRealize Automation Code Stream deaktiviert die Pipeline, die Sie standardmäßig erstellt haben, was sicherstellt, dass Sie sie vor der Ausführung überprüfen.
- RollingUpgrade-Example_Dev_Rollback. Beim Ausfall von Aufgaben in der Entwicklungsphase, z. B. **Dienst erstellen**, **Geheimen Schlüssel erstellen**, **Bereitstellung erstellen** und **Bereitstellung überprüfen**, wird diese Entwicklungs-Rollback-Pipeline aufgerufen. Um das Rollback von Entwicklungsaufgaben sicherzustellen, aktiviert vRealize Automation Code Stream standardmäßig die Entwicklungs-Rollback-Pipeline.
- RollingUpgrade-Example_Prod_Rollback. Diese Produktions-Rollback-Pipeline wird beim Ausfall von Aufgaben in der Produktionsphase aufgerufen, z. B. bei **Phase 1 bereitstellen**, **Phase 1 überprüfen**, **Rollout-Phase bereitstellen**, **Rollout-Phase abschließen** und **Rollout-Phase überprüfen**. Um das Rollback von Produktionsaufgaben sicherzustellen, aktiviert vRealize Automation Code Stream standardmäßig die Produktions-Rollback-Pipeline.



- 5 Aktivieren Sie die von Ihnen erstellte Pipeline und führen Sie sie aus.

Wenn Sie die Ausführung starten, fordert vRealize Automation Code Stream Sie zur Angabe von Eingabeparametern auf. Sie stellen das Image und das Tag für den Endpoint im Docker-Repository bereit, das Sie verwenden.

- 6 Klicken Sie auf der Seite „Ausführungen“ auf **Aktionen > Ausführung anzeigen** und überwachen Sie die Pipeline-Ausführung.

Die Pipeline startet mit dem Status **RUNNING** und durchläuft die Aufgaben der Entwicklungsphase. Wenn die Pipeline während der Entwicklungsphase keine Aufgabe ausführen kann, wird die Pipeline mit dem Namen „RollingUpgrade-Example_Dev_Rollback“ ausgelöst, die die Bereitstellung zurücksetzt, und der Pipelinestatus ändert sich in **ROLLING_BACK**.

[< 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

Nach dem Rollback werden auf der Seite „Ausführungen“ zwei RollingUpgrade-Example-Pipeline-Ausführungen aufgelistet.

- Die von Ihnen erstellte Pipeline, für die ein Rollback durchgeführt wurde, zeigt **ROLLBACK_COMPLETED** an.
- Die Entwicklungs-Rollback-Pipeline, die ausgelöst wurde und das Rollback durchgeführt hat, zeigt **COMPLETED** an.

Executions 604 items

[+ NEW EXECUTION](#) Q

RollbackUpgrade-Example_Dev... #1 COMPLETED Stages: ● ● ●
 1 Rollback for RollbackUpgrade-Example#1
 By Cloud on 01/11/2019 1:36 PM
 Execution Completed.
 Comments: Triggered to rollback Development. Create Deployment of RollbackUpgrade-Example#1

RollbackUpgrade-Example#1 ROLLBACK_COMPLETED Stages: ● ● ● ● ●
 0
 By Cloud on 01/11/2019 1:24 PM
 Create Deployment ROLLBACK_COMPLETED

Ergebnisse

Herzlichen Glückwunsch! Sie haben erfolgreich eine Pipeline mit Rollback definiert und die Durchführung des Pipeline-Rollbacks durch vRealize Automation Code Stream zum Zeitpunkt des Fehlers überwacht.

Planen eines nativen Builds, der Integration und Bereitstellung von Code in vRealize Automation Code Stream

4

Bevor Sie mit vRealize Automation Code Stream mithilfe der nativen Funktion, die eine CICD-, CI- oder CD-Pipeline für Sie erstellt, Ihren Code erstellen, integrieren und liefern, planen Sie Ihren nativen Build. Anschließend können Sie Ihre Pipeline mit einer der intelligenten Pipeline-Vorlagen oder durch das manuelle Hinzufügen von Phasen und Aufgaben erstellen.

Um Ihren Build für die kontinuierliche Integration und die kontinuierliche Bereitstellung zu planen, haben wir mehrere Beispiele zusammengestellt, mit denen gezeigt wird, wie Sie dabei vorgehen. In diesen Plänen werden die Voraussetzungen und Übersichten beschrieben, mit deren Hilfe Sie die native Build-Funktion effektiv vorbereiten und verwenden können, wenn Sie Ihre Pipelines erstellen.

Dieses Kapitel enthält die folgenden Themen:

- Planen eines nativen CICD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage
- Planen eines nativen CI-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage
- Planen eines nativen CD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage
- Planen eines nativen CICD-Builds in vRealize Automation Code Stream vor dem manuellen Hinzufügen von Aufgaben
- Planen für ein Rollback in vRealize Automation Code Stream

Planen eines nativen CICD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage

Zum Erstellen einer CICD-Pipeline (Continuous Integration und Continuous Delivery) in vRealize Automation Code Stream können Sie die intelligente Pipeline-Vorlage für CICD verwenden. Zum Planen Ihres nativen CICD-Builds sammeln Sie die Informationen, die Sie zum Ausfüllen der intelligenten Pipeline-Vorlage benötigen, bevor Sie sie zum Erstellen der Pipeline in diesem Beispielplan verwenden.

Nachdem Sie die Informationen in der intelligenten Pipeline-Vorlage eingegeben und gespeichert haben, erstellt die Vorlage eine Pipeline, die die Phasen und Aufgaben enthält. Außerdem wird angegeben, wo das Image basierend auf den ausgewählten Umgebungstypen (z. B. „Entwicklung“ oder „Produktion“) bereitgestellt werden soll. Die Pipeline veröffentlicht Ihr Docker-Image und führt die Aktionen durch, die zur Ausführung des Images erforderlich sind. Nachdem Ihre Pipeline ausgeführt wurde, können Sie Trends über die Pipeline-Ausführungen hinweg überwachen.

Zum Erstellen einer CICD-Pipeline müssen Sie sowohl die CI- (Continuous Integration, Kontinuierliche Integration) als auch die CD-Phase (Continuous Delivery, Kontinuierliche Bereitstellung) Ihrer Pipeline planen.

Wenn eine Pipeline ein Image aus dem Docker-Hub enthält, müssen Sie vor der Pipeline-Ausführung sicherstellen, dass cURL in das Image eingebettet wurde. Wenn die Pipeline ausgeführt wird, lädt vRealize Automation Code Stream eine Binärdatei herunter, die cURL zum Ausführen von Befehlen verwendet.

Planen der CI-Phase

Zum Planen der CI-Phase Ihrer Pipeline richten Sie die externen und internen Anforderungen ein und bestimmen die Informationen, die in den CI-Abschnitt der intelligenten Pipeline-Vorlage eingegeben werden. Zusammenfassung.

Endpoints und Repositorys, die Sie benötigen:

- Ein Git-Quellcode-Repository, in das die Entwickler Code einchecken. vRealize Automation Code Stream übergibt den aktuellen Code an die Pipeline, wenn Entwickler Änderungen vornehmen.
- Ein Git-Endpoint für das Repository, in dem sich der Quellcode der Entwickler befindet.
- Ein Docker-Endpoint für den Docker-Build-Host, der die Build-Befehle in einem Container ausführt.
- Ein Kubernetes-Endpoint, damit vRealize Automation Code Stream Ihr Image auf einem Kubernetes-Cluster bereitstellen kann.
- Ein Builder-Image, das den Container erstellt, auf dem die Tests für die kontinuierliche Integration ausgeführt werden.
- Ein Image-Registrierungs-Endpoint, damit der Docker-Build-Host das Builder-Image daraus abrufen kann.

Sie benötigen Zugriff auf ein Projekt. Ein Projekt, in dem Ihre gesamte Arbeit, einschließlich Ihrer Pipeline, Endpoints und Dashboards, gruppiert wird. Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls nicht, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).

Sie brauchen einen Git-Webhook, der es vRealize Automation Code Stream ermöglicht, den Git-Auslöser zu verwenden, der Ihre Pipeline auslöst, wenn Entwickler Codeänderungen vornehmen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwenden des Git-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#).

Ihre Build-Toolsets:

- Ihr Build-Typ, z. B. Maven.
- Alle von Ihnen verwendeten Tools für die Nachbearbeitung von Builds, einschließlich JUnit, JaCoCo, Checkstyle und FindBugs.

Ihr Veröffentlichungstool:

- Ein Tool, wie z. B. Docker, das den Build-Container bereitstellt.
- Ein Image-Tag, bei dem es sich entweder um die Commit-ID oder die Build-Nummer handelt.

Ihr Build-Arbeitsbereich:

- Ein Docker-Build-Host, der den Docker-Endpoint darstellt.
- Eine Image-Registrierung. Der CI-Abschnitt der Pipeline ruft das Image aus dem ausgewählten Registrierungs-Endpoint ab. Der Container führt die CI-Aufgaben aus und stellt das Image bereit. Wenn bei der Registrierung Anmeldedaten benötigt werden, müssen Sie zuerst einen Image-Registrierungs-Endpoint erstellen und diesen dann hier auswählen, damit der Host das Image aus der Registrierung abrufen kann.
- URL für das Builder-Image, das den Container erstellt, auf dem die Aufgaben für die kontinuierliche Integration ausgeführt werden.

Planen der CD-Phase

Zum Planen der CD-Phase Ihrer Pipeline richten Sie die externen und internen Anforderungen ein und bestimmen die Informationen, die in den CD-Abschnitt der intelligenten Pipeline-Vorlage eingegeben werden.

Endpoints, die Sie benötigen:

- Ein Kubernetes-Endpoint, damit vRealize Automation Code Stream Ihr Image auf einem Kubernetes-Cluster bereitstellen kann.

Umgebungstypen und Dateien:

- Alle Umgebungstypen, in denen vRealize Automation Code Stream Ihre Anwendung bereitstellt, z. B. „Entwicklung“ und „Produktion“. Die intelligente Pipeline-Vorlage erstellt die Phasen und Aufgaben in Ihrer Pipeline basierend auf den ausgewählten Umgebungstypen.

Tabelle 4-1. Von der intelligenten Pipeline-Vorlage für CICD erstellte Pipeline-Phasen

| Pipeline-Inhalt | Funktionsweise |
|-------------------------------|--|
| Build-/Veröffentlichungsphase | Erstellt und testet Ihren Code, erstellt das Builder-Image und veröffentlicht das Image auf Ihrem Docker-Host. |
| Entwicklungsphase | Verwendet einen Amazon Web Services-Entwicklungscluster (AWS), um das Image zu erstellen und bereitzustellen. In dieser Phase können Sie einen Namespace auf dem Cluster sowie einen geheimen Schlüssel erstellen. |
| Produktionsphase | Verwendet eine Produktionsversion der VMware Tanzu Kubernetes Grid Integrated Edition (zuvor als VMware Enterprise PKS bezeichnet), um das Image in einem Kubernetes-Produktionscluster bereitzustellen. |

- Eine Kubernetes-YAML-Datei, die Sie im CD-Abschnitt der intelligenten Pipeline-Vorlage für CICD auswählen.

Zum Anwenden der Datei klicken Sie auf **Auswählen**, wählen die Kubernetes-YAML-Datei aus und klicken auf **Verarbeiten**. Die intelligente Pipeline-Vorlage zeigt die verfügbaren Dienste und Bereitstellungsumgebungen an. Sie wählen einen Dienst, den Cluster-Endpoint und die Bereitstellungsstrategie aus. Um beispielsweise das Canary-Bereitstellungsmodell zu verwenden, wählen Sie **Canary** aus und geben den Prozentsatz für die Bereitstellungsphase ein:

Intelligente Vorlage: KI/KB

Schritt 2 von 2

Umgebung * ☒ Entwicklung ☒ Produktion

Kubernetes-YAML-Datei... *

Verarbeitete Dateien: codestream.yaml

Dienst auswählen

| Name der Bereitstellung | Dienst | Namespace | Image |
|-------------------------|-----------------|-----------|-------|
| codestream-demo | codestream-demo | bgreen1 | 7 |

1 Dienste

Bereitstellung

| Umgebung | Cluster-Endpoint | Namespace |
|-------------|--|---------------|
| Entwicklung | 1030Endpoint-Kubernetes 騎家表ホあA中在e諸停日選Ü8äü*ñ | bgreen1-14724 |
| Produktion | 1030Endpoint-Kubernetes 騎家表ホあA中在e諸停日選Ü8äü*ñ | bgreen1 |

Bereitstellungsmodell * ☒ Canary ☐ Paralleles Upgrade ☐ Blau/Grün

Phase 1 * %

Rollback ☐

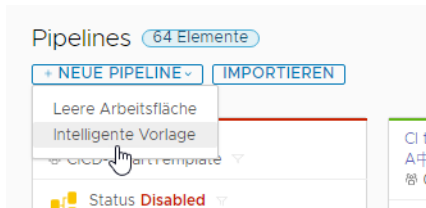
Integritätsprüfungs-URL *

Ein Beispiel für die Verwendung der intelligenten Pipeline-Vorlage zum Erstellen einer Pipeline für eine Blau/Grün-Bereitstellung finden Sie unter [Vorgehensweise zum Bereitstellen meiner Anwendung in vRealize Automation Code Stream für meine Blau/Grün-Bereitstellung](#).

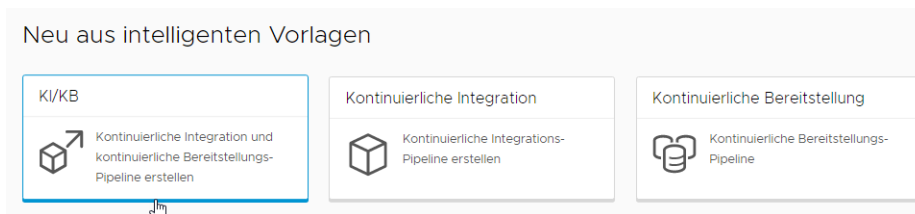
Vorgehensweise zum Erstellen der CICD-Pipeline mithilfe der intelligenten Pipeline-Vorlage

Nachdem Sie alle Informationen gesammelt und alles Notwendige eingerichtet haben, erhalten Sie hier Informationen zum Erstellen einer Pipeline anhand der intelligenten Pipeline-Vorlage für CICD.

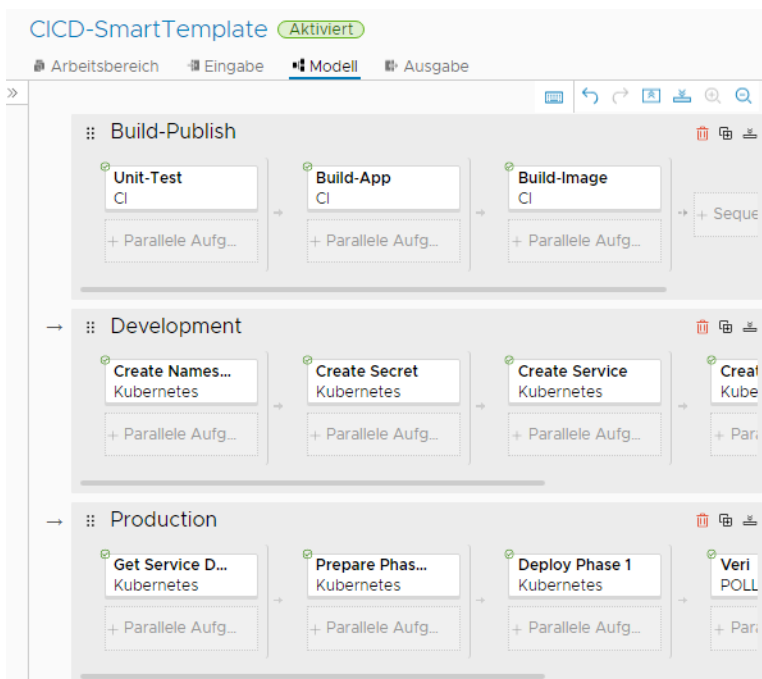
Klicken Sie unter „Pipelines“ auf **Neue Pipeline > Intelligente Vorlagen**.



Sie wählen die intelligente Pipeline-Vorlage für CICD aus.



Sie füllen die Vorlage aus und speichern die Pipeline mit den von ihr erstellten Phasen. Wenn Sie abschließende Änderungen vornehmen müssen, können Sie die Pipeline bearbeiten und speichern.



Anschließend aktivieren Sie die Pipeline und führen sie aus. Nach der Ausführung der Pipeline finden Sie hier einige Dinge, nach denen Sie suchen können:

- Stellen Sie sicher, dass Ihre Pipeline erfolgreich war. Klicken Sie auf **Ausführungen** und suchen Sie nach Ihrer Pipeline. Wenn sie fehlgeschlagen ist, korrigieren Sie alle Fehler und führen Sie sie erneut aus.
- Stellen Sie sicher, dass der Git-Webhook ordnungsgemäß funktioniert. Auf der Git-Registerkarte **Aktivität** werden die Ereignisse angezeigt. Klicken Sie auf **Auslöser > Git > Aktivität**.
- Sehen Sie sich das Pipeline-Dashboard an und untersuchen Sie die Trends. Klicken Sie auf **Dashboards** und suchen Sie nach Ihrem Pipeline-Dashboard. Sie können auch ein benutzerdefiniertes Dashboard erstellen, um zusätzliche KPIs zu melden.

Ein detailliertes Beispiel finden Sie unter [Vorgehensweise zur kontinuierlichen Integration von Code aus einem GitHub- oder GitLab-Repository in eine Pipeline in vRealize Automation Code Stream](#).

Planen eines nativen CI-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage

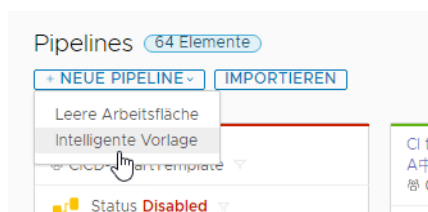
Um eine CI-Pipeline (Continuous Integration) in VMware Code Stream zu erstellen, können Sie die intelligente Pipeline-Vorlage für CI verwenden. Um Ihren nativen CI-Build zu planen, erfassen Sie die Informationen, die Sie zum Ausfüllen der intelligenten Pipeline-Vorlage benötigen, bevor Sie sie zum Erstellen der Pipeline in diesem Beispielplan verwenden.

Wenn Sie die intelligente Pipeline-Vorlage ausfüllen, erstellt sie eine CI-Pipeline in Ihrem Repository und führt die für die Ausführung erforderlichen Aktionen aus. Nachdem Ihre Pipeline ausgeführt wurde, können Sie Trends über die Pipeline-Ausführungen hinweg überwachen.

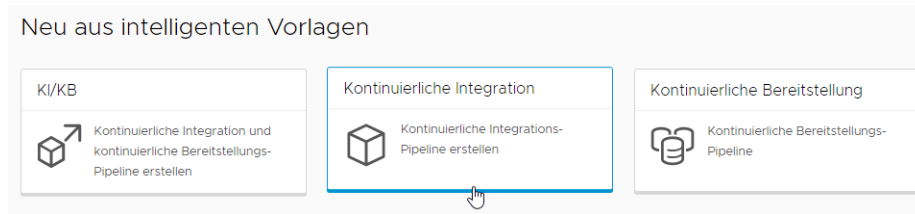
Um den Build vor der Verwendung der intelligenten Pipeline-Vorlage für CI zu planen, sammeln Sie die Informationen für Ihren Build und folgen dann dem CI-Abschnitt von [Planen eines nativen CI/CD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage](#).

Nachdem Sie alle Informationen erfasst und Ihre Anforderungen eingerichtet haben, erstellen Sie eine Pipeline anhand der intelligenten Pipeline-Vorlage für CI.

Unter „Pipelines“ wählen Sie **Intelligente Vorlagen** aus.



Sie wählen die intelligente Pipeline-Vorlage für CI aus.



Um die Pipeline mit den von ihr erstellten Phasen zu speichern, füllen Sie die Vorlage aus und klicken auf **Erstellen**.

Sie können die Pipeline bearbeiten, um alle endgültigen Änderungen vorzunehmen, die Sie möglicherweise benötigen. Anschließend können Sie die Pipeline aktivieren und ausführen. Führen Sie nach der Ausführung der Pipeline folgende Schritte aus:

- Überprüfen Sie, ob die Pipeline erfolgreich ausgeführt wurde. Klicken Sie auf **Ausführungen** und suchen Sie nach Ihrer Pipeline. Wenn Fehler gemeldet wurden, beheben Sie diese und führen Sie die Pipeline erneut aus.
- Überprüfen Sie, ob der Git-Webhook ordnungsgemäß funktioniert. Auf der Git-Registerkarte **Aktivität** werden die Ereignisse angezeigt. Klicken Sie auf **Auslöser > Git > Aktivität**.
- Sehen Sie sich das Dashboard zur Pipeline an und untersuchen Sie die Trends. Klicken Sie auf **Dashboards** und suchen Sie nach dem Dashboard für Ihre Pipeline. Sie können auch ein benutzerdefiniertes Dashboard für Berichte zu weiteren KPIs erstellen.

Ein detailliertes Beispiel finden Sie unter [Vorgehensweise zur kontinuierlichen Integration von Code aus einem GitHub- oder GitLab-Repository in eine Pipeline in vRealize Automation Code Stream](#).

Planen eines nativen CD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage

Um eine CD (Continuous Delivery)-Pipeline in vRealize Automation Code Stream zu erstellen, können Sie die intelligente Pipeline-Vorlage für CD verwenden. Um Ihren nativen CD-Build zu planen, erfassen Sie die Informationen, die Sie zum Ausfüllen der intelligenten Pipeline-Vorlage benötigen, bevor Sie sie zum Erstellen der Pipeline in diesem Beispielplan verwenden.

Wenn Sie die intelligente Pipeline-Vorlage ausfüllen, erstellt sie eine CD-Pipeline in Ihrem Repository und führt die für die Ausführung erforderlichen Aktionen aus. Nachdem Ihre Pipeline ausgeführt wurde, können Sie Trends über die Pipeline-Ausführungen hinweg überwachen.

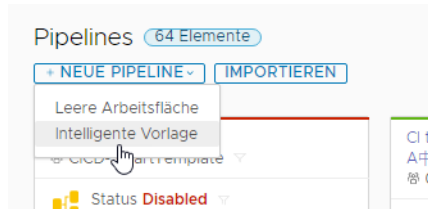
Führen Sie die folgenden Schritte aus, um Ihren Build vor der Verwendung der intelligenten Pipeline-Vorlage für CD zu planen:

- Erfassen Sie die Informationen für Ihren Build und folgen Sie dann dem CD-Teil von [Planen eines nativen CICD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage](#).

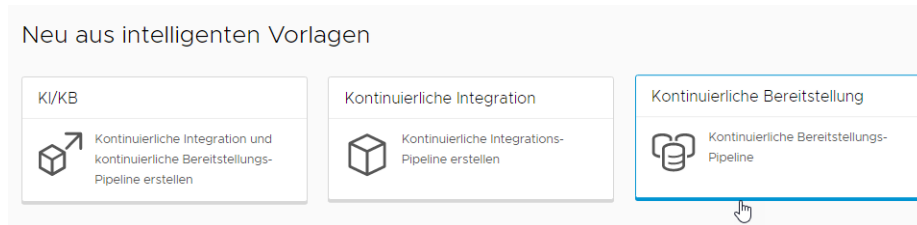
- Fügen Sie einen Kubernetes-Endpoint hinzu, auf dem vRealize Automation Code Stream den Container bereitstellt.
- Bezeichnen Sie ein Projekt, in dem Ihre gesamte Arbeit, einschließlich Pipeline, Endpoints und Dashboards, gruppiert wird.

Nachdem Sie alle Informationen erfasst und Ihre Anforderungen eingerichtet haben, erstellen Sie eine Pipeline aus der intelligenten Pipeline-Vorlage für CD.

Unter „Pipelines“ wählen Sie **Intelligente Vorlagen** aus.



Sie wählen die intelligente Pipeline-Vorlage für CD aus.



Sie füllen die Vorlage aus, geben einen Namen für die Pipeline ein und klicken dann auf **Erstellen**, um die Pipeline mit den von ihr erstellten Phasen zu speichern.

Sie können die Pipeline bearbeiten, um alle endgültigen Änderungen vorzunehmen, die Sie möglicherweise benötigen. Anschließend können Sie die Pipeline aktivieren und ausführen.

Führen Sie nach der Ausführung der Pipeline folgende Schritte aus:

- Stellen Sie sicher, dass Ihre Pipeline erfolgreich war. Klicken Sie auf **Ausführungen** und suchen Sie nach Ihrer Pipeline. Wenn sie fehlgeschlagen ist, korrigieren Sie alle Fehler und führen Sie sie erneut aus.
- Stellen Sie sicher, dass der Git-Webhook ordnungsgemäß funktioniert. Auf der Git-Registerkarte **Aktivität** werden die Ereignisse angezeigt. Klicken Sie auf **Auslöser > Git > Aktivität**.
- Sehen Sie sich das Pipeline-Dashboard an und untersuchen Sie die Trends. Klicken Sie auf **Dashboards** und suchen Sie nach Ihrem Pipeline-Dashboard. Sie können auch ein benutzerdefiniertes Dashboard erstellen, um zusätzliche KPIs zu melden.

Ein detailliertes Beispiel finden Sie unter [Vorgehensweise zur kontinuierlichen Integration von Code aus einem GitHub- oder GitLab-Repository in eine Pipeline in vRealize Automation Code Stream](#).

Planen eines nativen CI/CD-Builds in vRealize Automation Code Stream vor dem manuellen Hinzufügen von Aufgaben

Um eine CI/CD-Pipeline (Continuous Integration and Continuous Delivery) in vRealize Automation Code Stream zu erstellen, können Sie manuell Phasen und Aufgaben hinzufügen. Um Ihren nativen CI/CD-Build zu planen, erfassen Sie die benötigten Informationen, erstellen dann eine Pipeline und fügen manuell Phasen und Aufgaben hinzu.

Sie müssen sowohl die Phase der kontinuierlichen Integration (CI) als auch die der kontinuierlichen Bereitstellung (CD) Ihrer Pipeline planen. Nachdem Sie Ihre Pipeline erstellt und ausgeführt haben, können Sie Trends über die Pipeline-Ausführungen hinweg überwachen.

Um die CI- und CD-Phasen Ihrer Pipeline zu planen, stellen Sie sicher, dass alle Anforderungen erfüllt sind, bevor Sie Ihre Pipeline erstellen.

Planen der externen und internen Anforderungen

Um eine Pipeline aus diesem Beispielplan zu erstellen, verwenden Sie einen Docker-Host, ein Git-Repository, Maven und mehrere Tools für die Nachbearbeitung von Builds.

Endpoints und Repositorys, die Sie benötigen:

- Ein Git-Quellcode-Repository, in das die Entwickler Code einchecken. vRealize Automation Code Stream übergibt den aktuellen Code an die Pipeline, wenn Entwickler Änderungen vornehmen.
- Ein Docker-Endpoint für den Docker-Build-Host, der die Build-Befehle in einem Container ausführt.
- Ein Kubernetes-Endpoint, damit vRealize Automation Code Stream Ihr Image auf einem Kubernetes-Cluster bereitstellen kann.
- Ein Builder-Image, das den Container erstellt, auf dem die Tests für die kontinuierliche Integration ausgeführt werden.
- Ein Image-Registrierungs-Endpoint, damit der Docker-Build-Host das Builder-Image daraus abrufen kann.

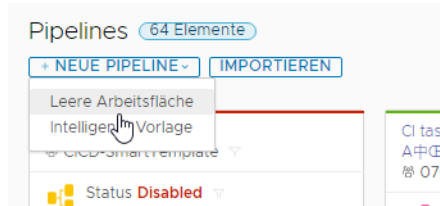
Sie benötigen Zugriff auf ein Projekt. Ein Projekt, in dem Ihre gesamte Arbeit, einschließlich Ihrer Pipeline, Endpoints und Dashboards, gruppiert wird. Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls nicht, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).

Sie brauchen einen Git-Webhook, der es vRealize Automation Code Stream ermöglicht, den Git-Auslöser zu verwenden, der Ihre Pipeline auslöst, wenn Entwickler Codeänderungen vornehmen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwenden des Git-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#).

Vorgehensweise zum Erstellen der CICD-Pipeline und Konfigurieren des Arbeitsbereichs

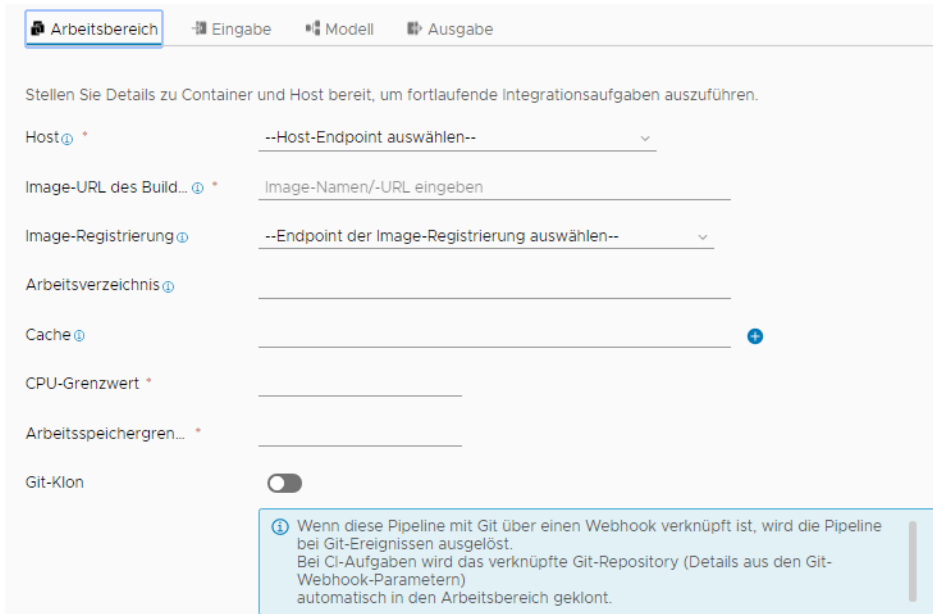
Sie müssen die Pipeline erstellen und dann den Arbeitsbereich, die Pipeline-Eingabeparameter und die Aufgaben konfigurieren.

Um die Pipeline zu erstellen, klicken Sie auf **Pipelines > Neue Pipeline > Leere Arbeitsfläche**.



Geben Sie auf der Registerkarte „Arbeitsbereich“ die Informationen zur kontinuierlichen Integration ein:

- Schließen Sie Ihren Docker-Build-Host ein.
- Geben Sie die URL für Ihr Builder-Image ein.
- Wählen Sie den Endpoint für die Image-Registrierung aus, damit die Pipeline das Image daraus abrufen kann. Der Container führt die CI-Aufgaben aus und stellt Ihr Image bereit. Wenn bei der Registrierung Anmeldedaten benötigt werden, müssen Sie zuerst den Image-Registrierungs-Endpoint erstellen und diesen dann an dieser Stelle auswählen, damit der Host das Image aus der Registrierung abrufen kann.
- Fügen Sie die Artefakte hinzu, die zwischengespeichert werden müssen. Damit ein Build erfolgreich ausgeführt werden kann, werden Artefakte wie zum Beispiel Verzeichnisse als Abhängigkeiten heruntergeladen. Der Cache ist der Speicherort, an dem sich diese Artefakte befinden. Abhängige Artefakte können zum Beispiel das Verzeichnis `.m2` für Maven und das Verzeichnis `node_modules` für Node.js enthalten. Diese Verzeichnisse werden zwischen den Pipeline-Ausführungen zwischengespeichert, um während der Builds Zeit zu sparen.



Konfigurieren Sie auf der Registerkarte „Eingabe“ die Pipeline-Eingabeparameter.

- Wenn in Ihrer Pipeline Eingabeparameter aus einem Git-, Gerrit- oder Docker-Auslöserereignis verwendet werden, wählen Sie den Auslösertyp für Auto-Injektionsparameter aus. Zu den Ereignissen kann die Änderung des Betreffs für Gerrit oder Git oder des Namens des Ereignisbesitzers für Docker zählen. Wenn in Ihrer Pipeline keine vom Ereignis übergebenen Eingabeparameter verwendet werden, behalten Sie für „Parameter automatisch einfügen“ **Keine** bei.
- Um einen Wert und eine Beschreibung auf den Pipeline-Eingabeparameter anzuwenden, klicken Sie auf die drei vertikalen Punkte und dann auf **Bearbeiten**. Der eingegebene Wert wird als Eingabe für Aufgaben, Phasen oder Benachrichtigungen verwendet.
- Um einen Pipeline-Eingabeparameter hinzuzufügen, klicken Sie auf **Hinzufügen**. Sie können beispielsweise `approvers` hinzufügen, um für jede Ausführung einen Standardwert anzuzeigen, der jedoch zur Laufzeit mit einer anderen genehmigenden Person überschrieben werden kann.
- Um einen eingefügten Parameter hinzuzufügen oder zu entfernen, klicken Sie auf **Eingefügten Parameter hinzufügen/entfernen**. Sie können beispielsweise einen nicht verwendeten Parameter entfernen, damit die Ergebnisseite übersichtlicher ist und nur die verwendeten Eingabeparameter angezeigt werden.

Arbeitsbereich **Eingabe** Modell Ausgabe

Eingabeparameter ⓘ

Parameter automati... ☐ Gerrit ☒ Git ☐ Docker ☐ Keine

HINZUFÜGEN EINGEFÜGTE PARAMETER HINZUFÜGEN/ENTFERNEN

| Mit Stern markiert ⓘ | Name | Wert | Beschreibung |
|----------------------|-----------------------|------|--------------|
| ⋮ ☆ | GIT_BRANCH_NAME | | |
| ⋮ ☆ | GIT_CHANGE_SUBJECT | | |
| ⋮ ☆ | GIT_COMMIT_ID | | |
| ⋮ ☆ | GIT_EVENT_DESCRIPTION | | |
| ⋮ ☆ | GIT_EVENT_OWNER_NAME | | |
| ⋮ ☆ | GIT_EVENT_TIMESTAMP | | |
| ⋮ ☆ | GIT_REPO_NAME | | |
| ⋮ ☆ | GIT_SERVER_URL | | |

Konfigurieren Sie die Pipeline, um Ihren Code zu testen:

- Fügen Sie eine CI-Aufgabe hinzu und testen Sie sie.
- Schließen Sie die Schritte zum Ausführen von `mvn test` in Ihrem Code ein.
- Um nach der Ausführung der Aufgabe Probleme zu identifizieren, führen Sie Build-Tools zur Nachbearbeitung aus, z. B. JUnit und JaCoCo, FindBugs und Checkstyle.

Tâche: **Unit-Test** Notifications Restaurer **VALIDER LA TACHE**

Nom de la t... ⓘ Unit-Test

Type ⓘ CI

Continuer e... ☐

exécuter la t... ☒ Toujours ☐ Sur condition

Intégration continue ⓘ

Etapes ⓘ

```

1 cd demo-project
2 mvn test

```

Conserver le... ⓘ

Exporter

Traiter **AJOUTER**

| | | | |
|------------|-------------------|----------------------|---|
| JUnit | <u>JUnit</u> | <u>/demo-project</u> | + |
| JaCoCo | <u>Jacoco</u> | <u>/demo-project</u> | + |
| Checkstyle | <u>Checkstyle</u> | <u>/demo-project</u> | + |
| FindBugs | <u>Findbugs</u> | <u>/demo-project</u> | + |

Paramètres de sortie

[status](#) [exports](#)

Konfigurieren Sie die Pipeline, um Ihren Code zu erstellen:

- Fügen Sie eine CI-Aufgabe hinzu und testen Sie sie.
- Schließen Sie die Schritte ein, mit denen `mvn clean install` auf Ihrem Code ausgeführt wird.
- Schließen Sie den Speicherort und den JAR-Dateinamen ein, damit er Ihr Artefakt beibehält.

Aufgabe: **Build-App** Benachrichtigungen Rollback **AUFGABE VALIDIEREN**

Aufgabenname: **Build-App**

Typ: **CI**

Bei Fehler f... ☐

Aufgabe au... ☒ Immer ☐ Bei Bedingung

Kontinuierliche Integration

Schritte:

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

Artefakte b... **+**

Exportieren

Verarbeiten **HINZUFÜGEN**

| | | | |
|------------|------------|---------------|----------|
| JUnit | JUnit | /demo-project | + |
| JaCoCo | Jacoco | /demo-project | + |
| Checkstyle | Checkstyle | /demo-project | + |
| FindBugs | Findbugs | /demo-project | + |

Ausgabeparameter

status **exports**

Konfigurieren Sie die Pipeline, um Ihr Image auf Ihrem Docker-Host zu veröffentlichen:

- Fügen Sie eine CI-Aufgabe hinzu und testen Sie sie.
- Fügen Sie Schritte hinzu, die Ihr Image überschreiben, exportieren, erstellen und weitergeben.
- Fügen Sie den Exportschlüssel von `IMAGE` für die nächste zu verwendende Aufgabe hinzu.

Aufgabe: **Build-Image** Benachrichtigungen Rollback **AUFGABE VALIDIEREN**

Aufgabenname: **Build-Image**

Typ: **CI**

Bei Fehler fortfa... ☐

Aufgabe ausfüh... ☒ Immer ☐ Bei Bedingung

Kontinuierliche Integration

Schritte:

```
1 cd demo-pro-
2 export IMAGE=automation/demo-cicd-smart-template:${executionIndex}
3 export DOCKER_HOST=tcp://172.17.0.1:2376
4 docker login --username=auto --password=VM
5 docker build -t $IMAGE --file ./docker/Dockerfile .
6 docker push $IMAGE
```

Artefakte beibeh... **+**

Exportieren **IMAGE**

Verarbeiten **HINZUFÜGEN**

Ausgabeparameter

status **exports**

Nachdem Sie den Arbeitsbereich, die Eingabeparameter, die Testaufgaben und die Build-Aufgaben konfiguriert haben, speichern Sie die Pipeline.

Vorgehensweise zum Aktivieren und Ausführen Ihrer Pipeline

Nachdem Sie Ihre Pipeline mit Phasen und Aufgaben konfiguriert haben, können Sie die Pipeline speichern und aktivieren.

Warten Sie, bis die Pipeline ausgeführt und abgeschlossen ist, und vergewissern Sie sich dann, dass der Vorgang erfolgreich war. Wenn sie fehlgeschlagen ist, korrigieren Sie alle Fehler und führen Sie sie erneut aus.

Nach erfolgreicher Ausführung der Pipeline können Sie folgende Punkte bestätigen:

- Untersuchen Sie die Pipeline-Ausführung und zeigen Sie die Ergebnisse der Aufgabenschritte an.
- Suchen Sie im Arbeitsbereich der Pipeline-Ausführung nach den Details zu Ihrem Container und nach dem geklonten Git-Repository.
- Prüfen Sie im Arbeitsbereich die Ergebnisse Ihrer Tools zur Nachbearbeitung und suchen Sie nach Fehlern, Codeabdeckung, Bugs und Formatierungsproblemen.
- Bestätigen Sie, dass Ihr Artefakt beibehalten wurde. Stellen Sie auch sicher, dass das Image mit dem IMAGE-Namen und -Wert exportiert wurde.
- Wechseln Sie zu Ihrem Docker-Repository und stellen Sie sicher, dass die Pipeline Ihren Container veröffentlicht hat.

Ein detailliertes Beispiel, das zeigt, wie der Code von vRealize Automation Code Stream kontinuierlich integriert wird, finden Sie unter [Vorgehensweise zur kontinuierlichen Integration von Code aus einem GitHub- oder GitLab-Repository in eine Pipeline in vRealize Automation Code Stream](#).

Planen für ein Rollback in vRealize Automation Code Stream

Wenn eine Pipeline-Ausführung fehlschlägt, können Sie mithilfe eines Rollbacks Ihre Umgebung in einen früheren stabilen Zustand zurückversetzen. Um das Rollback zu verwenden, planen Sie einen Rollback-Ablauf und machen Sie sich damit vertraut, wie dieser implementiert wird.

In einem Rollback-Ablauf sind die erforderlichen Schritte festgelegt, die ausgeführt werden müssen, um einen Fehler bei der Bereitstellung rückgängig zu machen. Der Ablauf hat die Form einer Rollback-Pipeline, die eine oder mehrere sequenzielle Aufgaben enthält, welche je nach Typ der ausgeführten und fehlgeschlagenen Bereitstellung variieren. Die Bereitstellung und das Rollback einer herkömmlichen Anwendung unterscheiden sich beispielsweise von der Bereitstellung und dem Rollback einer Containeranwendung.

Zur Wiederherstellung eines funktionierenden Bereitstellungszustands enthält eine Rollback-Pipeline in der Regel Aufgaben zum:

- Bereinigen von Zuständen oder Umgebungen.
- Ausführen eines benutzerdefinierten Skripts, um Änderungen rückgängig zu machen.
- Bereitstellen einer vorherigen Revision einer Bereitstellung.

Um ein Rollback zu einer vorhandenen Bereitstellungs-Pipeline hinzuzufügen, hängen Sie vor deren Ausführung die Rollback-Pipeline an die Aufgaben oder Phasen in der Bereitstellungs-Pipeline an, für die Sie das Rollback durchführen möchten.

Vorgehensweise zum Konfigurieren eines Rollbacks

Um das Rollback in Ihrer Bereitstellung zu konfigurieren, müssen Sie Folgendes tun:

- Erstellen Sie eine Bereitstellungs-Pipeline.
- Ermitteln Sie potenzielle Fehlerpunkte in der Bereitstellungs-Pipeline, die das Rollback auslösen, damit Sie Ihre Rollback-Pipeline anhängen können. Sie können Ihre Rollback-Pipeline beispielsweise an einen Bedingungs- oder Abfrageaufgabentyp in der Bereitstellungs-Pipeline anhängen, der prüft, ob eine vorherige Aufgabe erfolgreich abgeschlossen wurde. Informationen zu Bedingungsaufgaben finden Sie unter [Vorgehensweise zum Verwenden von Variablenbindungen in einer Bedingungs Aufgabe zum Ausführen oder Anhalten einer Pipeline in vRealize Automation Code Stream](#).
- Ermitteln Sie den Umfang des Fehlers, durch den die Rollback-Pipeline ausgelöst wird, beispielsweise ein Fehler in einer Aufgabe oder Phase. Sie können ein Rollback auch an eine Phase anhängen.
- Legen Sie fest, welche Rollback-Aufgabe(n) im Falle eines Fehlers ausgeführt werden soll(en). Mit diesen Aufgaben erstellen Sie Ihre Rollback-Pipeline.

Sie haben die Möglichkeit, eine Rollback-Pipeline manuell zu erstellen oder von vRealize Automation Code Stream erstellen zu lassen.

- Mithilfe einer leeren Arbeitsfläche können Sie manuell eine Rollback-Pipeline erstellen, die einem Ablauf parallel zu einer vorhandenen Bereitstellungs-Pipeline folgt. Anschließend hängen Sie die Rollback-Pipeline an eine oder mehrere Aufgaben in der Bereitstellungs-Pipeline an, die bei einem Fehler ein Rollback auslösen.
- Mithilfe einer intelligenten Pipeline-Vorlage können Sie eine Bereitstellungs-Pipeline mit der Rollback-Aktion konfigurieren. Anschließend erstellt vRealize Automation Code Stream automatisch eine oder mehrere standardmäßige Rollback-Pipelines mit vordefinierten Aufgaben, mit denen die Bereitstellung bei einem Fehler zurückgesetzt wird.

Ein detailliertes Beispiel für das Konfigurieren einer CD-Pipeline mit Rollback unter Verwendung einer intelligenten Pipeline-Vorlage finden Sie unter [Vorgehensweise zum Rollback meiner Bereitstellung in vRealize Automation Code Stream](#).

Was geschieht, wenn eine Bereitstellungs-Pipeline mehrere Aufgaben oder Phasen mit Rollback aufweist?

Wenn Sie mehrere Aufgaben bzw. Aufgaben und Phasen mit Rollback hinzugefügt haben, beachten Sie, dass die Rollback-Abfolge variiert.

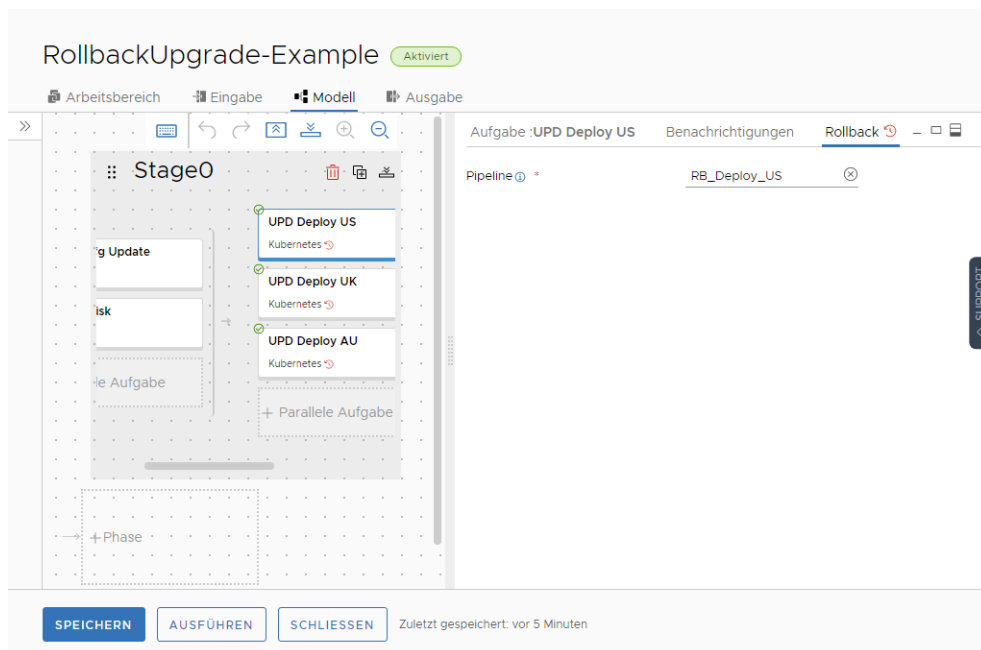
Tabelle 4-2. Festlegen der Rollback-Abfolge

| Elemente, denen das Rollback hinzugefügt wird | Ausführung des Rollbacks |
|---|---|
| Parallele Aufgaben | Wenn eine der parallelen Aufgaben fehlschlägt, wird ein Rollback für diese Aufgabe ausgeführt, nachdem alle parallelen Aufgaben abgeschlossen wurden oder fehlgeschlagen sind. Das Rollback wird nicht sofort nach dem Fehlschlagen der Aufgabe ausgeführt. |
| Sowohl die Aufgabe innerhalb einer Phase als auch die Phase | Wenn eine Aufgabe fehlschlägt, wird das Rollback der Aufgabe ausgeführt. Falls die Aufgabe zu einer Gruppe paralleler Aufgaben gehört, wird das Rollback der Aufgabe ausgeführt, nachdem alle parallelen Aufgaben abgeschlossen wurden oder fehlgeschlagen sind. Wenn das Rollback der Aufgabe abgeschlossen wird oder nicht abgeschlossen werden kann, wird das Rollback der Phase ausgeführt. |

Angenommen, Sie haben eine Pipeline mit:

- Einer Produktionsphase mit Rollback.
- Einer Gruppe paralleler Aufgaben mit jeweils eigenem Rollback für jede Aufgabe.

Die Aufgabe **UPD Deploy US** verfügt über die Rollback-Pipeline **RB_Deploy_US**. Wenn **UPD Deploy US** fehlschlägt, folgt das Rollback dem in der Pipeline **RB_Deploy_US** definierten Ablauf.



Wenn **UPD Deploy US** fehlschlägt, wird die Pipeline **RB_Deploy_US** ausgeführt, nachdem **UPD Deploy UK** und **UPD Deploy AU** ebenfalls abgeschlossen worden oder fehlgeschlagen sind. Das Rollback wird nicht sofort nach dem Fehlschlagen von **UPD Deploy US** ausgeführt. Da die Produktionsphase ebenfalls über ein Rollback verfügt, wird nach der Ausführung der **RB_Deploy_US**-Pipeline die Pipeline für das Rollback der Phase ausgeführt.

Lernprogramme für die Verwendung von vRealize Automation Code Stream

5

vRealize Automation Code Stream modelliert und unterstützt Ihren DevOps-Versionslebenszyklus und testet und gibt Ihre Anwendungen kontinuierlich in Entwicklungsumgebungen und Produktionsumgebungen frei.

Sie haben bereits alle erforderlichen Einstellungen konfiguriert, um vRealize Automation Code Stream verwenden zu können. Weitere Informationen finden Sie unter [Kapitel 2 Einrichten von vRealize Automation Code Stream zum Modellieren des Freigabeprozesses](#).

Sie können jetzt Pipelines erstellen, die die Erstellung und den Test von Entwicklercode automatisieren, bevor Sie ihn in der Produktionsumgebung freigeben. Sie können mit vRealize Automation Code Stream containerbasierte oder herkömmliche Anwendungen bereitstellen.

Tabelle 5-1. Verwenden von vRealize Automation Code Stream in Ihrem DevOps-Lebenszyklus

| Funktionen | Beispiele für Anwendungsmöglichkeiten |
|--|---|
| Verwenden der nativen Build-Funktion in vRealize Automation Code Stream. | <p>CICD-, CI- und CD-Pipelines erstellen, die Ihren Code kontinuierlich integrieren, containerisieren und liefern.</p> <ul style="list-style-type: none">■ Eine intelligente Pipeline-Vorlage verwenden, die eine Pipeline erstellt.■ Phasen und Aufgaben einer Pipeline manuell hinzufügen. |
| Freigeben Ihrer Anwendungen und Automatisieren der Versionen. | <p>Integrieren und Freigeben Ihrer Anwendungen auf verschiedene Arten.</p> <ul style="list-style-type: none">■ Kontinuierliche Integration Ihres Code aus einem GitHub- oder GitLab-Repository in Ihre Pipeline.■ Automatisieren der Bereitstellung Ihrer Anwendung mithilfe einer YAML-Cloud-Vorlage.■ Automatisieren der Bereitstellung Ihrer Anwendung in einem Kubernetes-Cluster.■ Freigeben Ihrer Anwendung in einer Blau/Grün-Bereitstellung.■ Integrieren von vRealize Automation Code Stream in Ihre eigenen Build-, Test- und Bereitstellungstools.■ Verwenden einer REST API, die vRealize Automation Code Stream mit anderen Anwendungen integriert. |

Tabelle 5-1. Verwenden von vRealize Automation Code Stream in Ihrem DevOps-Lebenszyklus (Fortsetzung)

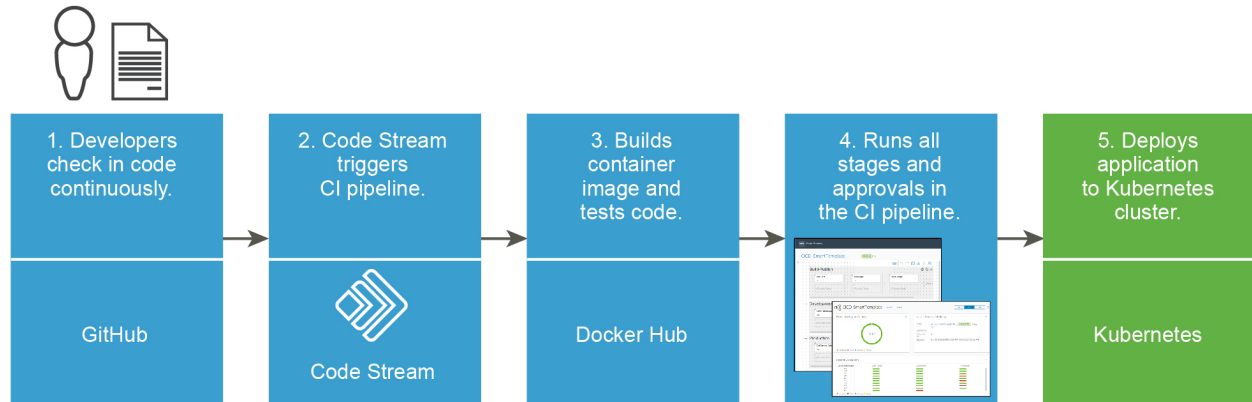
| Funktionen | Beispiele für Anwendungsmöglichkeiten |
|--|---|
| Verfolgen Sie Trends, Metriken und wichtige Leistungsindikatoren (Key Performance Indicators, KPIs). | Erstellen von benutzerdefinierten Dashboards und Einblicke in die Leistung Ihrer Pipelines. |
| Beheben von Problemen. | Wenn eine Pipeline-Ausführung fehlschlägt, müssen Sie über vRealize Automation Code Stream ein JIRA-Ticket erstellen. |

Dieses Kapitel enthält die folgenden Themen:

- [Vorgehensweise zur kontinuierlichen Integration von Code aus einem GitHub- oder GitLab-Repository in eine Pipeline in vRealize Automation Code Stream](#)
- [Vorgehensweise zum Automatisieren der Version einer Anwendung, die von einer YAML-Cloud-Vorlage in vRealize Automation Code Stream bereitgestellt wird](#)
- [Vorgehensweise zum Automatisieren der Freigabe einer Anwendung in vRealize Automation Code Stream in einem Kubernetes-Cluster](#)
- [Vorgehensweise zum Bereitstellen meiner Anwendung in vRealize Automation Code Stream für meine Blau/Grün-Bereitstellung](#)
- [Vorgehensweise zum Integrieren von eigenen Build-, Test- und Bereitstellungstools mit vRealize Automation Code Stream](#)
- [Vorgehensweise zum Verwenden der Ressourceneigenschaften einer Cloud-Vorlagentaufgabe in der nächsten Aufgabe](#)
- [Vorgehensweise zum Verwenden einer REST API für die Integration von vRealize Automation Code Stream in andere Anwendungen](#)

Vorgehensweise zur kontinuierlichen Integration von Code aus einem GitHub- oder GitLab-Repository in eine Pipeline in vRealize Automation Code Stream

Als Entwickler möchten Sie Ihren Code kontinuierlich aus einem GitHub oder GitLab Enterprise-Repository integrieren. Wenn Ihre Entwickler ihren Code aktualisieren und Änderungen an das Repository übergeben, kann vRealize Automation Code Stream diese Änderungen überwachen und die Pipeline auslösen.



Damit vRealize Automation Code Stream Ihre Pipeline bei Codeänderungen auslösen kann, verwenden Sie den Git-Auslöser. vRealize Automation Code Stream löst Ihre Pipeline jedes Mal dann aus, wenn Sie Änderungen an Ihrem Code festschreiben.

Um Ihren Code zu erstellen, verwenden Sie einen Docker-Host. Sie verwenden JUnit und JaCoCo als Test-Framework-Tools, die Komponententests und Codeabdeckung ausführen und in Ihrer Pipeline aufgenommen werden.

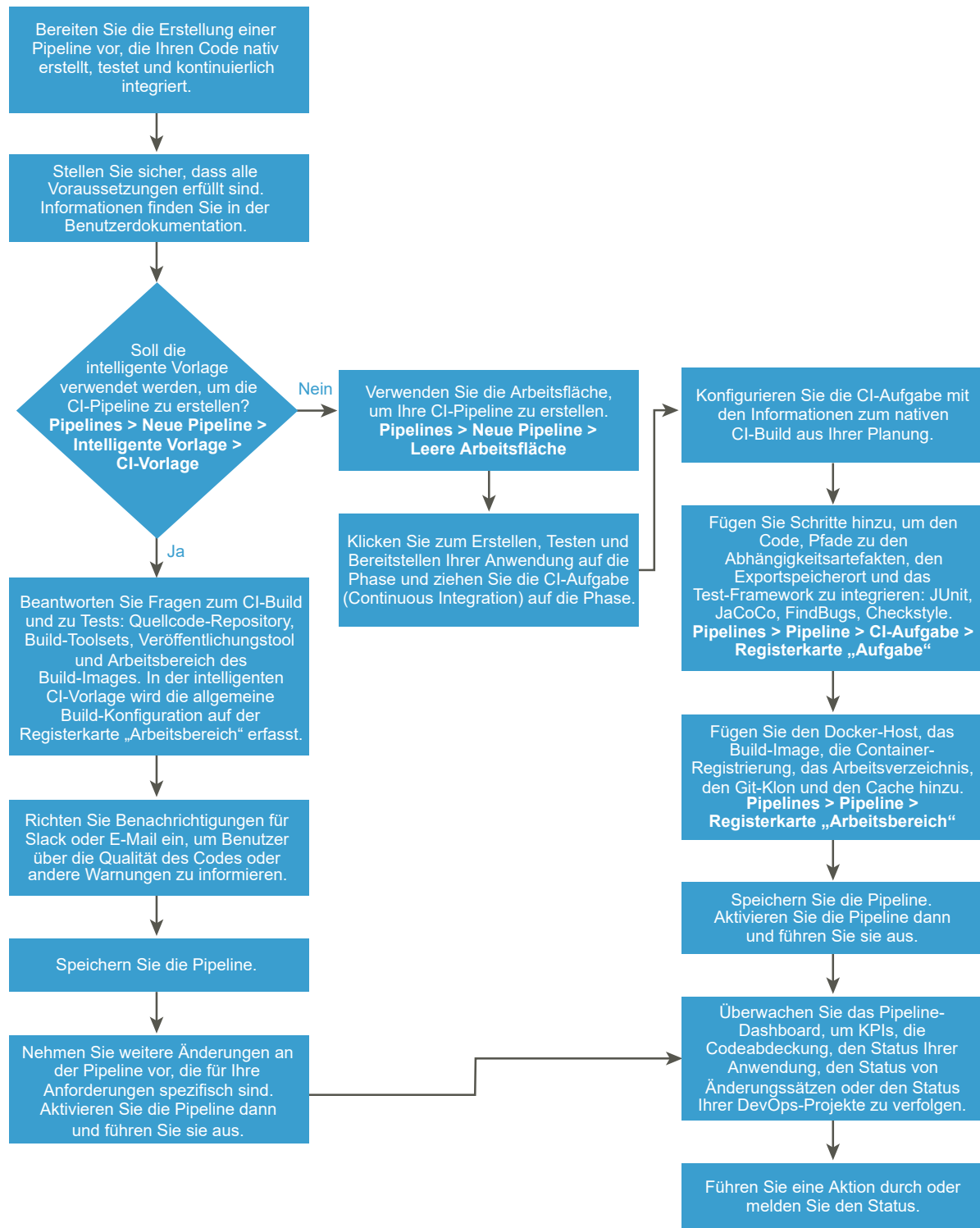
Dann können Sie die Smart-Pipeline-Vorlage für die kontinuierliche Integration (CI) verwenden, die eine Pipeline für kontinuierliche Integration erstellt, die Ihren Code erstellt, testet und auf dem Kubernetes-Cluster Ihres Projektteams auf AWS bereitstellt. Um die Codeabhängigkeits-Artefakte für Ihre CI-Aufgabe zu speichern, wodurch Zeit bei Code-Builds eingespart wird, verwenden Sie einen Cache.

In der Pipeline-Aufgabe, die Ihren Code erstellt und testet, können Sie mehrere kontinuierliche Integrationsschritte hinzufügen. Diese Schritte können sich im selben Arbeitsverzeichnis befinden, in dem vRealize Automation Code Stream den Quellcode beim Auslösen der Pipeline kloniert.

Um den Code auf dem Kubernetes-Cluster bereitzustellen, können Sie eine Kubernetes-Aufgabe in Ihrer Pipeline verwenden. Anschließend müssen Sie Ihre Pipeline aktivieren und ausführen. Nehmen Sie dann eine Änderung an Ihrem Code im Repository vor und überwachen Sie den Pipeline-Auslöser. Um Ihre Pipeline-Trends nach der Ausführung Ihrer Pipeline zu überwachen und darüber zu berichten, verwenden Sie die Dashboards.

Das folgende Flussdiagramm zeigt den Workflow, den Sie nutzen können, wenn Sie eine intelligente Pipeline-Vorlage zum Erstellen Ihrer Pipeline verwenden oder die Pipeline manuell erstellen.

Abbildung 5-1. Workflow, der eine intelligente Pipeline-Vorlage verwendet oder eine Pipeline manuell erstellt



Im folgenden Beispiel verwenden Sie die intelligente Pipeline-Vorlage für kontinuierliche Integration, um eine Pipeline für kontinuierliche Integration zu erstellen, die Ihren Code kontinuierlich in Ihre Pipeline integriert.

Auf Wunsch können Sie die Pipeline manuell erstellen und ihr Phasen und Aufgaben hinzufügen. Weitere Informationen zum Planen eines kontinuierlichen Integrations-Builds und zum manuellen Erstellen der Pipeline finden Sie unter [Planen eines nativen CICD-Builds in vRealize Automation Code Stream vor dem manuellen Hinzufügen von Aufgaben](#).

Voraussetzungen

- Planen Sie Ihren kontinuierlichen Integrations-Build. Weitere Informationen hierzu finden Sie unter [Planen eines nativen CI-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage](#).
- Stellen Sie sicher, dass ein GitLab-Quellcode-Repository vorhanden ist. Weitere Informationen erhalten Sie bei Ihrem vRealize Automation Code Stream-Administrator.
- Fügen Sie einen Git-Endpoint hinzu. Ein Beispiel finden Sie unter [Vorgehensweise zum Verwenden des Git-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#).
- Damit vRealize Automation Code Stream die Änderungen in Ihrem GitHub- oder GitLab-Repository überwachen und beim Auftreten von Änderungen eine Pipeline auslösen kann, fügen Sie einen Webhook hinzu. Ein Beispiel finden Sie unter [Vorgehensweise zum Verwenden des Git-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#).
- Fügen Sie einen Docker-Host-Endpoint hinzu, der einen Container für die CI-Aufgabe (kontinuierliche Integration) erstellt, den mehrere CI-Aufgaben verwenden können. Weitere Informationen zu Endpoints finden Sie unter [Definition von Endpoints in vRealize Automation Code Stream](#).
- Rufen Sie die Image-URL, den Build-Host und die URL für das Build-Image ab. Weitere Informationen erhalten Sie bei Ihrem vRealize Automation Code Stream-Administrator.
- Stellen Sie sicher, dass Sie JUnit und JaCoCo für Ihre Test-Framework-Tools verwenden.
- Richten Sie eine externe Instanz für Ihren CI-Build ein: Jenkins, TFS oder Bamboo. Das Kubernetes-Plug-In stellt Ihren Code bereit. Weitere Informationen erhalten Sie bei Ihrem vRealize Automation Code Stream-Administrator.

Verfahren

- 1 Befolgen Sie die Voraussetzungen.

- 2 Um die Pipeline mithilfe der intelligenten Pipeline-Vorlage zu erstellen, öffnen Sie die intelligente Pipeline-Vorlage für die kontinuierliche Integration und füllen Sie das Formular aus.
 - a Klicken Sie auf **Pipelines > Neue Pipeline > Intelligente Vorlage > Kontinuierliche Integration**.
 - b Beantworten Sie die Fragen in der Vorlage zu Ihrem Quellcode-Repository, zu den Build-Toolsets, dem Veröffentlichungstool und dem Build-Image-Arbeitsbereich.
 - c Fügen Sie Slack- oder E-Mail-Benachrichtigungen für Ihr Team hinzu.
 - d Damit die intelligente Pipeline-Vorlage die Pipeline erstellt, klicken Sie auf **Erstellen**.
 - e Um weitere Änderungen an der Pipeline vorzunehmen, klicken Sie auf **Bearbeiten**, nehmen Sie Ihre Änderungen vor und klicken Sie auf **Speichern**.
 - f Aktivieren Sie die Pipeline und führen Sie sie aus.
- 3 Um die Pipeline manuell zu erstellen, fügen Sie der Arbeitsfläche Phasen und Aufgaben hinzu und beziehen Sie Ihre nativen CI-Build-Informationen in die CI-Aufgabe mit ein.
 - a Klicken Sie auf **Pipelines > Neue Pipeline > Leere Arbeitsfläche**.
 - b Klicken Sie auf die Phase und ziehen Sie dann die verschiedenen CI-Aufgaben aus dem Navigationsbereich auf die Phase.
 - c Klicken Sie die CI-Aufgabe zum Konfigurieren an und wählen Sie die Registerkarte **Aufgabe**.
 - d Fügen Sie die Schritte hinzu, die Ihren Code kontinuierlich integrieren.
 - e Schließen Sie die Pfade zu den Abhängigkeits-Artefakten ein.
 - f Fügen Sie den Exportspeicherort hinzu.
 - g Fügen Sie die Test-Framework-Tools hinzu, die Sie verwenden möchten.
 - h Fügen Sie den Docker-Host und das Build-Image hinzu.
 - i Fügen Sie die Container-Registrierung, das Arbeitsverzeichnis und den Cache hinzu.
 - j Speichern Sie die Pipeline und aktivieren Sie sie.
- 4 Nehmen Sie eine Änderung an Ihrem Code in Ihrem GitHub-oder GitLab-Repository vor. Der Git-Auslöser aktiviert Ihre Pipeline, deren Ausführung beginnt.
- 5 Wenn Sie überprüfen möchten, ob die Codeänderung die Pipeline ausgelöst hat, klicken Sie auf **Auslöser > Git > Aktivität**.

- 6 Um die Ausführung für Ihre Pipeline anzuzeigen, klicken Sie auf **Ausführungen** und vergewissern Sie sich, dass die Schritte zum Erstellen und Exportieren Ihres Build-Images durchgeführt wurden.

The screenshot shows the vRealize Automation Code Stream interface. On the left is a sidebar with navigation options: Dashboards, Executions, User Operations, Pipelines, Manage (with sub-items: Endpoints, Variables, Triggers, Gerrit, Git), and Triggers. The main content area displays the execution details for a pipeline named 'CICD-SmartTemplate #51'. The pipeline is in a 'COMPLETED' state. The execution progress is shown in two stages: 'Build-Publish' and 'Development'. The 'Build-Publish' stage includes tasks 'Unit-Test', 'Build-App', and 'Build-Image'. The 'Build-Image' task is highlighted and shows a 'COMPLETED' status with the message 'Execution Completed.'. The 'Development' stage includes tasks 'Create Namespace', 'Create Secret', 'Create Service', and 'Create Deployment'. The 'Build-Image' task details are shown below, including its name, type (CI), status, duration, and a log of the steps executed. The log shows the following steps: 'set -e', 'cd demo-project', 'export IMAGE=automationbeta/demo-cicd-smart-template:51', 'export DOCKER_HOST=tcp://18.211.211.27:4243', 'docker login --username=automation --password=VM', a warning about password security, 'Login Succeeded', 'docker build -t automation/cicd-smart-template:51 --file ./docker/Dockerfile', and 'Sending build context to Docker daemon 1529MB'. The 'Preserved Artifacts' section shows the path '/sharedPath/pipelines/CICD-SmartTemplate/51/Build-Publish/Build-Image/artifacts/'. The 'Exports' section shows a table with 'Exported' and 'Value' columns, containing the entry 'IMAGE automation/cicd-smart-template:51'. The 'Process' section shows 'No process results available.' and the 'Input' section is empty.

- 7 Um das Pipeline-Dashboard zu überwachen, sodass Sie KPIs und Trends verfolgen können, klicken Sie auf **Dashboards > Pipeline-Dashboards**.

Ergebnisse

Herzlichen Glückwunsch! Sie haben eine Pipeline erstellt, die Ihren Code kontinuierlich aus einem GitHub- oder GitLab-Repository in Ihre Pipeline integriert und Ihr Build-Image bereitstellt.

Nächste Schritte

Weitere Informationen finden Sie unter [Weitere Ressourcen für vRealize Automation Code Stream-Administratoren und -Entwickler](#).

Vorgehensweise zum Automatisieren der Version einer Anwendung, die von einer YAML-Cloud-Vorlage in vRealize Automation Code Stream bereitgestellt wird

Als Entwickler benötigen Sie eine Pipeline, die bei jeder Übergabe einer Änderung eine Cloud-Automatisierungsvorlage aus einer lokalen GitHub-Instanz abruft. Sie benötigen die Pipeline, um eine WordPress-Anwendung für Amazon Web Services (AWS) EC2 oder ein Datencenter bereitzustellen. vRealize Automation Code Stream ruft die Cloud-Vorlage aus der Pipeline auf und automatisiert die kontinuierliche Integration und die kontinuierliche Bereitstellung (CICD) dieser Cloud-Vorlage, um Ihre Anwendung bereitzustellen.

Um Ihre Pipeline zu erstellen und auszulösen, benötigen Sie eine VMware Cloud-Vorlage.

Für die **Cloud-Vorlagenquelle** in Ihrer vRealize Automation Code Stream-Cloud-Vorlagenaufgabe können Sie eine der folgenden Optionen auswählen:

- **Cloud Assembly-Vorlage** als Quellcodeverwaltung. In diesem Fall benötigen Sie kein GitLab- oder GitHub-Repository.
- **Quellcodeverwaltung**, wenn Sie GitLab oder GitHub für die Quellcodeverwaltung verwenden. In diesem Fall benötigen Sie einen Webhook und müssen die Pipeline über den Webhook auslösen.

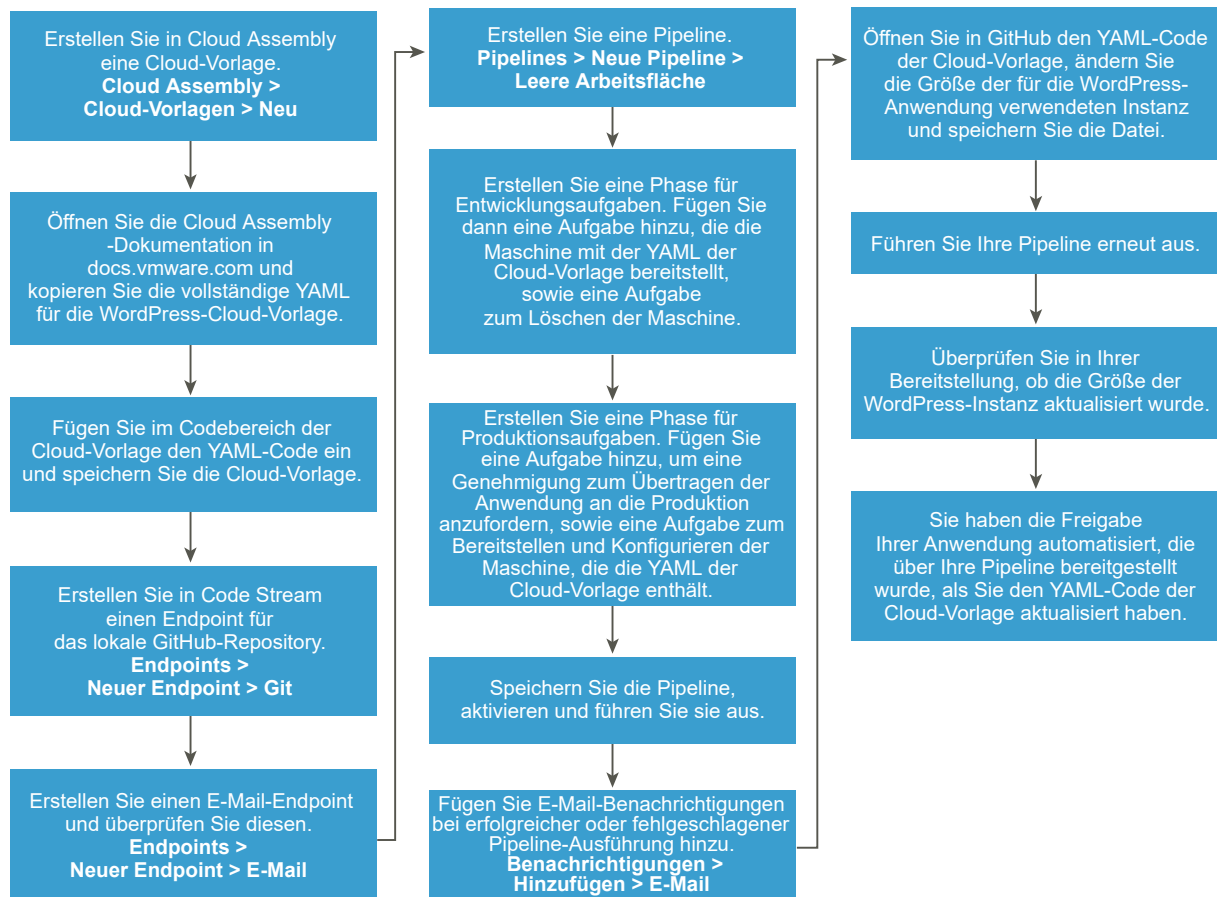
Wenn Ihr GitHub-Repository eine YAML-Cloud-Vorlage enthält und Sie diese Cloud-Vorlage in Ihrer Pipeline verwenden möchten, müssen Sie wie folgt vorgehen.

- 1 Übertragen Sie die Cloud-Vorlage in vRealize Automation Cloud Assembly in Ihr GitHub-Repository.
- 2 Erstellen Sie in vRealize Automation Code Stream einen Git-Endpoint. Erstellen Sie anschließend einen Git-Webhook, der Ihren Git-Endpoint und Ihre Pipeline verwendet.
- 3 Aktualisieren Sie zum Auslösen der Pipeline eine beliebige Datei im GitHub-Repository und übergeben Sie die Änderung.

Falls Ihr GitHub-Repository keine YAML-Cloud-Vorlage enthält und Sie eine Cloud-Vorlage aus der Quellcodeverwaltung verwenden möchten, erfahren Sie im folgenden Verfahren, wie Sie dabei vorgehen müssen. Ihnen wird gezeigt, wie Sie eine Cloud-Vorlage für eine WordPress-Anwendung erstellen und diese über ein lokales GitHub-Repository auslösen. Immer dann, wenn Sie eine Änderung an der YAML-Cloud-Vorlage vornehmen, löst die Pipeline die Freigabe Ihrer Anwendung aus und automatisiert sie.

- In vRealize Automation Cloud Assembly fügen Sie ein Cloud-Konto und eine Cloud-Zone hinzu und erstellen die Cloud-Vorlage.
- In vRealize Automation Code Stream fügen Sie einen Endpoint für das lokale GitHub-Repository hinzu, in dem Ihre Cloud-Vorlage gehostet wird. Anschließend fügen Sie die Cloud-Vorlage Ihrer Pipeline hinzu.

In diesem Anwendungsbeispiel wird gezeigt, wie Sie eine Cloud-Vorlage aus einem lokalen GitHub-Repository verwenden.

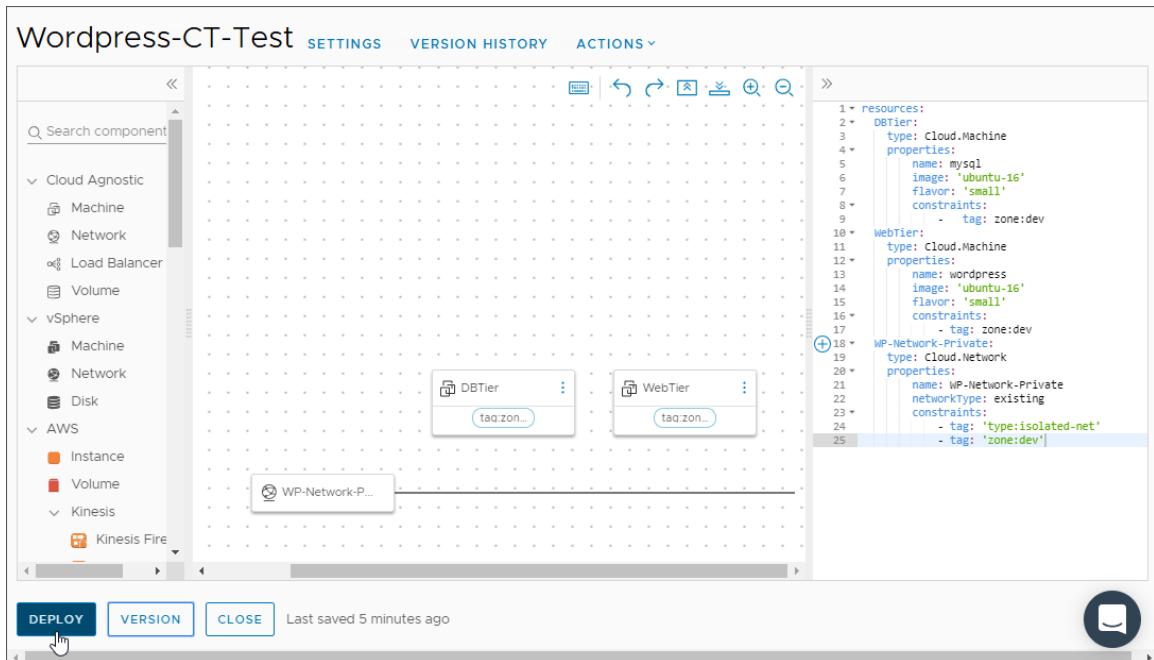


Voraussetzungen

- Fügen Sie ein Cloud-Konto und eine Cloud-Zone zu Ihrer vRealize Automation Cloud Assembly-Infrastruktur hinzu. Weitere Informationen finden Sie in der Dokumentation zu vRealize Automation Cloud Assembly.
- Um Ihre Cloud-Vorlage im folgenden Verfahren zu erstellen, kopieren Sie den WordPress-YAML-Code in die Zwischenablage. Weitere Informationen zum YAML-Code der Cloud-Vorlage im WordPress-Anwendungsfall finden Sie in der Dokumentation zu vRealize Automation Cloud Assembly.
- Fügen Sie den YAML-Code für die WordPress-Anwendung zu Ihrer GitHub-Instanz hinzu.
- Fügen Sie einen Webhook für den Git-Auslöser hinzu, damit Ihre Pipeline den YAML-Code immer dann abrufen kann, wenn Sie Änderungen daran übernehmen. Klicken Sie in vRealize Automation Code Stream auf **Auslöser > Git > Webhooks für Git**.
- Um mit einer Cloud-Vorlagenaufgabe zu arbeiten, müssen Sie über eine der vRealize Automation Cloud Assembly-Rollen verfügen.

Verfahren

- 1 Führen Sie in vRealize Automation Cloud Assembly die folgenden Schritte aus.
 - a Klicken Sie auf **VMware Cloud Templates** und erstellen Sie dann eine Cloud-Vorlage und eine Bereitstellung für die WordPress-Anwendung.
 - b Fügen Sie den WordPress-YAML-Code, den Sie in Ihre Zwischenablage kopiert haben, in Ihre Cloud-Vorlage ein und stellen Sie ihn bereit.



2 Erstellen Sie Endpoints in vRealize Automation Code Stream.

- a Erstellen Sie einen Git-Endpoint für Ihr lokales GitHub-Repository, in dem sich Ihre YAML-Datei befindet.
- b Fügen Sie einen E-Mail-Endpoint hinzu, der Benutzer über den Pipeline-Status benachrichtigen kann, wenn er ausgeführt wird.

Neuer Endpoint

Projekt * Codestream

Typ * Email

Name * Wert

Beschreibung

Als eingeschränkt ke... ☐ nicht eingeschränkt

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

VARIABLE ERSTELLEN

ERSTELLEN ÜBERPRÜFEN ABBRECHEN

- Erstellen Sie eine Pipeline und fügen Sie Benachrichtigungen für eine erfolgreiche und fehlerhafte Ausführung hinzu.

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 Fügen Sie eine Phase für die Bereitstellung und eine Cloud-Vorlagenaufgabe hinzu.
- a Fügen Sie eine Cloud-Vorlagenaufgabe hinzu, die die Maschine bereitstellt, und konfigurieren Sie die Aufgabe so, dass Sie die Cloud-Vorlagen-YAML für die WordPress-Anwendung verwendet.

```
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 Fügen Sie eine Cloud-Vorlagenaufgabe zum Löschen der Maschine hinzu, um Ressourcen freizugeben.

5 Fügen Sie eine Phase für die Produktion hinzu und schließen Sie Genehmigungs- und Bereitstellungsaufgaben ein.

- a Fügen Sie eine Benutzervorgangsaufgabe hinzu, um die Genehmigung für die Übertragung der WordPress-Anwendung auf die Produktionsebene anzufordern.
- b Fügen Sie eine Cloud-Vorlagenaufgabe hinzu, um die Maschine bereitzustellen, und konfigurieren Sie sie mit der Cloud-Vorlagen-YAML für die WordPress-Anwendung.

Wenn Sie **Erstellen** auswählen, muss der Bereitstellungsname eindeutig sein. Wenn Sie den Namen leer lassen, weist vRealize Automation Code Stream einen zufälligen eindeutigen Namen zu.

Sie müssen Folgendes wissen, wenn Sie in Ihrem eigenen Anwendungsbeispiel **Rollback** auswählen: Wenn Sie die Aktion **Rollback** auswählen und eine **Rollback-Version** eingeben, muss die Version in folgender Form angegeben werden: **n-x**. Beispiel: **n-1**, **n-2**, **n-3** usw. Wenn Sie die Bereitstellung an einem anderen Speicherort als vRealize Automation Code Stream erstellen und aktualisieren, ist das Rollback zulässig.

Wenn Sie sich bei vRealize Automation Code Stream anmelden, wird ein Benutzertoken abgerufen, das 30 Minuten lang gültig ist. Wenn die Aufgabe vor der Cloud-Vorlagenaufgabe 30 Minuten oder länger dauert, läuft das Benutzertoken bei lang andauernden Pipeline-Ausführungen ab. Dies führt dazu, dass die Cloud-Vorlagenaufgabe fehlschlägt.

Um sicherzustellen, dass Ihre Pipeline mehr als 30 Minuten lang ausgeführt werden kann, können Sie ein optionales API-Token eingeben. Wenn vRealize Automation Code Stream die Cloud-Vorlage aufruft, bleibt das API-Token bestehen und wird weiterhin von der Cloud-Vorlagenaufgabe verwendet.

Wenn Sie das API-Token als Variable verwenden, wird es verschlüsselt. Andernfalls wird es als Klartext verwendet.

6 Führen Sie die Pipeline aus.

Um sicherzustellen, dass jede Aufgabe erfolgreich abgeschlossen wurde, klicken Sie auf die Aufgabe in der Ausführung und überprüfen Sie den Status in den Bereitstellungsdetails, in denen detaillierte Ressourceninformationen angezeigt werden.

7 Ändern Sie in GitHub den Typ der WordPress-Serverinstanz von `small` in `medium`.

Wenn Sie Änderungen übernehmen, wird die Pipeline ausgelöst. Sie ruft Ihren aktualisierten Code aus dem GitHub-Repository ab und erstellt Ihre Anwendung.

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

8 Führen Sie die Pipeline erneut aus, stellen Sie sicher, dass sie erfolgreich war und dass sie den Typ der WordPress-Instanz von `small` in `medium` geändert hat.

Ergebnisse

Herzlichen Glückwunsch! Sie haben die Version Ihrer Anwendung automatisiert, die Sie über eine YAML-Cloud-Vorlage bereitgestellt haben.

Nächste Schritte

Weitere Informationen zur Verwendung von vRealize Automation Code Stream finden Sie unter [Kapitel 5 Lernprogramme für die Verwendung von vRealize Automation Code Stream](#).

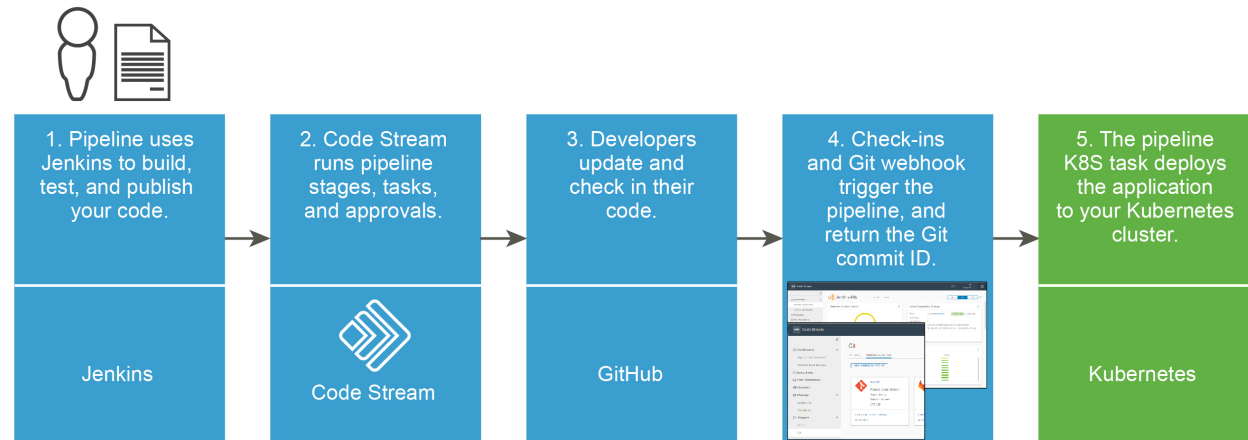
Weitere Referenzen finden Sie unter [Weitere Ressourcen für vRealize Automation Code Stream-Administratoren und -Entwickler](#).

Vorgehensweise zum Automatisieren der Freigabe einer Anwendung in vRealize Automation Code Stream in einem Kubernetes-Cluster

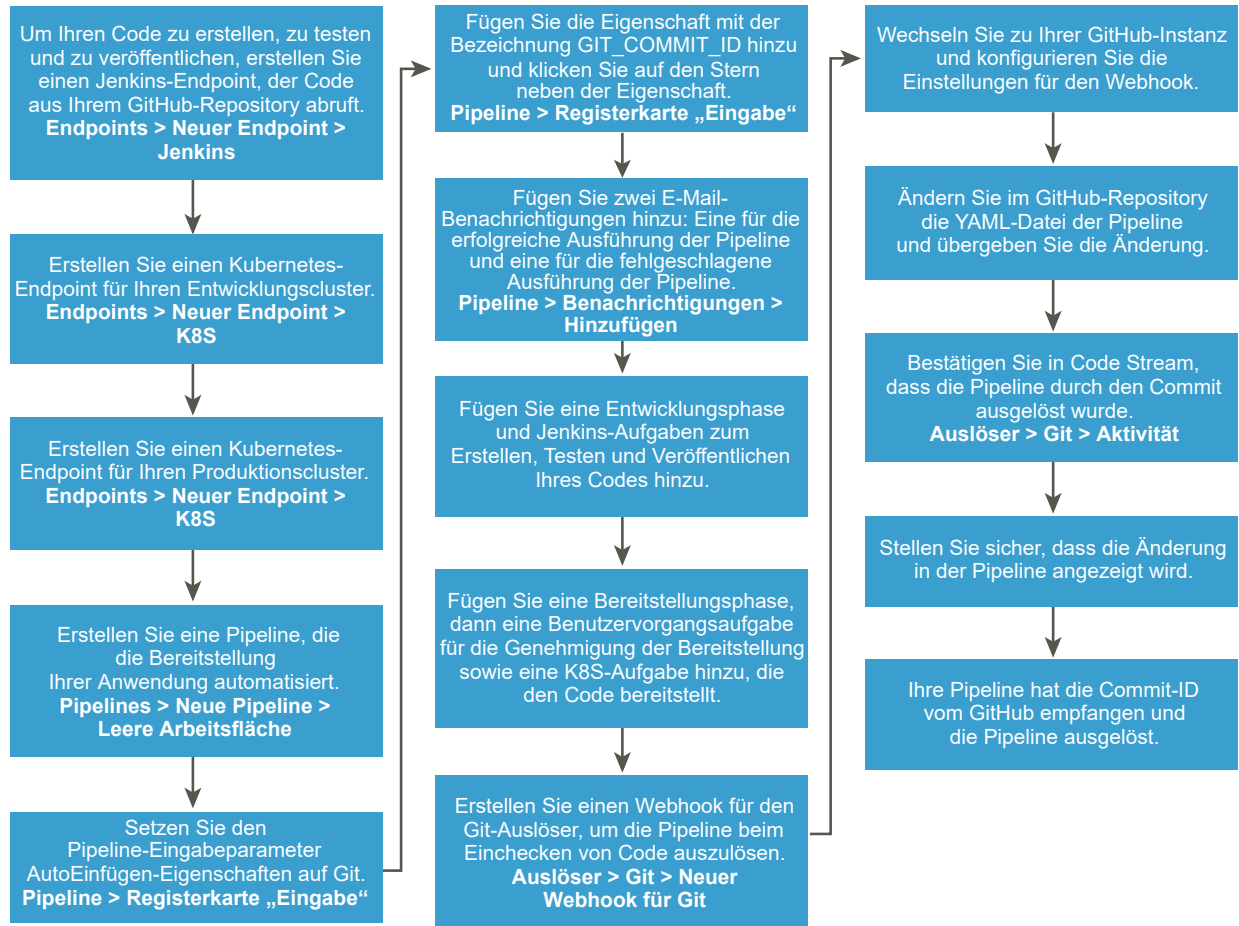
Als vRealize Automation Code Stream-Administrator oder Entwickler können Sie vRealize Automation Code Stream und VMware Tanzu Kubernetes Grid Integrated Edition (früher als VMware Enterprise PKS bezeichnet) verwenden, um die Bereitstellung Ihrer Softwareanwendungen in einem Kubernetes-Cluster zu automatisieren. In diesem Anwendungsfall werden weitere Methoden erläutert, mit denen Sie die Freigabe Ihrer Anwendung automatisieren können.

In diesem Anwendungsfall erstellen Sie eine Pipeline, die zwei Phasen umfasst und Jenkins zum Erstellen und Bereitstellen Ihrer Anwendung verwendet.

- Die erste Phase ist die Entwicklungsphase. Sie verwendet Jenkins, um den Code aus einem Branch im GitHub-Repository zu extrahieren und dann zu erstellen, zu testen und zu veröffentlichen.
- Die zweite Phase ist die Bereitstellungsphase. Während dieser Phase wird eine Benutzervorgangsaufgabe durchgeführt, die von Key-Usern genehmigt werden muss. Erst dann kann die Pipeline Ihre Anwendung auf dem Kubernetes-Cluster bereitstellen.



Die Entwicklungstools, Bereitstellungsinstanzen und die YAML-Datei der Pipeline müssen verfügbar sein, damit die Anwendung von der Pipeline erstellt, getestet, veröffentlicht und bereitgestellt werden kann. Die Pipeline stellt die Anwendung auf Entwicklungs- und Produktionsinstanzen der Kubernetes-Cluster auf AWS bereit.



Weitere Methoden zum Automatisieren der Freigabe Ihrer Anwendung:

- Statt Jenkins können Sie zum Erstellen Ihrer Anwendung die native Erstellungsfunktion von vRealize Automation Code Stream oder einen Docker-Build-Host verwenden.
- Statt Ihre Anwendung in einem Kubernetes-Cluster bereitzustellen, können Sie sie in einem Amazon Web Services-Cluster (AWS) bereitstellen.

Weitere Informationen zur Verwendung der nativen Erstellungsfunktion von vRealize Automation Code Stream und eines Docker-Hosts finden Sie unter:

- [Planen eines nativen CI/CD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage](#)
- [Planen eines nativen CI/CD-Builds in vRealize Automation Code Stream vor dem manuellen Hinzufügen von Aufgaben](#)

Voraussetzungen

- Stellen Sie sicher, dass sich der bereitzustellende Anwendungscode in einem funktionierenden GitHub-Repository befindet.
- Stellen Sie sicher, dass Sie über eine funktionierende Jenkins-Instanz verfügen.
- Stellen Sie sicher, dass Sie über einen funktionierenden E-Mail-Server verfügen.

- Erstellen Sie in vRealize Automation Code Stream einen E-Mail-Endpoint, der eine Verbindung mit Ihrem E-Mail-Server herstellt.
- Richten Sie zwei Kubernetes-Cluster für Entwicklung und Produktion auf Amazon Web Services (AWS) ein, auf denen die Pipeline Ihre Anwendung bereitstellt.
- Stellen Sie sicher, dass das GitHub-Repository den YAML-Code für die Pipeline und alternativ eine YAML-Datei enthält, in der die Metadaten und Spezifikationen für Ihre Umgebung definiert werden.

Verfahren

- 1 Klicken Sie in vRealize Automation Code Stream auf **Endpoints > Neuer Endpoint** und erstellen Sie einen Jenkins-Endpoint, den Sie in der Pipeline zum Extrahieren von Code aus dem GitHub-Repository verwenden.
- 2 Klicken Sie zum Erstellen von Kubernetes-Endpoints auf **Neuer Endpoint**.
 - a Erstellen Sie einen Endpoint für den Kubernetes-Entwicklungscluster.
 - b Erstellen Sie einen Endpoint für den Kubernetes-Produktionscluster.

Die URL Ihres Kubernetes-Clusters enthält unter Umständen eine Portnummer.

Beispiel:

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

```
https://api.kubernetesserver.fa2c1d78-9f00-4e30-8268-4ab81862080d.k8s-user.com
```
- 3 Erstellen Sie eine Pipeline, die einen Container Ihrer Anwendung, wie z. B. WordPress, im Kubernetes-Entwicklungscluster bereitstellt, und legen Sie die Eingabeeigenschaften für die Pipeline fest.
 - a Damit die Pipeline einen Code-Commit in GitHub erkennt, der die Pipeline auslöst, klicken Sie in der Pipeline auf die Registerkarte **Eingabe** und wählen Sie **AutoEinfügen-Eigenschaften** aus.
 - b Fügen Sie die Eigenschaft mit dem Namen **GIT_COMMIT_ID** hinzu und klicken Sie auf den Stern.

Bei Ausführung der Pipeline wird die Commit-ID angezeigt, die vom Git-Auslöser zurückgegeben wird.

The screenshot shows the Jenkins-K8s pipeline editor. The pipeline is titled "Jenkins-K8s" and is in an "Enabled" state. It consists of two main stages: "Dev" and "Deploy".

Dev Stage:

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

Deploy Stage:

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

The right-hand panel shows the "Pipeline Input Parameters" section. It includes a table of input parameters:

| Starred | Name | Value | Description |
|---------|-----------------------|-------|-------------|
| ☆ | GIT_BRANCH_NAME | | |
| ☆ | GIT_CHANGE_SUBJECT | | |
| ☆ | GIT_COMMIT_ID | | |
| ☆ | GIT_EVENT_DESCRIPTION | | |
| ☆ | GIT_EVENT_OWNER_NAME | | |
| ☆ | GIT_EVENT_TIMESTAMP | | |
| ☆ | GIT_REPO_NAME | | |
| ☆ | GIT_SERVER_URL | | |

At the bottom of the editor, there are buttons for "SAVE", "RUN", and "CLOSE", along with a status message "Last saved 5 days ago".

- 4 Fügen Sie Benachrichtigungen hinzu, um bei erfolgreicher oder fehlgeschlagener Ausführung der Pipeline eine E-Mail zu senden.
 - a Klicken Sie in der Pipeline auf die Registerkarte **Benachrichtigungen** und dann auf **Hinzufügen**.
 - b Um nach Abschluss der Pipeline-Ausführung eine E-Mail-Benachrichtigung hinzuzufügen, wählen Sie **E-Mail** und dann **Abschluss** aus. Wählen Sie anschließend den E-Mail-Server aus, geben Sie die E-Mail-Adressen ein und klicken Sie auf **Speichern**.
 - c Zum Hinzufügen einer weiteren E-Mail-Benachrichtigung bei einem Pipeline-Fehler wählen Sie **Fehler** aus und klicken Sie auf **Speichern**.

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

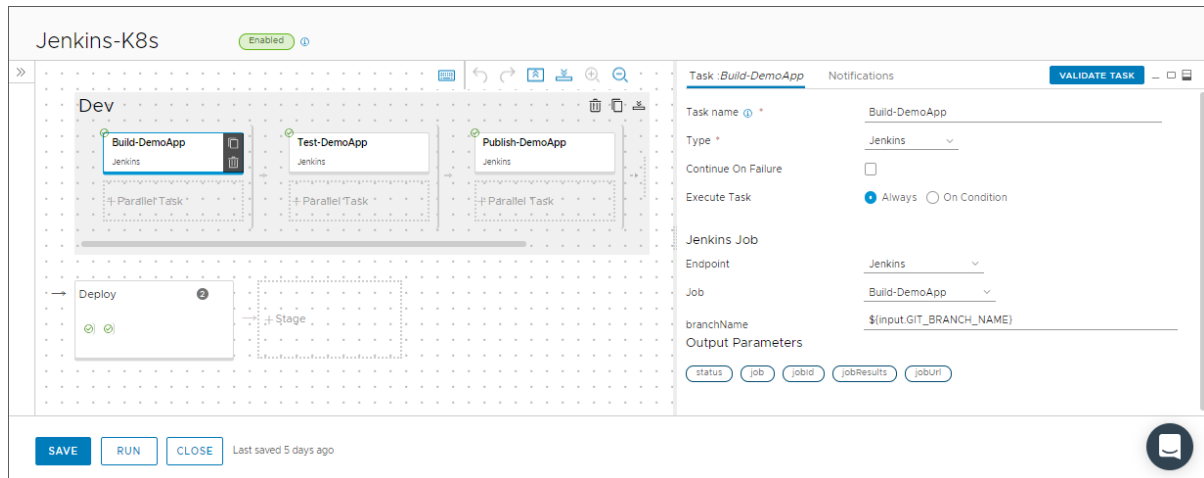
Subject \$ * Email Subject

Body ⓘ \$ *

1

CANCEL
SAVE

- 5 Fügen Sie Ihrer Pipeline eine Entwicklungsphase und Aufgaben zum Erstellen, Testen und Veröffentlichen der Anwendung hinzu. Überprüfen Sie anschließend jede Aufgabe.
 - a Fügen Sie zum Erstellen der Anwendung eine Jenkins-Aufgabe hinzu, die den Jenkins-Endpoint verwendet und einen Erstellungsauftrag über den Jenkins-Server ausführt. Damit die Pipeline den Code abrufen kann, geben Sie den Git-Branch in folgendem Format ein: `$ {Input. GIT_BRANCH_NAME}`
 - b Fügen Sie zum Testen der Anwendung eine Jenkins-Aufgabe hinzu, die denselben Jenkins-Endpoint verwendet und einen Testauftrag über den Jenkins-Server ausführt. Geben Sie anschließend denselben Git-Branch ein.
 - c Fügen Sie zum Veröffentlichen der Anwendung eine Jenkins-Aufgabe hinzu, die denselben Jenkins-Endpoint verwendet und einen Veröffentlichungsauftrag über den Jenkins-Server ausführt. Geben Sie anschließend denselben Git-Branch ein.



- 6 Fügen Sie der Pipeline eine Bereitstellungsphase und eine Aufgabe hinzu, die eine Genehmigung für die Bereitstellung Ihrer Anwendung erfordert, sowie eine weitere Aufgabe, die die Anwendung in Ihrem Kubernetes-Cluster bereitstellt. Überprüfen Sie anschließend jede Aufgabe.
 - a Fügen Sie zum Anfordern einer Genehmigung für die Bereitstellung Ihrer Anwendung eine Benutzervorgangsaufgabe sowie E-Mail-Adressen für den Benutzer hinzu, der die Bereitstellung genehmigen muss, und geben Sie eine Nachricht ein. Aktivieren Sie anschließend **E-Mail senden**.
 - b Fügen Sie zum Bereitstellen der Anwendung eine Kubernetes-Aufgabe hinzu. Wählen Sie in den Eigenschaften der Kubernetes-Aufgabe den Kubernetes-Entwicklungscluster, dann die Aktion **Erstellen** und anschließend die Nutzlastquelle **Lokale Definition** aus. Wählen Sie dann Ihre lokale YAML-Datei aus.

- 7 Fügen Sie einen Git-Webhook hinzu, der vRealize Automation Code Stream die Verwendung des Git-Auslösers ermöglicht. Mit diesem Auslöser wird die Pipeline ausgelöst, wenn Entwickler Ihren Code übergeben.

- 8 Navigieren Sie zum Testen der Pipeline zum GitHub-Repository, aktualisieren Sie die YAML-Datei der Anwendung und übernehmen Sie die Änderung.
- Stellen Sie in vRealize Automation Code Stream sicher, dass der Commit angezeigt wird.
 - Klicken Sie auf **Auslöser > Git > Aktivität**.
 - Suchen Sie nach dem Auslöser Ihrer Pipeline.
 - Klicken Sie auf **Dashboards > Pipeline-Dashboards**.
 - Suchen Sie im Pipeline-Dashboard im Bereich mit den letzten erfolgreichen Änderungen nach der GIT_COMMIT_ID.
- 9 Überprüfen Sie den Pipeline-Code und stellen Sie sicher, dass die Änderung angezeigt wird.

Ergebnisse

Herzlichen Glückwunsch! Sie haben die Bereitstellung Ihrer Softwareanwendung in Ihrem Kubernetes-Cluster automatisiert.

Beispiel: Beispiel für eine Pipeline-YAML, die eine Anwendung in einem Kubernetes-Cluster bereitstellt

Für den in diesem Beispiel verwendeten Pipeline-Typ ähnelt die YAML dem folgenden Code:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
    eyJzeWlwaG9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJlc2VybmFtZSI6InRhbmRvLWJldGEyIiwicGFzc3dvcmQI Oi
    JhRGstcmVOLWlUQil1IejciLCJlbWpCI6InRhbmRvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoizEdGdVoyOHRZbVYw
    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

```

Nächste Schritte

Zur Bereitstellung Ihrer Softwareanwendung im Kubernetes-Produktionscluster führen Sie die Schritte erneut aus und wählen Sie den Produktionscluster aus.

Weitere Informationen zur Integration von vRealize Automation Code Stream mit Jenkins finden Sie unter [Vorgehensweise zum Integrieren von vRealize Automation Code Stream in Jenkins](#).

Vorgehensweise zum Bereitstellen meiner Anwendung in vRealize Automation Code Stream für meine Blau/Grün-Bereitstellung

Blau/Grün ist ein Bereitstellungsmodell, das zwei Docker-Hosts verwendet, die Sie in einem Kubernetes-Cluster identisch bereitstellen und konfigurieren. Mit dem Blau/Grün-Bereitstellungsmodell reduzieren Sie die Ausfallzeiten, die in Ihrer Umgebung auftreten können, wenn Ihre Pipelines in vRealize Automation Code Stream Ihre Anwendungen bereitstellen.

Die Blau/Grün-Instanzen in Ihrem Bereitstellungsmodell dienen jeweils einem anderen Zweck. Immer nur eine Instanz akzeptiert den Live-Datenverkehr, der Ihre Anwendung bereitstellt, und jede Instanz akzeptiert diesen Datenverkehr zu bestimmten Zeiten. Die Blau-Instanz empfängt die erste Version Ihrer Anwendung, und die Grün-Instanz empfängt die zweite.

Der Lastausgleichsdienst in Ihrer Blau/Grün-Umgebung legt fest, welche Route der Live-Datenverkehr während der Bereitstellung Ihrer Anwendung nimmt. Durch die Verwendung des Blau/Grün-Modells bleibt Ihre Umgebung betriebsbereit, Benutzer bemerken keine Ausfallzeiten, und Ihre Pipeline integriert Ihre Anwendung kontinuierlich und stellt sie kontinuierlich in Ihrer Produktionsumgebung bereit.

Die Pipeline, die Sie in vRealize Automation Code Stream erstellen, stellt Ihr Blau/Grün-Bereitstellungsmodell in zwei Phasen dar. Eine Phase ist für die Entwicklung und die andere Phase ist für die Produktion.

Tabelle 5-2. Aufgaben in der Entwicklungsphase für die Blau/Grün-Bereitstellung

| Aufgabentyp | Aufgabe |
|-------------|--|
| Kubernetes | Erstellen Sie einen Namespace für Ihre Blau/Grün-Bereitstellung. |
| Kubernetes | Erstellen Sie einen geheimen Schlüssel für Docker-Hub. |
| Kubernetes | Erstellen Sie den Dienst, der für die Bereitstellung der Anwendung verwendet wird. |
| Kubernetes | Erstellen Sie die Blau-Bereitstellung. |
| Abfrage | Überprüfen Sie die Blau-Bereitstellung. |
| Kubernetes | Entfernen Sie den Namespace. |

Tabelle 5-3. Aufgaben in der Produktionsphase für die Blau/Grün-Bereitstellung

| Aufgabentyp | Aufgabe |
|-------------|---|
| Kubernetes | Grün ruft die Dienstdetails von Blau ab. |
| Kubernetes | Rufen Sie die Details für den Grün-Replikatsatz ab. |
| Kubernetes | Erstellen Sie die Grün-Bereitstellung und verwenden Sie den geheimen Schlüssel, um das Container-Image abzurufen. |
| Kubernetes | Aktualisieren Sie den Dienst. |
| Abfrage | Stellen Sie sicher, dass die Bereitstellung auf der Produktions-URL erfolgreich war. |
| Kubernetes | Beenden Sie die Blau-Bereitstellung. |
| Kubernetes | Entfernen Sie die Blau-Bereitstellung. |

Um Ihre Anwendung in Ihrem eigenen Blau/Grün-Bereitstellungsmodell bereitzustellen, erstellen Sie eine Pipeline in vRealize Automation Code Stream, die zwei Phasen umfasst. Die erste Phase umfasst die Blau-Aufgaben, die Ihre Anwendung für die Blau-Instanz bereitstellen. Die zweite Phase umfasst Grün-Aufgaben, die Ihre Anwendung für die Grün-Instanz bereitstellen.

Sie können Ihre Pipeline mithilfe der intelligenten Pipeline-Vorlage für CICD erstellen. Die Vorlage erstellt Ihre Pipeline-Phasen und -Aufgaben für Sie und enthält die Bereitstellungsauswahl.

Wenn Sie Ihre Pipeline manuell erstellen, müssen Sie die Pipeline-Phasen planen. Ein Beispiel finden Sie unter [Planen eines nativen CICD-Builds in vRealize Automation Code Stream vor dem manuellen Hinzufügen von Aufgaben](#).

In diesem Beispiel verwenden Sie die intelligente Pipeline-Vorlage für CICD, um Ihre Blau/Grün-Pipeline zu erstellen.

Voraussetzungen

- Vergewissern Sie sich, dass Sie auf einen funktionierenden Kubernetes-Cluster auf AWS zugreifen können.

- Vergewissern Sie sich, dass Sie eine Blau/Grün-Bereitstellungsumgebung eingerichtet haben und Ihre Blau/Grün-Instanzen so konfiguriert haben, dass sie identisch sind.
- Erstellen Sie einen Kubernetes-Endpoint in vRealize Automation Code Stream, der Ihr Anwendungs-Image im Kubernetes-Cluster auf AWS bereitstellt.
- Machen Sie sich mit der Verwendung der intelligenten Pipeline-Vorlage für CI/CD vertraut. Weitere Informationen hierzu finden Sie unter [Planen eines nativen CI/CD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage](#).

Verfahren

- 1 Klicken Sie auf **Pipelines > Neue Pipeline > Intelligente Vorlagen > CI-/CD-Vorlage**.
- 2 Geben Sie die Informationen für den CI-Abschnitt der intelligenten Pipeline-Vorlage für CI/CD ein und klicken Sie auf **Weiter**.

Weitere Hilfe finden Sie unter [Planen eines nativen CI/CD-Builds in vRealize Automation Code Stream vor der Verwendung der intelligenten Pipeline-Vorlage](#).

- 3 Abschließen des CD-Abschnitts der intelligenten Pipeline-Vorlage
 - a Wählen Sie die Umgebungen für Ihre Anwendungsbereitstellung aus. Beispiel: **Dev** und **Prod**.
 - b Wählen Sie den Dienst aus, den die Pipeline für die Bereitstellung verwenden soll.
 - c Wählen Sie im Bereitstellungsbereich den Cluster-Endpoint für die Dev- und die Prod-Umgebung aus.
 - d Wählen Sie für das Produktionsbereitstellungsmodell **Blau/Grün** aus und klicken Sie auf **Erstellen**.

Intelligente Vorlage: KI/KB

Schritt 2 von 2

Umgebung ⓘ * ☒ Entwicklung ☒ Produktion

Kubernetes-YAML-Datei... *

Verarbeitete Dateien: codestream.yaml

Dienst auswählen

| Name der Bereitstellung | Dienst | Namespace | Image |
|-------------------------|-----------------|-----------|-------|
| • codestream-demo | codestream-demo | bgreen1 | 1212 |

1 Dienste

Bereitstellung

| Umgebung | Cluster-Endpoint | Namespace |
|-------------|---|----------------|
| Entwicklung | 1030Endpoint-Kubernetes 騎家表術あA中在e臨停B道Ü8au*ñ | bgreen1-954081 |
| Produktion | 1030Endpoint-Kubernetes 騎家表術あA中在e臨停B道Ü8au*ñ | bgreen1 |

Bereitstellungsmodell * ☐ Canary ☐ Paralleles Upgrade ☒ Blau/Grün

Rollback ☐

Integritätsprüfungs-URL *

Ergebnisse

Herzlichen Glückwunsch! Sie haben die intelligente Pipeline-Vorlage verwendet, um eine Pipeline zu erstellen, die Ihre Anwendung für Ihre Blau/Grün-Instanzen in Ihrem Kubernetes-Produktionscluster auf AWS bereitstellt.

Beispiel: YAML-Beispielcode für einige Blau/Grün-Bereitstellungsaufgaben

Der YAML-Code, der in den Kubernetes-Pipeline-Aufgaben für Ihre Blau/Grün-Bereitstellung angezeigt wird, ähnelt möglicherweise den folgenden Beispielen. Nachdem die intelligente Pipeline-Vorlage Ihre Pipeline erstellt hat, können Sie die Aufgaben nach Bedarf für Ihre eigene Bereitstellung ändern.

YAML-Code zum Erstellen eines Beispiel-Namespace:

```
apiVersion: v1
kind: Namespace
metadata:
```

```
name: codestream-82855
namespace: codestream-82855
```

YAML-Code zum Erstellen eines Beispieldienstes:

```
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 zum Erstellen einer Beispielbereitstellung:

```
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
```

Nächste Schritte

Weitere Informationen zur Verwendung von vRealize Automation Code Stream finden Sie unter [Kapitel 5 Lernprogramme für die Verwendung von vRealize Automation Code Stream](#).

Informationen zum Rollback einer Bereitstellung finden Sie unter [Vorgehensweise zum Rollback meiner Bereitstellung in vRealize Automation Code Stream](#).

Weitere Referenzen finden Sie unter [Weitere Ressourcen für vRealize Automation Code Stream-Administratoren und -Entwickler](#).

Vorgehensweise zum Integrieren von eigenen Build-, Test- und Bereitstellungstools mit vRealize Automation Code Stream

Als DevOps-Administrator oder -Entwickler können Sie benutzerdefinierte Skripts erstellen, die die Funktion von vRealize Automation Code Stream erweitern. Mit Ihrem Skript können Sie vRealize Automation Code Stream in Ihre eigenen CI-Tools (Continuous Integration), CD-Tools (Continuous Delivery) und APIs integrieren, die Ihre Anwendungen erstellen, testen und bereitstellen. Benutzerdefinierte Skripts sind besonders nützlich, wenn Sie Ihre Anwendungs-APIs nicht öffentlich verfügbar machen.

Ihr benutzerdefiniertes Skript kann fast alles tun, was Sie für die Integration in Ihre Build-, Test- und Bereitstellungstools benötigen. Beispielsweise kann es den Arbeitsbereich in Ihrer Pipeline dazu nutzen, CI-Aufgaben zum Erstellen und Testen Ihrer Anwendung und CD-Aufgaben zum Bereitstellen der Anwendung zu unterstützen. Es kann eine Meldung an Slack senden, wenn eine Pipeline abgeschlossen ist, und vieles mehr.

Sie schreiben Ihr benutzerdefiniertes Skript in einer der unterstützten Sprachen. Sie nehmen Ihre Geschäftslogik im Skript auf und definieren Ein- und Ausgaben. Ausgabetypen können Zahl, Zeichenfolge, Text und Kennwort enthalten. Sie können mehrere Versionen eines benutzerdefinierten Skripts mit unterschiedlicher Geschäftslogik, Eingaben und Ausgaben erstellen.

In der Pipeline wurde eine Version des Skripts in einer benutzerdefinierten Aufgabe ausgeführt. Die von Ihnen erstellten Skripts befinden sich in Ihrer vRealize Automation Code Stream-Instanz.

Wenn eine Pipeline eine benutzerdefinierte Integration verwendet und Sie versuchen, die benutzerdefinierte Integration zu löschen, wird eine Fehlermeldung angezeigt, die besagt, dass Sie sie nicht löschen können.

Beim Löschen einer benutzerdefinierten Integration werden alle Versionen des benutzerdefinierten Skripts entfernt. Wenn Sie über eine vorhandene Pipeline mit einer benutzerdefinierten Aufgabe verfügen, die eine beliebige Version des Skripts verwendet, schlägt diese Pipeline fehl. Damit vorhandene Pipelines nicht fehlschlagen, können Sie die nicht mehr zu verwendende Version des Skripts als veraltet einstufen und zurückziehen. Wenn diese Version von keiner Pipeline verwendet wird, können Sie sie löschen.

Tabelle 5-4. Vorgehensweisen nach dem Schreiben des benutzerdefinierten Skripts

| Vorgehensweise... | Weitere Informationen zu dieser Aktion... |
|---|--|
| Hinzufügen einer benutzerdefinierten Aufgabe zu Ihrer Pipeline. | <p>Die benutzerdefinierte Aufgabe:</p> <ul style="list-style-type: none"> ■ Wird auf demselben Container wie andere CI-Aufgaben in Ihrer Pipeline ausgeführt. ■ Enthält Eingabe- und Ausgabevariablen, die Ihr Skript auffüllt, bevor die Pipeline die benutzerdefinierte Aufgabe ausführt. ■ Unterstützt mehrere Datentypen und verschiedene Arten von Metadaten, die Sie als Eingaben oder Ausgaben in Ihrem Skript definieren. |
| Auswählen Ihres Skripts in der benutzerdefinierten Aufgabe. | Sie deklarieren die Eingabe- und Ausgabeigenschaften im Skript. |
| Speichern, Aktivieren und Ausführen Ihrer Pipeline. | Bei Ausführung der Pipeline ruft die benutzerdefinierte Aufgabe die Version des angegebenen Skripts auf und führt die darin enthaltene Geschäftslogik aus, die von Ihrem Build-, Test- und Bereitstellungstool in vRealize Automation Code Stream integriert wird. |
| Anzeigen der Ausführungen nach dem Ausführen Ihrer Pipeline. | Vergewissern Sie sich, dass die Pipeline die erwarteten Ergebnisse bereitgestellt hat. |

Wenn Sie eine benutzerdefinierte Aufgabe verwenden, die eine Version der benutzerdefinierten Integration aufruft, können Sie benutzerdefinierte Umgebungsvariablen als Name/Wert-Paare auf der Registerkarte **Arbeitsbereich** der Pipeline einbeziehen. Wenn das Builder-Image den Arbeitsbereichscontainer erstellt, der die CI-Aufgabe ausführt und Ihr Image bereitstellt, übergibt vRealize Automation Code Stream die Umgebungsvariablen an diesen Container.

Beispiel: Wenn Ihre vRealize Automation Code Stream-Instanz einen Web-Proxy benötigt und Sie einen Docker-Host zum Erstellen eines Containers für eine benutzerdefinierte Integration verwenden, führt vRealize Automation Code Stream die Pipeline aus und übergibt die Variablen der Web-Proxy-Einstellungen an diesen Container.

Tabelle 5-5. Beispiel für Name/Wert-Paare der Umgebungsvariable

| Name | Wert |
|-------------|--|
| HTTPS_PROXY | http://10.0.0.255:1234 |
| https_proxy | http://10.0.0.255:1234 |
| NO_PROXY | 10.0.0.32, *.dept.vsphere.local |
| no_proxy | 10.0.0.32, *.dept.vsphere.local |
| HTTP_PROXY | http://10.0.0.254:1234 |
| http_proxy | http://10.0.0.254:1234 |
| PATH | /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin |

Name/Wert-Paare werden auf der Benutzeroberfläche folgendermaßen angezeigt:



In diesem Beispiel wird eine benutzerdefinierte Integration erstellt, die vRealize Automation Code Stream mit Ihrer Slack-Instanz verbindet und eine Nachricht an einen Slack-Kanal sendet.

Voraussetzungen

- Um Ihr benutzerdefiniertes Skript zu schreiben, stellen Sie sicher, dass Sie über eine der folgenden Sprachen verfügen: Python 2, Python 3, Node.js oder eine der folgenden Shell-Sprachen: Bash, sh oder zsh.
- Generieren Sie mithilfe der installierten Node.js- oder Python-Laufzeit ein Container-Image.

Verfahren

1 Erstellen Sie die benutzerdefinierte Integration.

- a Klicken Sie auf **Benutzerdefinierte Integrationen > Neu** und geben Sie einen entsprechenden Namen ein.
- b Wählen Sie die bevorzugte Laufzeitumgebung aus.
- c Klicken Sie auf **Erstellen**.

Ihr Skript wird geöffnet und zeigt den Code an, der die erforderliche Laufzeitumgebung enthält. Beispiel: `runtime: "nodejs"`. Das Skript muss die Laufzeit enthalten, die vom Builder-Image verwendet wird, sodass die benutzerdefinierte Aufgabe, die Sie zu Ihrer Pipeline hinzufügen, bei Ausführung der Pipeline erfolgreich ausgeführt wird. Andernfalls schlägt die benutzerdefinierte Aufgabe fehl.

Die YAML-Hauptbereiche Ihrer benutzerdefinierten Integration umfassen die Laufzeiteigenschaften, Code sowie Eingabe- und Ausgabeeigenschaften. In diesem Verfahren werden verschiedene Typen und die Syntax erläutert.

| YAML-Schlüssel der benutzerdefinierten Integration | Beschreibung |
|--|--|
| runtime | <p>Laufzeitumgebung der Aufgabe, in der vRealize Automation Code Stream den Code ausführt, der eine der folgenden Zeichenfolgen ohne Unterscheidung der Groß-/Kleinschreibung darstellen kann:</p> <ul style="list-style-type: none"> ■ nodejs ■ python2 ■ python3 ■ shell <p>Wenn kein Wert vorhanden ist, wird „Shell“ als Standardeinstellung übernommen.</p> |
| code | Benutzerdefinierte Geschäftslogik, die als Teil der benutzerdefinierten Aufgabe durchgeführt wird. |

| YAML-Schlüssel der benutzerdefinierten Integration | |
|--|--|
| Integration | Beschreibung |
| inputProperties | Array der Eingabeeigenschaften, die im Rahmen der Konfiguration der benutzerdefinierten Aufgabe erfasst werden. Diese Eigenschaften werden in der Regel im Code verwendet. |
| outputProperties | Array der Ausgabeeigenschaften, die aus der benutzerdefinierten Aufgabe exportiert und an die Pipeline weitergegeben werden können. |

- 2 Deklarieren Sie die Eingabeeigenschaften in Ihrem Skript unter Verwendung der verfügbaren Datentypen und Metadaten.

Die Eingabeeigenschaften werden als Kontext an Ihr Skript im Abschnitt `code:` der YAML übergeben.

| YAML-Eingabeschlüssel der benutzerdefinierten Aufgabe | | |
|---|--|--------------|
| | Beschreibung | Erforderlich |
| type | Anzuzeigende Eingabetypen: <ul style="list-style-type: none"> ■ text ■ textarea ■ number ■ checkbox ■ password ■ select | Ja |
| name | Name oder Zeichenfolge der Eingabe für die benutzerdefinierte Aufgabe, die in den YAML-Code der benutzerdefinierten Integration eingefügt wird. Muss für jede Eingabeeigenschaft, die für eine benutzerdefinierte Integration definiert wird, eindeutig sein. | Ja |
| title | Bezeichnung der Textzeichenfolge der Eingabeeigenschaft für die benutzerdefinierte Aufgabe auf der Arbeitsfläche des Pipeline-Modells. Wenn kein Wert angegeben ist, wird standardmäßig name verwendet. | Nein |
| required | Gibt an, ob ein Benutzer die Eingabeeigenschaft beim Konfigurieren der benutzerdefinierten Aufgabe eingeben muss. Legen Sie diesen Wert auf „true“ oder „false“ fest. Bei einem Wert von „true“ verbleibt der Status der Aufgabe bei „Nicht konfiguriert“, wenn ein Benutzer beim Konfigurieren der benutzerdefinierten Aufgabe auf der Pipeline-Arbeitsfläche keinen Wert bereitstellt. | Nein |
| placeholder | Standardtext für den Eingabebereich der Eingabeeigenschaft, wenn kein Wert vorhanden ist. Ist dem HTML-Platzhalterattribut zugeordnet. Wird nur für bestimmte Typen von Eingabeeigenschaften unterstützt. | Nein |
| defaultValue | Standardwert, mit dem der Eingabebereich der Eingabeeigenschaft befüllt wird, wenn die benutzerdefinierte Aufgabe auf der Seite des Pipeline-Modells ausgeführt wird. | Nein |

| YAML-Eingabeschlüssel der benutzerdefinierten Aufgabe | Beschreibung | Erforderlich |
|---|---|--------------|
| bindable | Legt fest, ob die Eingabeeigenschaft Dollarzeichenvariablen akzeptiert, wenn die benutzerdefinierte Aufgabe auf der Arbeitsfläche der Pipeline modelliert wird. Fügt den Indikator \$ neben dem Titel ein. Wird nur für bestimmte Typen von Eingabeeigenschaften unterstützt. | Nein |
| labelMessage | Zeichenfolge, die als QuickInfo für Benutzer fungiert. Fügt das QuickInfo-Symbol i neben dem Eingabetitel ein. | Nein |
| enum | <p>Nimmt ein Array von Werten auf, in dem die Optionen zum Auswählen von Eingabeeigenschaften angezeigt werden. Wird nur für bestimmte Typen von Eingabeeigenschaften unterstützt.</p> <p>Wenn ein Benutzer eine Option auswählt und für die benutzerdefinierte Aufgabe speichert, entspricht der Wert für inputProperty diesem Wert und wird in der benutzerdefinierten Aufgabe angezeigt.</p> <p>Beispielsweise der Wert 2015.</p> <ul style="list-style-type: none"> ■ 2015 ■ 2016 ■ 2017 ■ 2018 ■ 2019 ■ 2020 | Nein |
| options | <p>Nimmt ein Array von Objekten mithilfe von optionKey und optionValue auf.</p> <ul style="list-style-type: none"> ■ optionKey. Der Wert wird an den Codeabschnitt der Aufgabe weitergegeben. ■ optionValue. Zeichenfolge, die die Option auf der Benutzeroberfläche anzeigt. <p>Wird nur für bestimmte Typen von Eingabeeigenschaften unterstützt.</p> <p>Optionen:</p> <p>optionKey: key1. Wenn dieser Wert für die benutzerdefinierte Aufgabe ausgewählt und gespeichert wurde, entspricht der Wert dieser inputProperty dem Wert key1 im Codeabschnitt.</p> <p>optionValue: 'Bezeichnung für 1'. Anzeigewert für key1 auf der Benutzeroberfläche, der an keiner anderen Stelle für die benutzerdefinierte Aufgabe angezeigt wird.</p> <p>optionKey: key2</p> <p>optionValue: 'Bezeichnung für 2'</p> <p>optionKey: key3</p> <p>optionValue: 'Bezeichnung für 3'</p> | Nein |

| YAML-Eingabeschlüssel der benutzerdefinierten Aufgabe | | |
|---|--|--------------|
| | Beschreibung | Erforderlich |
| minimum | Nimmt eine Zahl auf, die als Mindestwert fungiert und für diese Eingabeeigenschaft gültig ist. Wird nur für Eingabeeigenschaften vom Typ „Zahl“ unterstützt. | Nein |
| maximum | Nimmt eine Zahl auf, die als Höchstwert fungiert und für diese Eingabeeigenschaft gültig ist. Wird nur für Eingabeeigenschaften vom Typ „Zahl“ unterstützt. | Nein |

Tabelle 5-6. Unterstützte Datentypen und Metadaten für benutzerdefinierte Skripts

| Unterstützte Datentypen | Unterstützte Metadaten für die Eingabe |
|---|--|
| <ul style="list-style-type: none"> ■ Zeichenfolge ■ Text ■ Liste: als Liste eines beliebigen Typs ■ Zuordnung: als map[string]any ■ Sicher: wird als Kennwort-Textfeld dargestellt und verschlüsselt, wenn Sie die benutzerdefinierte Aufgabe speichern ■ Zahl ■ Boolesch: wird in Form von Textfeldern angezeigt ■ URL: identisch mit Zeichenfolge, mit zusätzlicher Validierung ■ Auswahl, Optionsschaltfläche | <ul style="list-style-type: none"> ■ type: einer von „string“ oder „text“... ■ default: Standardwert ■ options: Liste mit oder Zuordnung von Optionen, die mit der Auswahl oder der Optionsschaltfläche verwendet werden sollen ■ min: Minimalwert oder -größe ■ max: Maximalwert oder -größe ■ title: detaillierter Name des Textfeldes ■ placeholder: UI-Platzhalter ■ description: wird zur QuickInfo |

Beispiel:

```
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 Deklarieren Sie die Ausgabeeigenschaften in Ihrem Skript.

Das Skript erfasst Ausgabeeigenschaften aus dem Abschnitt `code:` der Geschäftslogik Ihres Skripts, in dem Sie den Kontext für die Ausgabe deklarieren.

Wenn die Pipeline ausgeführt wird, können Sie den Antwortcode für die Aufgabenausgabe eingeben. Beispiel: **200**.

Schlüssel, die von vRealize Automation Code Stream für jede **outputProperty** unterstützt werden.

| key | Beschreibung |
|-------|---|
| Typ | Enthält zurzeit einen einzelnen Wert vom Typ label . |
| name | Schlüssel, der vom Codeblock der benutzerdefinierten Integrations-YAML ausgegeben wird. |
| title | Bezeichnung auf der Benutzeroberfläche, die outputProperty anzeigt. |

Beispiel:

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

- 4 Um mit der Eingabe und Ausgabe Ihres benutzerdefinierten Skripts zu interagieren, rufen Sie eine Eingabeeigenschaft ab oder legen Sie eine Ausgabeeigenschaft unter Verwendung von **context** fest.

Für eine Eingabeeigenschaft: `(context.getInput("key"))`

Für eine Ausgabeeigenschaft: `(context.setOutput("key", "value"))`

Für Node.js:

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

Für Python:

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

Für Shell:

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

- 5 Deklarieren Sie im Abschnitt `code`: die gesamte Geschäftslogik für Ihre benutzerdefinierte Integration.

Beispielsweise mit der Node.js-Laufzeitumgebung

```
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 Bevor Sie Ihr benutzerdefiniertes Integrationsskript mit einer Versionsangabe versehen und freigeben, laden Sie die Kontextdatei für Python oder Node.js herunter. Testen Sie dann die Geschäftslogik, die Sie in Ihr Skript aufgenommen haben.
 - a Positionieren Sie den Mauszeiger im Skript und klicken Sie danach oben auf der Arbeitsfläche auf die Schaltfläche für die Kontextdatei. Wenn Ihr Skript beispielsweise in Python programmiert ist, klicken Sie auf **CONTEXT.PY**.
 - b Ändern Sie die Datei und speichern Sie sie.
 - c Führen Sie auf Ihrem Entwicklungssystem Ihr benutzerdefiniertes Skript mithilfe der Kontextdatei aus und testen Sie es.
- 7 Wenden Sie eine Version auf Ihr benutzerdefiniertes Integrationsskript an.
 - a Klicken Sie auf **Version**.
 - b Geben Sie die Versionsinformationen ein.

- c Klicken Sie auf **Version**, damit Sie das Skript in Ihrer benutzerdefinierten Aufgabe auswählen können.
- d Um die Version zu erstellen, klicken Sie auf **Erstellen**.

Version wird erstellt

| | |
|-------------------|--|
| Version * | 1.0 |
| Beschreibung | <input type="text" value="New"/> |
| Protokoll ändern | <input type="text" value="New for 1.0"/> |
| Version freigeben | <input checked="" type="checkbox"/> |

ABBRECHEN
ERSTELLEN

- 8 Zum Speichern des Skripts klicken Sie auf **Speichern**.
- 9 Konfigurieren Sie den Arbeitsbereich in Ihrer Pipeline.
 - a Klicken Sie auf die Registerkarte **Arbeitsumgebung**.
 - b Wählen Sie den Docker-Host und die Builder-Image-URL aus.

Demo-customTask-nodejs
Aktiviert

Arbeitsbereich
Eingabe
Modell
Ausgabe

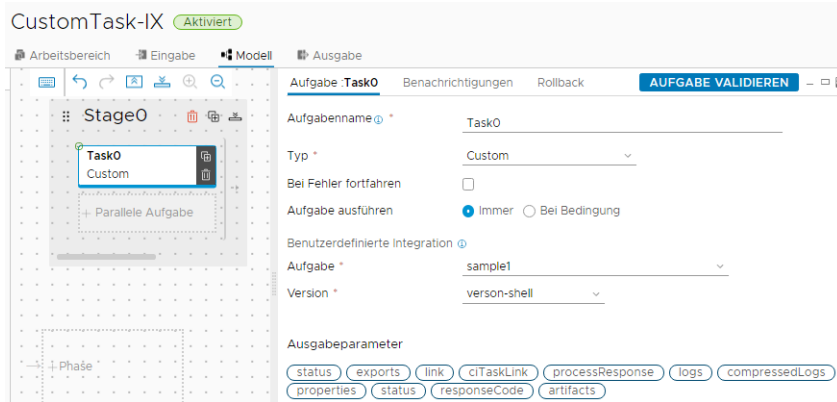
Stellen Sie Details zu Container und Host bereit, um fortlaufende Integrationsaufgaben auszuführen.

| | |
|--------------------------|---|
| Host * | <input type="text" value="Docker-saas"/> |
| Image-URL des Build... * | <input type="text" value="node:lasest"/> |
| Image-Registrierung | <input type="text" value="--Endpoint der Image-Registrierung auswählen--"/> |
| Arbeitsverzeichnis | <input type="text"/> |
| Cache | <input type="text"/> |
| CPU-Grenzwert * | <input type="text"/> |
| Arbeitsspeichergren... | <input type="text"/> |
| Git-Klon | <input type="checkbox"/> |

i Wenn diese Pipeline mit Git über einen Webhook verknüpft ist, wird die Pipeline bei Git-Ereignissen ausgelöst. Für CI-Aufgaben wird das verknüpfte Git-Repository (Details aus den Git-Webhook-Parametern) automatisch in den Arbeitsbereich geklont.

- 10 Fügen Sie Ihrer Pipeline eine benutzerdefinierte Aufgabe hinzu und konfigurieren Sie sie.
 - a Klicken Sie auf die Registerkarte **Modell**.
 - b Fügen Sie eine Aufgabe hinzu, wählen Sie als Typ **Benutzerdefiniert** aus und geben Sie einen entsprechenden Namen ein.

- c Wählen Sie Ihr benutzerdefiniertes Integrationsskript und die Version aus.
- d Um eine benutzerdefinierte Meldung in Slack anzuzeigen, geben Sie den Meldungstext ein.
Jeder eingegebene Text überschreibt den `defaultValue` in Ihrem benutzerdefinierten Integrationsskript. Beispiel:



- 11 Speichern und aktivieren Sie Ihre Pipeline.
 - a Klicken Sie auf **Speichern**.
 - b Klicken Sie auf der Registerkarte „Pipeline“ auf **Pipeline aktivieren**, damit der Kreis nach rechts verschoben wird.
- 12 Führen Sie Ihre Pipeline aus.
 - a Klicken Sie auf **Ausführen**.
 - b Sehen Sie sich die Pipeline-Ausführung an.

- c Bestätigen Sie, dass die Ausgabe den erwarteten Statuscode, den Antwortcode, den Status und die deklarierte Ausgabe enthält.

Sie haben **statusCode** als Ausgabeeigenschaft definiert. Beispiel: Ein **statusCode** von 200 deutet möglicherweise auf einen erfolgreichen Slack-Beitrag hin, und ein **responseCode** von 0 gibt möglicherweise an, dass das Skript ohne Fehler erfolgreich war.

- d Um die Ausgabe in den Ausführungsprotokollen zu bestätigen, klicken Sie auf **Ausführungen**, klicken Sie auf den Link zu Ihrer Pipeline, klicken Sie auf die Aufgabe und sehen Sie sich die protokollierten Daten an. Beispiel:

The screenshot displays the vRealize Automation interface for a pipeline named 'custom-int-demo #5'. The pipeline status is 'COMPLETED' with 0 errors. The execution progress shows 'Stage0' completed, with 'Task0' and 'Task1' also completed. The details for 'Task1' are as follows:

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

[View Full Log](#)

13 Wenn ein Fehler auftritt, beheben Sie das Problem und führen Sie die Pipeline erneut aus.

Wenn beispielsweise eine Datei oder ein Modul im Basis-Image fehlt, müssen Sie ein weiteres Basis-Image erstellen, das die fehlende Datei enthält. Geben Sie dann die Docker-Datei an und übertragen Sie das Image über die Pipeline.

Ergebnisse

Herzlichen Glückwunsch! Sie haben ein benutzerdefiniertes Integrationsskript erstellt, das vRealize Automation Code Stream mit Ihrer Slack-Instanz verbindet und eine Nachricht an einen Slack-Kanal sendet.

Nächste Schritte

Erstellen Sie weiterhin benutzerdefinierte Integrationen, um die Verwendung benutzerdefinierter Aufgaben in Ihren Pipelines zu unterstützen, sodass Sie die Funktion von vRealize Automation Code Stream in der Automatisierung des Lebenszyklus Ihrer Softwareversion erweitern können.

Vorgehensweise zum Verwenden der Ressourceneigenschaften einer Cloud-Vorlagenaufgabe in der nächsten Aufgabe

Bei Verwendung einer Cloud-Vorlagenaufgabe in vRealize Automation Code Stream stellt sich die allgemeine Frage, wie die Ausgabe dieser Aufgabe in einer Folgeaufgabe in Ihrer Pipeline verwendet werden soll. Zur Verwendung der Ausgabe einer Cloud-Vorlagenaufgabe, wie z. B. einer Cloud-Maschine, müssen Sie wissen, wie Sie in den Bereitstellungsdetails der Cloud-Vorlagenaufgabe und der IP-Adresse der Cloud-Maschine nach den Ressourceneigenschaften suchen können.

Beispielsweise enthalten die Bereitstellungsdetails einer VMware Cloud-Vorlage die Cloud-Maschinenressource und die zugehörige IP-Adresse. Sie können die Cloud-Maschine und IP-Adresse in Ihrer Pipeline als Variable verwenden, um eine Cloud-Vorlagenaufgabe an eine REST-Aufgabe zu binden.

Die zum Auffinden der IP-Adresse für die Cloud-Maschine verwendete Methode ist nicht typisch, da die Bereitstellung der VMware Cloud-Vorlage abgeschlossen werden muss, bevor die Bereitstellungsdetails verfügbar sind. Anschließend können Sie die Ressourcen aus der VMware Cloud-Vorlagenbereitstellung verwenden, um Ihre Pipeline-Aufgabe zu binden.

- Die in einer Cloud-Vorlagenaufgabe in Ihrer Pipeline angezeigten Ressourceneigenschaften werden in der VMware Cloud-Vorlage in vRealize Automation Cloud Assembly definiert.
- Sie wissen unter Umständen nicht, wann eine Bereitstellung dieser Cloud-Vorlage abgeschlossen wurde.
- Die Ausgabeeigenschaften der VMware Cloud-Vorlage können in einer Cloud-Vorlagenaufgabe in vRealize Automation Code Stream erst angezeigt werden, wenn die Bereitstellung abgeschlossen ist.

Dieses Beispiel kann besonders nützlich sein, wenn Sie eine Anwendung bereitstellen und verschiedene APIs aufrufen. Wenn Sie beispielsweise eine Cloud-Vorlagenaufgabe zum Aufrufen einer VMware Cloud-Vorlage verwenden, die eine Wordpress-Anwendung mit einer REST API bereitstellt, können Sie nach der IP-Adresse der bereitgestellten Maschine in den Bereitstellungsdetails suchen und die API zu Testzwecken verwenden.

Die Cloud-Vorlagenaufgabe unterstützt Sie bei der Verwendung von Variablenbindungen, indem der Typ vor den Details zur automatischen Befüllung angezeigt wird. Sie bestimmen, wie die Variable gebunden wird.

Die entsprechende Vorgehensweise finden Sie in diesem Beispiel:

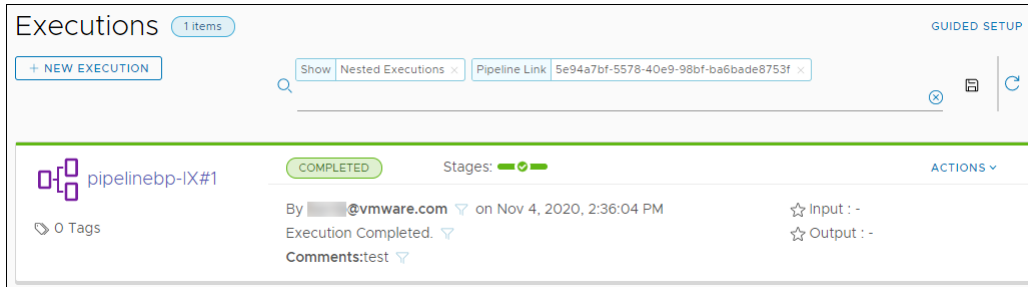
- Suchen Sie nach den Bereitstellungsdetails und Ressourceneigenschaften für Ihre Cloud-Vorlagenaufgabe in einer Pipeline, die erfolgreich ausgeführt wurde.
- Suchen Sie nach der IP-Adresse der Cloud-Maschine im Abschnitt „Ressourcen“ der Bereitstellungsdetails.
- Fügen Sie eine REST-Aufgabe im Anschluss an die Cloud-Vorlagenaufgabe in Ihrer Pipeline hinzu.
- Binden Sie die Cloud-Vorlagenaufgabe an die REST-Aufgabe, indem Sie die IP-Adresse der Cloud-Maschine in der URL der REST-Aufgabe verwenden.
- Führen Sie Ihre Pipeline aus und beobachten Sie den Bindungsvorgang zwischen der Cloud-Vorlagenaufgabe und der REST-Aufgabe.

Voraussetzungen

- Stellen Sie sicher, dass Sie über eine funktionierende VMware Cloud-Vorlage verfügen.
- Stellen Sie sicher, dass die Bereitstellung der VMware Cloud-Vorlage in vRealize Automation Cloud Assembly erfolgreich verlaufen ist.
- Stellen Sie sicher, dass Sie über eine Pipeline mit einer Cloud-Vorlagenaufgabe verfügen, die diese VMware Cloud-Vorlage verwendet.
- Stellen Sie sicher, dass Ihre Pipeline erfolgreich ausgeführt wurde.

Verfahren


- 1 Suchen Sie in Ihrer Pipeline nach der IP-Adresse der Cloud-Maschine im Abschnitt „Ressourcen“ der Bereitstellungsdetails der Cloud-Vorlagenaufgabe.
 - a Klicken Sie auf **Aktionen > Ausführungen anzeigen**.
 - b Klicken Sie in einer erfolgreich ausgeführten Pipeline auf den Link neben der Pipeline-Ausführung.



- c Klicken Sie unter dem Namen der Pipeline auf den Link zur **Aufgabe**.



- d Suchen Sie im Bereich „Ausgabe“ nach den Bereitstellungsdetails.


pipelinebp-IX #1
COMPLETED
ACTIONS ▾

✓ Stage0
✓ Task0

| | |
|---------------------|--|
| Task name | Task0 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-b6c0-3bf3638a71bc",
7   "blueprintId": "8d1dd801-3a32-4f3b-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".

```

Input

Action
Create Deployment

Cloud template
bhawesh

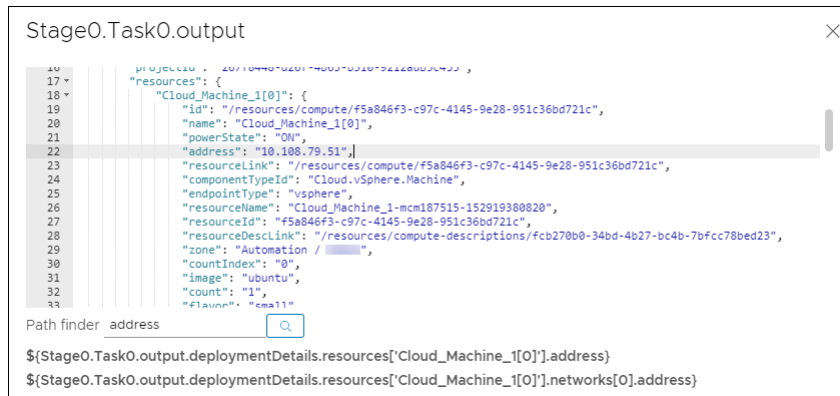
Cloud template version
4

- e Suchen Sie im Abschnitt „Ressourcen“ der Bereitstellungsdetails nach dem Namen der Cloud-Maschine.

Sie beziehen die Syntax für den Namen der Cloud-Maschine in die URL Ihrer REST-Aufgabe ein.

- f Klicken Sie zum Auffinden des Bindungsausdrucks für die Ausgabeeigenschaft der Cloud-Vorlagenaufgabe auf **JSON-AUSGABE ANZEIGEN** und suchen Sie nach der Adresseigenschaft und der IP-Adresse der Cloud-Maschine.

Der Bindungsausdruck wird unterhalb der Eigenschaft und dem Suchsymbol in der JSON-Ausgabe angezeigt.



Die Eigenschaft „Adressressource“ wird in der IP-Adresse der Cloud-Maschine angezeigt. Beispiel:

```

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

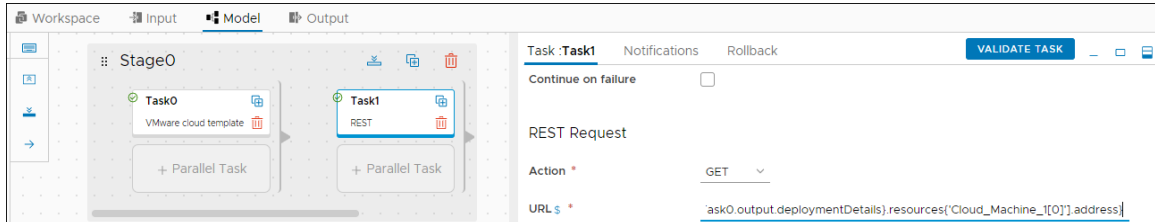
```

- 2 Kehren Sie zu Ihrem Pipeline-Modell zurück und geben Sie die URL in der REST-Aufgabe ein.
 - a Klicken Sie auf **Aktionen > Pipeline anzeigen**.
 - b Klicken Sie auf die REST-Aufgabe.

- c Geben Sie `$` im Bereich „REST-Anforderungs-URL“ ein, wählen Sie **Phase**, **Aufgabe**, **Ausgabe**, **Bereitstellungsdetails** aus. Geben Sie anschließend **Ressourcen** ein.

Automatische Vervollständigung kann solange verwendet werden, bis **Ressourcen** eingegeben werden müssen.

- d Geben Sie die restlichen Ressourcen der Cloud-Maschine aus den Bereitstellungsdetails als `{ 'Cloud_Machine_1[0]' }.address` ein.



Für den Cloud-Maschineneintrag müssen Sie die angezeigte Notation mit eckigen Klammern verwenden.

Das vollständige URL-Format lautet: `$`

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

- 3 Führen Sie Ihre Pipeline aus und beobachten Sie, wie die REST-Aufgabe die Cloud-Maschine und IP-Adresse aus der Ausgabe Ihrer Cloud-Vorlagenaufgabe als URL zum Testen verwendet.

Ergebnisse

Herzlichen Glückwunsch! Sie haben den Namen und die IP-Adresse der Cloud-Maschine in den Bereitstellungsdetails und der JSON-Ausgabe einer Cloud-Vorlagenaufgabe gefunden und sie verwendet, um die Ausgabe der Cloud-Vorlagenaufgabe an die Eingabe der REST-Aufgaben-URL in der Pipeline zu binden.

Nächste Schritte

Erkunden Sie weiterhin die Verwendung von Bindungsvariablen aus Ressourcen in der Cloud-Vorlagenaufgabe mit anderen Aufgaben in der Pipeline.

Vorgehensweise zum Verwenden einer REST API für die Integration von vRealize Automation Code Stream in andere Anwendungen

vRealize Automation Code Stream bietet ein REST-Plug-In, mit dem Sie vRealize Automation Code Stream in andere Anwendungen integrieren können, die eine REST API verwenden, sodass Sie Softwareanwendungen, die miteinander interagieren müssen, kontinuierlich entwickeln und bereitstellen können. Das REST-Plug-In ruft eine API auf, die Informationen zwischen vRealize Automation Code Stream und einer anderen Anwendung sendet und empfängt.

Mit dem REST-Plug-In können Sie:

- Externe REST-API-basierte Systeme in eine vRealize Automation Code Stream-Pipeline integrieren.
- Eine vRealize Automation Code Stream-Pipeline als Teil des Ablaufs von externen Systemen integrieren.

Das REST-Plug-In funktioniert mit jeder REST API und unterstützt GET-, POST-, PUT-, PATCH- und DELETE-Methoden zum Senden oder empfangen von Informationen zwischen vRealize Automation Code Stream und anderen Anwendungen.

Tabelle 5-7. Vorbereiten einer Pipeline für die Kommunikation über die REST API

| Aktion | Ergebnis |
|---|--|
| Fügen Sie der Pipeline eine REST-Aufgabe hinzu. | Die REST-Aufgabe kommuniziert Informationen zwischen Anwendungen und kann Statusinformationen für eine nachfolgende Aufgabe in der Pipeline bereitstellen. |
| Wählen Sie die REST-Aktion aus und schließen Sie dabei die URL ein. | Die Pipeline-Aufgabe ruft die URL auf, wenn die Pipeline ausgeführt wird. Für POST-, PUT- und PATCH-Aktionen müssen Sie eine Nutzlast hinzufügen. In der Nutzlast können Sie Ihre Pipeline- und Aufgabeneigenschaften binden, wenn die Pipeline ausgeführt wird. |
| Beachten Sie dieses Beispiel. | Beispiel für die Verwendung des REST-Plug-Ins: Sie können eine REST-Aufgabe hinzufügen, um ein Tag auf einem Git-Commit für einen Build zu erstellen. Zudem muss die Aufgabe eine Anforderung veröffentlichen, um die Eincheck-ID aus dem Repository abzurufen. Die Aufgabe kann eine Nutzlast an das Repository senden und ein Tag für den Build erstellen, und das Repository kann die Antwort mit dem Tag zurückgeben. |

Ähnlich wie bei Verwendung des REST-Plug-Ins zum Aufrufen einer API können Sie eine Abfrageaufgabe in Ihre Pipeline einfügen, um eine REST API aufzurufen und Sie abzufragen, bis diese abgeschlossen ist und die Pipeline Aufgabe die Beendigungskriterien erfüllt.

Sie können auch REST-APIs zum Importieren und Exportieren einer Pipeline verwenden und die Beispielskripts zum Ausführen einer Pipeline verwenden.

Bei dieser Vorgehensweise wird eine einfache URL abgerufen.

Verfahren

- 1 Um eine Pipeline zu erstellen, klicken Sie auf **Pipelines > Neue Pipeline > Leere Arbeitsfläche**.
- 2 Klicken Sie in Ihrer Pipeline-Phase auf **+ Sequenzielle Aufgabe**.
- 3 Fügen Sie im Aufgabenbereich die REST-Aufgabe hinzu:
 - a Geben Sie einen Namen für die Aufgabe ein.
 - b Wählen Sie im Dropdown-Menü „Typ“ die Option **REST** aus.
 - c Wählen Sie im Bereich „REST-Anforderung“ die Option **GET** aus.

Damit die REST-Aufgabe Daten von einer anderen Anwendung anfordern kann, wählen Sie die GET-Methode aus. Um Daten an eine andere Anwendung zu senden, wählen Sie die POST-Methode aus.

- d Geben Sie die URL zur Identifizierung des REST API-Endpoints ein. Beispiel: `https://www.google.com`.

Damit eine REST-Aufgabe Daten aus einer anderen Anwendung importieren kann, können Sie die Nutzlastvariable einbeziehen. Bei einer Importaktion können Sie beispielsweise `${Stage0.export.responseBody}` eingeben. Wenn die Größe der Antwortdaten 5 MB überschreitet, schlägt die REST-Aufgabe möglicherweise fehl.

- e Um die Autorisierung für die Aufgabe bereitzustellen, klicken Sie auf **Header hinzufügen** und geben Sie einen Headerschlüssel und einen Wert ein.

The screenshot shows the 'Test' interface with a 'Model' tab selected. A 'Stage0' contains a 'Task0' of type 'REST'. The right-hand configuration pane for 'Task0' is visible, showing the following settings:

- Task name:** Task0
- Type:** REST
- Continue on failure:** ☐
- Execute task:** ☒ Always ☐ On condition
- REST Request:**
 - Action:** GET
 - URL:** Enter URL
 - Agent endpoint:** --Select Agent endpoint--
 - Headers:**
 - Accept: application/json
 - Content-Type: application/json
- Output Parameters:** status

At the bottom, there are buttons for 'SAVE', 'RUN', and 'CLOSE', along with the text 'Last saved an hour ago'.

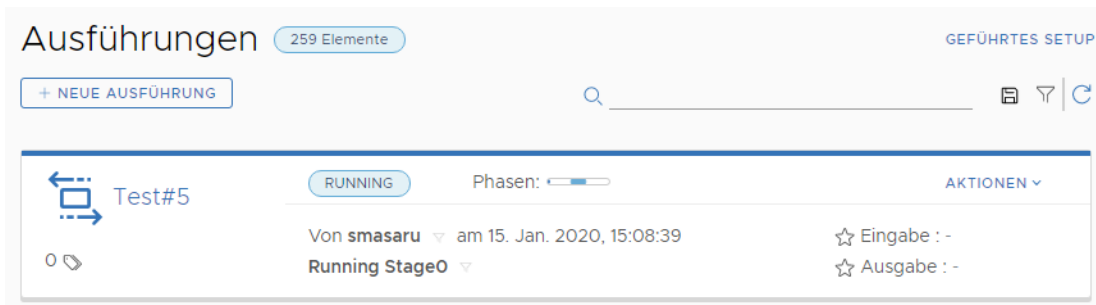
- 4 Um Ihre Pipeline zu speichern, klicken Sie auf **Speichern**.
- 5 Klicken Sie auf der Registerkarte „Pipeline“ auf **Pipeline aktivieren**.

The screenshot shows the 'Test' interface with a 'Model' tab selected. The 'Pipeline' configuration pane on the right is visible, showing the following settings:

- Projekt:** Project-1
- Name der Pipeline:** Test
- Pipeline aktivieren:** ☒
- Gleichzeitigkeit:** 10
- Beschreibung:**
- Symbol:** ☒ **ÄNDERN** **ENTFERNEN**
- Tags:** Tags für Pipeline eingeben

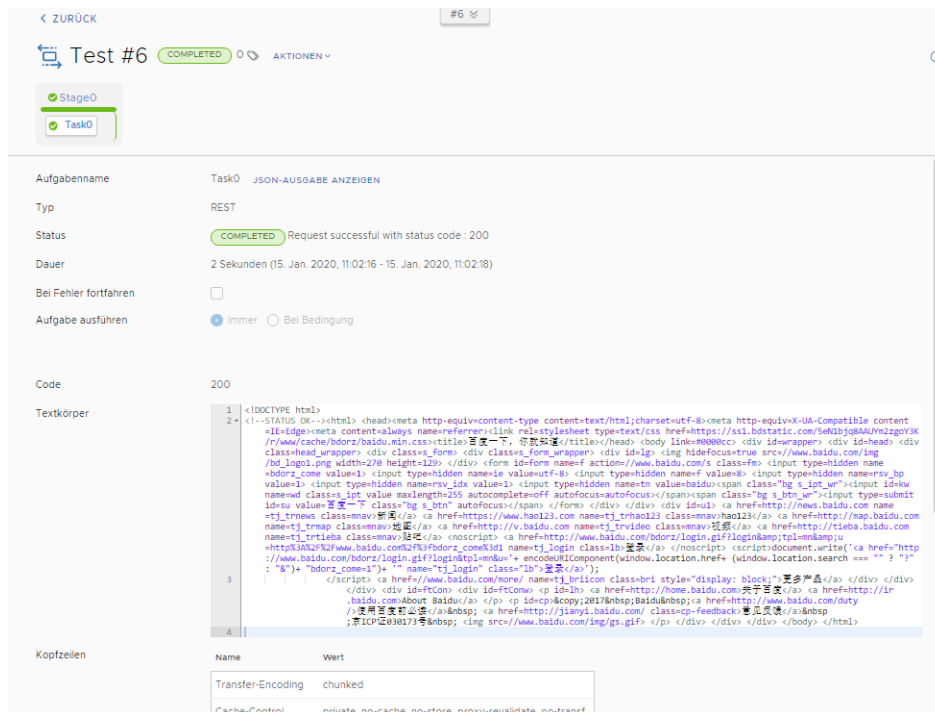
- 6 Klicken Sie auf **Speichern** und dann auf **Schließen**.
- 7 Klicken Sie auf **Ausführen**.

8 Um die Pipeline-Ausführung zu überwachen, klicken Sie auf **Ausführungen**.



- 9 Um zu überprüfen, ob die REST-Aufgabe die erwarteten Informationen zurückgibt, untersuchen Sie die Pipeline-Ausführung und die Aufgabenergebnisse.
 - a Nachdem die Pipeline abgeschlossen ist, klicken Sie auf den Link zur Ausführung der Pipeline, um zu bestätigen, dass die andere Anwendung die von Ihnen angeforderten Daten zurückgegeben hat.
 - b Klicken Sie auf die REST-Aufgabe in der Pipeline.
 - c Klicken Sie in der Pipeline-Ausführung auf die Aufgabe, sehen Sie sich die Aufgabendetails an und überprüfen Sie, ob die REST-Aufgabe die erwarteten Ergebnisse zurückgegeben hat.

Die Aufgabendetails zeigen den Antwortcode, den Textkörper, die Headerschlüssel und die Werte an.



10 Um die JSON-Ausgabe anzuzeigen, klicken Sie auf **JSON-AUSGABE ANZEIGEN**.

```

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
    =RTUkVjVhyg9KvAZR1S8yCCSEw8WosYf9mMDfQ1N5fNd5DvrxUM5B3J8PyKMX1Z_zRNp3usxttMpd7YiqRUOSfMkTC7cTERbd
    UmOnj3cTppHe3PHIXJPGHnT5ZEweb3cxtjVIhVolS85ezVXaTSRYfcg0B_XIHZ8kqB8uwl1aE; expires=Tue, 28-May-2019
    22:45:06 GMT; path=/; domain=.google.com; HttpOnly",
9    "Expires": "-1",
10   "P3P": "CP=\"This is not a P3P policy! See g.co/p3phelp for more info.\"";
11   "X-XSS-Protection": "1; mode=block",
12   "Date": "Mon, 26 Nov 2018 22:45:06 GMT",
13   "Content-Type": "text/html; charset=ISO-8859-1"
14 },
15 "responseBody": "<!doctype html><html itemscope=\"\" itemtype=\"http://schema.org/WebPage\" lang=\"en-IN\"
><head><meta content=\"text/html; charset=UTF-8\" http-equiv=\"Content-Type\"><meta content=\"/images
/branding/google/1x/google_standard_color_128dp.png\" itemprop=\"image\"><title>Google</title><script
nonce=\"aNww/ydugkGr9CHU6QQGz=\"><(function(){window.google={kEI:'cnf8w6KpJIEVkwXx-aLoDA',kEXPI:'0
,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457
,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,6056,636,2239
,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284
,2,579,727,612,1820,58,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8
,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483
,698,59,318,273,108,167,323,744,101,1119,38,363,557,438,135,145,155,497,2,718,383,978,487,47,1080,901
,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37
,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,247,109,1049,14

```

Path finder Enter key

Ergebnisse

Herzlichen Glückwunsch! Sie haben eine REST-Aufgabe konfiguriert, die eine REST API aufgerufen und Informationen zwischen vRealize Automation Code Stream und einer anderen Anwendung über das REST-Plug-In gesendet hat.

Nächste Schritte

Verwenden Sie weiterhin REST-Aufgaben in Ihren Pipelines, um Befehle auszuführen und vRealize Automation Code Stream in anderen Anwendungen zu integrieren, sodass Sie Ihre Softwareanwendungen entwickeln und bereitstellen können. Sie sollten Abfrageaufgaben verwenden, die die API abfragen, bis Sie abgeschlossen ist und die Pipeline-Aufgabe die Beendigungskriterien erfüllt.

Verbinden von vRealize Automation Code Stream mit Endpoints

6

vRealize Automation Code Stream wird über Plug-Ins in Entwicklungstools integriert. Zu den unterstützten Plug-Ins gehören u. a. Jenkins, Bamboo, vRealize Operations, Bugzilla, Team Foundation Server und Git.

Sie können auch Ihre eigenen Plug-Ins entwickeln, um vRealize Automation Code Stream mit anderen Entwicklungsanwendungen zu vernetzen.

Um vRealize Automation Code Stream mit JIRA zu vernetzen, benötigen Sie kein externes Plug-In, da vRealize Automation Code Stream die Erstellungsfunktion für das JIRA-Ticket als Benachrichtigungstyp enthält. Um JIRA-Tickets zum Pipeline-Status zu erstellen, müssen Sie einen JIRA-Endpoint hinzufügen.

Dieses Kapitel enthält die folgenden Themen:

- [Definition von Endpoints in vRealize Automation Code Stream](#)
- [Vorgehensweise zum Integrieren von vRealize Automation Code Stream in Jenkins](#)
- [Wie integriere ich vRealize Automation Code Stream in Git?](#)
- [Vorgehensweise zum Integrieren von vRealize Automation Code Stream in Gerrit](#)
- [Vorgehensweise zum Integrieren von vRealize Automation Code Stream in vRealize Orchestrator](#)

Definition von Endpoints in vRealize Automation Code Stream

Ein Endpoint ist eine Instanz einer DevOps-Anwendung, die eine Verbindung mit vRealize Automation Code Stream herstellt und Daten für die Ausführung Ihrer Pipelines bereitstellt, beispielsweise eine Datenquelle, ein Repository oder ein Benachrichtigungssystem.

Ihre Rolle in vRealize Automation Code Stream bestimmt, wie Sie Endpoints verwenden.

- Administratoren und Entwickler können Endpoints erstellen, aktualisieren, löschen und anzeigen.
- Administratoren können einen Endpoint als eingeschränkt markieren und Pipelines ausführen, die eingeschränkte Endpoints verwenden.

- Benutzer mit der Viewer-Rolle können Endpoints sehen, Sie können Sie jedoch nicht erstellen, aktualisieren oder löschen.

Weitere Informationen finden Sie unter [Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream](#).

Um vRealize Automation Code Stream mit einem Endpoint zu verbinden, fügen Sie eine Aufgabe in Ihrer Pipeline hinzu und konfigurieren Sie sie so, dass sie mit dem Endpoint kommuniziert. Um zu überprüfen, ob vRealize Automation Code Stream eine Verbindung zum Endpoint herstellen kann, klicken Sie auf **Validieren**. Wenn Sie dann die Pipeline ausführen, wird die Pipeline-Aufgabe mit dem Endpoint verbunden, um die Aufgabe auszuführen.

Weitere Informationen zu den Aufgabentypen, die diese Endpoints verwenden, finden Sie unter [In vRealize Automation Code Stream verfügbare Aufgabentypen](#).

Tabelle 6-1. Von vRealize Automation Code Stream unterstützte Endpoints

| Endpoint | Funktionalität | Unterstützte Versionen | Anforderungen |
|----------------------|--|---|---|
| Bambus | Erstellt Build-Pläne. | 6.9.* | |
| Docker | Native Builds können Docker-Hosts für die Bereitstellung verwenden. | | Wenn eine Pipeline ein Image aus dem Docker-Hub enthält, müssen Sie vor der Pipeline-Ausführung sicherstellen, dass cURL in das Image eingebettet wurde. Wenn die Pipeline ausgeführt wird, lädt vRealize Automation Code Stream eine binäre Datei herunter, die cURL zum Ausführen von Befehlen verwendet. |
| Docker-Registrierung | Registriert Container-Images, sodass ein Docker-Build-Host Images abrufen kann. | 2.7.1 | |
| Gerrit | Stellt eine Verbindung zu einem Gerrit-Server für Überprüfungen und Auslösung her | 2.14.* | |
| Git | Löst Pipelines aus, wenn Entwickler Code aktualisieren, und checkt den Code in das Repository ein. | Git Hub Enterprise 2.1.8 Git Lab Enterprise 11.9.12-ee | |
| Jenkins | Erstellt Code-Artefakte. | 1.6.* und 2.* | |
| Jira | Erstellt ein JIRA-Ticket, wenn eine Pipeline-Aufgabe fehlschlägt. | 8.3.* | |
| Kubernetes | Automatisiert die Schritte, die Containeranwendungen bereitstellen, skalieren und verwalten. | 1.9.* | |

Tabelle 6-1. Von vRealize Automation Code Stream unterstützte Endpoints (Fortsetzung)

| Endpoint | Funktionalität | Unterstützte | |
|-----------------------------|---|---------------|---------------|
| | | Versionen | Anforderungen |
| PowerShell | Erstellen Sie Aufgaben, die PowerShell-Skripts auf Windows- oder Linux-Maschinen ausführen. | 4 und 5 | |
| SSH | Erstellen Sie Aufgaben, die SSH-Skripts auf Windows- oder Linux-Maschinen ausführen. | 7.0 | |
| TFS, Team Foundation Server | Verwaltet den Quellcode, automatisierte Builds, Tests und zugehörige Aktivitäten. | 2015 und 2017 | |
| vRealize Orchestrator | Ordnet und automatisiert die Arbeitsabläufe in Ihrem Build-Prozess. | 7.* und 8.* | |

Beispiel-YAML-Code für einen GitHub-Endpoint

Dieser Beispiel-YAML-Code definiert einen GitHub-Endpoint, auf den Sie in einer Git-Aufgabe verweisen können.

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

Vorgehensweise zum Integrieren von vRealize Automation Code Stream in Jenkins

vRealize Automation Code Stream bietet ein Jenkins-Plug-In, das Jenkins-Aufträge auslöst, die Ihren Quellcode erstellen und testen. Das Jenkins-Plug-In führt Testfälle aus und kann benutzerdefinierte Skripts verwenden.

Um einen Jenkins-Auftrag in Ihrer Pipeline auszuführen, verwenden Sie einen Jenkins-Server und fügen den Jenkins-Endpoint in vRealize Automation Code Stream hinzu. Anschließend erstellen Sie eine Pipeline und fügen ihr eine Jenkins-Aufgabe hinzu.

Voraussetzungen

- Richten Sie einen Jenkins-Server ein, auf dem Version 1.561 oder höher ausgeführt wird.
- Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls Sie kein Mitglied sind, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).
- Stellen Sie sicher, dass auf dem Jenkins-Server eine Aufgabe vorhanden ist, damit sie von Ihrer Pipeline-Aufgabe ausgeführt werden kann.

Verfahren

- 1 Fügen Sie einen Jenkins-Endpoint hinzu und validieren Sie ihn.
 - a Klicken Sie auf **Endpoints > Neuer Endpoint**.
 - b Wählen Sie ein Projekt aus und wählen Sie **Jenkins** als Endpoint-Typ aus. Geben Sie dann einen Namen und eine Beschreibung ein.
 - c Wenn dieser Endpoint eine geschäftskritische Komponente in Ihrer Infrastruktur ist, aktivieren Sie **Als eingeschränkt kennzeichnen**.
 - d Geben Sie die URL für den Jenkins-Server ein.

- e Geben Sie den Benutzernamen und das Kennwort für die Anmeldung beim Jenkins-Server ein. Geben Sie dann die verbleibenden Informationen ein.

Tabelle 6-2. Verbleibende Informationen für den Jenkins-Endpoint

| Endpoint-Eintrag | Beschreibung |
|---|--|
| Ordnerpfad | <p>Pfad für den Ordner, in dem Ihre Aufträge gruppiert werden. Jenkins kann alle Aufträge im Ordner ausführen. Sie können Unterordner erstellen. Beispiel:</p> <ul style="list-style-type: none"> ■ folder_1 kann job_1 enthalten ■ folder_1 kann folder_2 enthalten, der job_2 enthalten kann <p>Wenn Sie einen Endpoint für folder_1 erstellen, lautet der Ordnerpfad job/folder_1 und der Endpoint listet nur job_1 auf.</p> <p>Um die Liste der Aufträge im untergeordneten Ordner folder_2 zu erhalten, müssen Sie einen weiteren Endpoint erstellen, der den Ordnerpfad als /job/folder_1/job/folder_2/ verwendet.</p> |
| URL | Die Host-URL des Jenkins-Servers Geben Sie die URL in der Form Protokoll:// Host:Port ein. Beispiel: http://192.10.121.13:8080 |
| Abrufintervall | Intervalldauer für vRealize Automation Code Stream, um Updates vom Jenkins-Server abzurufen. |
| Anzahl der Wiederholungen der Anforderung | Anzahl der Wiederholungen der geplanten Build-Anforderung für den Jenkins-Server. |
| Wartezeit für Wiederholungsversuche | Anzahl der Sekunden, die gewartet werden muss, bevor die Build-Anforderung für den Jenkins-Server erneut ausgeführt wird. |

- f Klicken Sie auf **Validieren** und stellen Sie sicher, dass der Endpoint mit vRealize Automation Code Stream verbunden ist. Wenn keine Verbindung hergestellt wird, korrigieren Sie alle Fehler und klicken Sie auf **Speichern**.

Endpoint bearbeiten

| | |
|-----------------------|--|
| Projekt | test1 |
| Typ | Jenkins |
| Name * | aa |
| Beschreibung | <input type="text"/> |
| Als eingeschränkt... | <input type="checkbox"/> nicht eingeschränkt |
| URL * | http(s)://<server_url>:<port> |
| Username | username |
| Password | password VARIABLE ERSTELLEN |
| Folder Path | /job/DevFolder/ |
| Poll Interval (sec) * | 15 |
| Request Retries * | 5 |
| Retry Wait Time ... * | 60 |

[SPEICHERN](#)
[ÜBERPRÜFEN](#)
[ABBRECHEN](#)

- 2 Um Ihren Code zu erstellen, erstellen Sie eine Pipeline und fügen Sie eine Aufgabe hinzu, die Ihren Jenkins-Endpoint verwendet.
 - a Klicken Sie auf **Pipelines > Neue Pipeline > Leere Arbeitsfläche**.
 - b Klicken Sie auf die Standardphase.
 - c Geben Sie im Aufgabenbereich einen Namen für die Aufgabe ein.
 - d Wählen Sie **Jenkins** als Aufgabentyp aus.
 - e Wählen Sie den von Ihnen erstellten Jenkins-Endpoint aus.
 - f Wählen Sie im Dropdown-Menü einen Auftrag aus dem Jenkins-Server aus, den Ihre Pipeline ausführen soll.
 - g Geben Sie die Parameter für den Auftrag ein.
 - h Geben Sie das Authentifizierungs-Token für den Jenkins-Auftrag ein.

The screenshot displays the 'Build and Deploy' configuration window in vRealize Automation. The interface is divided into two main sections: a visual pipeline editor on the left and a task configuration panel on the right.

Visual Pipeline Editor (Left):

- The title bar reads 'Build and Deploy' with a green 'Enabled' status indicator.
- The main workspace shows a stage named 'Stage0' containing two tasks: 'Build' and 'Test', both of type 'Jenkins'.
- Below 'Stage0' is a dashed box labeled '+ Parallel Task *'.
- At the bottom of the workspace is a dashed box labeled '+ Stage'.
- Navigation icons (back, forward, search, etc.) are visible at the top of the workspace.

Task Configuration Panel (Right):

- The tab 'Task : Build' is selected, with a 'Notifications' tab also visible.
- A 'VALIDATE TASK' button is located at the top right of the panel.
- Fields for task configuration include:
 - Task name:** Build
 - Type:** Jenkins (dropdown)
 - Continue On Failure:** ☐
 - Execute Task:** ☒ Always ☐ On Condition
 - Jenkins Endpoint:** aa (dropdown)
 - Job:** add_numbers (dropdown)
 - Num1:** 22
 - Num2:** 22
 - Token:** (empty text field)
- Output Parameters:** A row of buttons labeled status, job, jobId, jobResults, and jobUrl.

Bottom Bar:

- Buttons for 'SAVE', 'RUN', and 'CLOSE' are present.
- Text indicates 'Last saved a month ago'.
- A circular icon with a document symbol is on the far right.

- 3 Aktivieren und führen Sie Ihre Pipeline aus und zeigen Sie die Pipeline-Ausführung an.

[< 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 Zeigen Sie die Ausführungsdetails und den Status im Pipeline-Dashboard an.

Sie können alle Fehler und deren Ursachen identifizieren. Sie können auch Trends über die Dauer der Pipelineausführung, Fertigstellungen und Ausfälle anzeigen.

The screenshot shows the 'Build and Deploy' interface in vRealize Automation. It displays a list of recent successful executions under the heading 'Zuletzt erfolgte Ausführungen'. Below this, a table titled 'Ausführungsdetails' provides a detailed view of the execution results.

| Ausführung# | Status | Statusmeldung | Dauer | Aktualisiert |
|-------------|-----------|---|-------------|---------------------|
| #15 | COMPLETED | Execution Completed. | 13 Sekunden | 15. Jan. 2014:35:32 |
| #14 | FAILED | Stage0.Build: Unable to execute request : www.baidu232.com: Name or service not known | 13 Sekunden | 15. Jan. 2014:35:19 |
| #13 | COMPLETED | Execution Completed. | 13 Sekunden | 15. Jan. 2014:34:14 |
| #12 | COMPLETED | Execution Completed. | 13 Sekunden | 15. Jan. 2014:34:06 |
| #11 | COMPLETED | Execution Completed. | 13 Sekunden | 15. Jan. 2014:33:57 |
| #10 | COMPLETED | Execution Completed. | 12 Sekunden | 15. Jan. 2014:33:57 |

Ergebnisse

Herzlichen Glückwunsch! Sie haben vRealize Automation Code Stream mit Jenkins integriert, indem Sie einen Endpoint hinzugefügt, eine Pipeline erstellt und eine Jenkins-Aufgabe konfiguriert haben, die Ihren Code erstellt.

Beispiel: Beispiel-YAML für eine Jenkins-Build-Aufgabe

Für den Typ der in diesem Beispiel verwendeten Jenkins-Build-Aufgabe ähnelt die YAML dem folgenden Code, wobei die Benachrichtigungen aktiviert sind:

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

Nächste Schritte

Lesen Sie die anderen Abschnitte, um mehr zu erfahren. Weitere Informationen hierzu finden Sie unter [Kapitel 6 Verbinden von vRealize Automation Code Stream mit Endpoints](#).

Wie integriere ich vRealize Automation Code Stream in Git?

vRealize Automation Code Stream bietet eine Möglichkeit, eine Pipeline auszulösen, wenn ein Codewechsel in Ihrem GitHub-, GitLab- oder Bitbucket-Repository auftritt. Der Git-Auslöser

verwendet einen Git-Endpoint auf dem Zweig des Repositorys, den Sie überwachen möchten. vRealize Automation Code Stream verbindet sich über einen Webhook mit dem Git-Endpoint.

Um in vRealize Automation Code Stream einen Git-Endpoint zu definieren, wählen Sie ein Projekt aus und geben den Zweig des Git-Repositorys ein, in dem sich der Endpoint befindet. Das Projekt gruppiert die Pipeline mit dem Endpoint und anderen verwandten Objekten. Wenn Sie das Projekt in Ihrer Webhook-Definition auswählen, wählen Sie den Endpoint und die auszulösende Pipeline aus.

Hinweis Wenn Sie einen Webhook mit Ihrem Endpoint definieren und den Endpoint später bearbeiten, können Sie die Details des Endpoints im Webhook nicht ändern. Zum Ändern der Endpoint-Details müssen Sie den Webhook löschen und mit dem Endpoint neu definieren. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwenden des Git-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#).

Sie können mehrere Webhooks für verschiedene Zweige erstellen, indem Sie denselben Git-Endpoint verwenden und verschiedene Werte für den Namen des Zweigs auf der Konfigurationsseite des Webhooks bereitstellen. Zum Erstellen eines weiteren Webhooks für einen anderen Zweig im selben Git-Repository muss der Git-Endpoint nicht mehrmals für mehrere Zweige geklont werden. Stattdessen geben Sie den Namen des Zweigs im Webhook an, wodurch Sie den Git-Endpoint wiederverwenden können. Wenn der Zweig im Git-Webhook mit dem Zweig im Endpoint übereinstimmt, müssen Sie den Namen des Zweigs nicht auf der Seite des Git-Webhooks angeben.

Voraussetzungen

- Überprüfen Sie, ob Sie auf das GitHub-, GitLab- oder Bitbucket-Repository zugreifen können, zu dem Sie eine Verbindung herstellen möchten.
- Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls nicht, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).

Verfahren

- 1 Definieren Sie einen Git-Endpoint.
 - a Klicken Sie auf **Endpoints > Neuer Endpoint**.
 - b Wählen Sie ein Projekt aus, und wählen Sie als Endpoint-Typ **Git** aus. Geben Sie dann einen Namen und eine Beschreibung ein.

- c Wenn dieser Endpoint eine geschäftskritische Komponente in Ihrer Infrastruktur ist, aktivieren Sie **Als eingeschränkt kennzeichnen**.

Bei Verwendung eines eingeschränkten Endpoints in einer Pipeline kann ein Administrator die Pipeline ausführen und muss die Pipeline-Ausführung genehmigen. Wenn ein Endpoint oder eine Variable als eingeschränkt markiert ist und ein Nicht-Administratorbenutzer die Pipeline auslöst, hält die Pipeline bei dieser Aufgabe an und wartet darauf, dass sie von einem Administrator fortgesetzt wird.

Ein Projektadministrator kann eine Pipeline mit eingeschränkten Endpoints oder Variablen starten, wenn sich die Ressourcen in dem Projekt befinden, in dem der Benutzer als Projektadministrator fungiert.

Wenn ein Benutzer, der kein Administrator ist, versucht, eine Pipeline auszuführen, die eine eingeschränkte Ressource enthält, stoppt die Pipeline bei der Aufgabe, die die eingeschränkte Ressource verwendet. Anschließend muss ein Administrator die Ausführung der Pipeline fortsetzen.

Weitere Informationen zu eingeschränkten Ressourcen und benutzerdefinierten Rollen, die die Berechtigung mit der Bezeichnung **Eingeschränkte Pipelines verwalten** enthalten, finden Sie unter:

- [Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream](#)
- [Kapitel 2 Einrichten von vRealize Automation Code Stream zum Modellieren des Freigabeprozesses](#)

- d Wählen Sie einen der unterstützten Git-Servertypen aus.
- e Geben Sie die URL für das Repository mit dem API-Gateway für den Server im Pfad ein. Geben Sie beispielsweise **`https://api.github.com/vmware-example/repo-example`** ein.

- f Geben Sie den Zweig im Repository ein, in dem sich der Endpoint befindet.
- g Wählen Sie den Authentifizierungstyp aus und geben Sie den Benutzernamen für GitHub, GitLab oder Bitbucket ein. Geben Sie dann das Kennwort, das private Token oder den privaten Schlüssel für den Benutzernamen ein.

- Kennwort. Mit dem Kennwort erhalten Sie Vollzugriff auf das Repository. Sie können auch eine Variable für das Kennwort erstellen.


Verwenden Sie geheime Variablen, um vertrauliche Informationen auszublenden und zu verschlüsseln. Verwenden Sie eingeschränkte Variablen für Zeichenfolgen, Kennwörter und URLs, die ausgeblendet und verschlüsselt sein müssen, sowie zur Einschränkung ihrer Nutzung in Ausführungen. Verwenden Sie beispielsweise eine geheime Variable für ein Kennwort oder eine URL. Sie können geheime und eingeschränkte Variablen in jeder Art von Aufgabe in Ihrer Pipeline verwenden.

- Privates Token. Dieses Token ist Git-spezifisch und ermöglicht den Zugriff auf eine spezifische Aktion. Weitere Informationen hierzu finden Sie unter https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html. Sie können auch eine Variable für das private Token erstellen.
- Privater Schlüssel. Dieser SSH-Schlüssel ist ein privater Schlüssel, der den Zugriff auf ein spezifisches Repository ermöglicht. Wenn ein Git-Ereignis auftritt, verwendet vRealize Automation Code Stream diesen Schlüssel zum Klonen eines Repositories. Weitere Informationen finden Sie unter <https://help.github.com/articles/reviewing-your-ssh-keys/>.

- 2 Klicken Sie auf **Validieren** und überprüfen Sie, ob sich der Endpoint mit vRealize Automation Code Stream verbindet.

Wenn er sich nicht verbindet, beheben Sie die entsprechenden Fehler und klicken Sie dann auf **Erstellen**.

New endpoint

| | |
|-----------------------|--|
| Project * | test |
| Type * | GIT |
| Name * | DemoApp-Git |
| Description | Git example branch |
| Mark restricted | <input type="checkbox"/> 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

Nächste Schritte

Weitere Informationen finden Sie in den anderen Abschnitten. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwenden des Git-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#).

Vorgehensweise zum Integrieren von vRealize Automation Code Stream in Gerrit

Mit vRealize Automation Code Stream können Sie eine Pipeline auslösen, wenn eine Codeüberprüfung in Ihrem Gerrit-Projekt stattfindet. Der Auslöser für die Gerrit-Definition umfasst das Gerrit-Projekt und die Pipelines, die für verschiedene Ereignistypen ausgeführt werden müssen.

Der Auslöser für Gerrit verwendet einen Gerrit-Listener auf dem zu überwachenden Gerrit-Server. Um in vRealize Automation Code Stream einen Gerrit-Endpoint zu definieren, wählen Sie ein Projekt aus und geben die URL für den Gerrit-Server ein. Anschließend geben Sie den Endpoint an, wenn Sie einen Gerrit-Listener auf diesem Server erstellen.

Wenn Sie einen Gerrit-Server als vRealize Automation Code Stream-Endpoint in einer FIPS-fähigen vRealize Automation-Instanz verwenden, müssen Sie sicherstellen, dass Ihre Gerrit-Konfigurationsdatei die korrekten MAC-Schlüssel (Message Authentication Code) enthält. Wenn die Konfigurationsdatei des Gerrit-Servers nicht die korrekten MAC-Schlüssel enthält, kann der Server nicht ordnungsgemäß gestartet werden. Folgende Meldung wird angezeigt: `PrivateKey/PassPhrase is incorrect`

Voraussetzungen

- Vergewissern Sie sich, dass Sie auf den Gerrit-Server zugreifen können, mit dem Sie eine Verbindung herstellen möchten.
- Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls Sie kein Mitglied sind, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).

Verfahren

1 Definieren Sie einen Gerrit-Endpoint.

- a Klicken Sie auf **Konfigurieren > Endpoints** und dann auf **Neuer Endpoint**.
- b Wählen Sie ein Projekt aus und wählen Sie **Gerrit** als Endpoint-Typ aus. Geben Sie dann einen Namen und eine Beschreibung ein.
- c Wenn dieser Endpoint eine geschäftskritische Komponente in Ihrer Infrastruktur ist, aktivieren Sie **Als eingeschränkt kennzeichnen**.
- d Geben Sie die URL für den Gerrit-Server ein.

Sie können eine Portnummer mit der URL angeben oder keinen Wert eingeben, um den Standardport zu verwenden.

- e Geben Sie den Benutzernamen und das Kennwort für den Gerrit-Server ein.

Wenn das Kennwort verschlüsselt werden muss, klicken Sie auf **Variable erstellen** und wählen Sie den Typ aus:

- Geheimer Schlüssel: Das Kennwort wird aufgelöst, wenn ein Benutzer mit einer beliebigen Rolle die Pipeline ausführt.
- Eingeschränkt: Das Kennwort wird aufgelöst, wenn ein Benutzer mit der „Admin“-Rolle die Pipeline ausführt.

Geben Sie als Wert das Kennwort ein, das sicher sein muss, z. B. das Kennwort eines Jenkins-Servers.

- f Geben Sie für den privaten Schlüssel den SSH-Schlüssel ein, der für den sicheren Zugriff auf den Gerrit-Server verwendet wird.

Dieser Schlüssel ist der private RSA-Schlüssel, der sich im `.ssh`-Verzeichnis befindet.

- g (Optional) Wenn dem privaten Schlüssel eine Passphrase zugeordnet ist, geben Sie die Passphrase ein.

Um die Passphrase zu verschlüsseln, klicken Sie auf **Variable erstellen** und wählen Sie den Typ aus:

- Geheimer Schlüssel: Das Kennwort wird aufgelöst, wenn ein Benutzer mit einer beliebigen Rolle die Pipeline ausführt.
- Eingeschränkt: Das Kennwort wird aufgelöst, wenn ein Benutzer mit der „Admin“-Rolle die Pipeline ausführt.

Geben Sie als Wert die Passphrase ein, die sicher sein muss, z. B. die Passphrase für einen SSH-Server.

- 2 Klicken Sie auf **Validieren** und vergewissern Sie sicher, dass der Gerrit-Endpoint in vRealize Automation Code Stream mit dem Gerrit-Server verbunden ist.

Wenn keine Verbindung hergestellt wird, korrigieren Sie alle Fehler und klicken Sie erneut auf **Validieren**.

Neuer Endpoint

Projekt *

Typ *

Name *

Beschreibung

Als eingeschränkt kenn... ☐ nicht eingeschränkt

Cloud-Proxy *

URL *

Username *

Password * **VARIABLE ERSTELLEN**

Private Key *

Pass Phrase ⓘ **VARIABLE ERSTELLEN**

ERSTELLEN **ÜBERPRÜFEN** **ABBRECHEN**

- 3 Klicken Sie auf **Erstellen**.

- 4 Stellen Sie sicher, dass FIPS in der vRealize Automation-Umgebung aktiviert ist. Alternativ kann der Jenkins-Job zum Erstellen der FIPS-fähigen Umgebung mithilfe der Jenkins-URL verwendet werden.
 - a Zum Ausführen des Befehls über die Befehlszeile stellen Sie über SSH eine Verbindung zur vRealize Automation 8.x-Appliance her und melden sich als Root-Benutzer an.
Beispiel: Stellen Sie eine Verbindung zur vollqualifizierten Domännennamen-URL, wie z. B. `https://cava-1-234-567.yourcompanyFQDN.com`, auf Port 22, 5480 oder 443 her.
 - b Für die Suche nach FIPS in vRealize Automation führen Sie den Befehl `vracli security fips` aus.
 - c Stellen Sie sicher, dass `FIPS mode: strict` vom Befehl zurückgegeben wird.
- 5 Wenn Ihr Gerrit-Server als Endpoint in einer FIPS-fähigen vRealize Automation-Instanz fungiert, stellen Sie sicher, dass die Gerrit-Konfigurationsdatei die korrekten MAC-Schlüssel (Message Authentication Code) enthält.
 - a Öffnen Sie Gerrit und erstellen Sie ein SSH-Schlüsselpaar.
 - b Suchen Sie nach der Konfigurationsdatei des Gerrit-Servers unter `'$site_path'/etc/gerrit.config`.
 - c Stellen Sie sicher, dass die Konfigurationsdatei des Gerrit-Servers mindestens einen MAC-Schlüssel (Message Authentication Code) enthält, ausgenommen `hmac-MD5`.

Hinweis Im FIPS-Modus stellt `hmac-MD5` einen nicht unterstützten MAC-Algorithmus dar. Um den ordnungsgemäßen Start des Gerrit-Servers zu gewährleisten, darf die Konfigurationsdatei des Gerrit-Servers diesen Algorithmus nicht enthalten. Bei nicht ordnungsgemäßem Start des Gerrit-Servers wird folgende Meldung angezeigt:

```
PrivateKey/PassPhrase is incorrect
```

Unterstützte Schlüsselnamen des MAC-Schlüssels (Message Authentication Code), die mit einem Pluszeichen (+) beginnen, werden aktiviert. Die MAC-Schlüsselnamen, die mit einem Minuszeichen (-) beginnen, werden aus der Liste der Standard-MACs entfernt. Standardmäßig stehen diese unterstützten MACs in vRealize Automation Code Stream für den Gerrit-Server zur Verfügung:

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

Nächste Schritte

Weitere Informationen finden Sie in den anderen Abschnitten. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwenden des Gerrit-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#).

Vorgehensweise zum Integrieren von vRealize Automation Code Stream in vRealize Orchestrator

vRealize Automation Code Stream kann mit vRealize Orchestrator (vRO) integriert werden, um die zugehörigen Funktionen durch Ausführung von vRO-Workflows zu erweitern. vRealize Orchestrator enthält viele vordefinierte Workflows, die in Tools von Drittanbietern integriert werden können. Diese Workflows helfen dabei, Ihre DevOps-Prozesse zu automatisieren und zu verwalten, Massenvorgänge zu automatisieren und vieles mehr.

Sie können beispielsweise einen Workflow in einer vRO-Aufgabe in Ihrer Pipeline verwenden, um einen Benutzer zu aktivieren, einen Benutzer zu entfernen, VMs zu verschieben, in Test-Frameworks zu integrieren, um den Code während der Ausführung der Pipeline zu testen, und vieles mehr. Sie können Codebeispiele für vRealize Orchestrator-Workflows in code.vmware.com durchsuchen.

Mit einem vRealize Orchestrator-Workflow kann Ihre Pipeline eine Aktion ausführen, während diese Ihre Anwendung erstellt, testet und bereitstellt. Sie können vordefinierte Workflows in Ihre Pipeline aufnehmen oder benutzerdefinierte Workflows erstellen und verwenden. Jeder Workflow enthält Eingaben, Aufgaben und Ausgaben.

Um einen vRO-Workflow in Ihrer Pipeline auszuführen, muss der Workflow in der Liste der verfügbaren Workflows in der vRO-Aufgabe angezeigt werden, die Sie in Ihre Pipeline aufnehmen.

Bevor der Workflow in der vRO-Aufgabe in Ihrer Pipeline angezeigt werden kann, muss ein Administrator die folgenden Schritte in vRealize Orchestrator durchführen:

- 1 Wenden Sie das CODESTREAM-Tag auf den vRO-Workflow an.
- 2 Markieren Sie den vRO-Workflow als „Global“.

Voraussetzungen

- Stellen Sie sicher, dass Sie als Administrator auf eine lokale Instanz von vRealize Orchestrator zugreifen können. Weitere Informationen erhalten Sie von Ihrem Administrator und in der [vRealize Orchestrator-Dokumentation](#).
- Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls nicht, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).
- Erstellen Sie in vRealize Automation Code Stream eine Pipeline und fügen Sie eine Phase hinzu.

Verfahren

- 1 Bereiten Sie als Administrator einen vRealize Orchestrator-Workflow für die Ausführung Ihrer Pipeline vor.
 - a In vRealize Orchestrator finden Sie den Workflow, den Sie in Ihrer Pipeline verwenden müssen, wie z. B. einen Workflow zum Aktivieren eines Benutzers.
Wenn Sie einen Workflow benötigen, der nicht vorhanden ist, können Sie ihn erstellen.
 - b Geben Sie **Tag-Workflow** in der Suchleiste ein, um nach dem Workflow mit der Bezeichnung `Tag-Workflow` zu suchen.
 - c Klicken Sie auf der Karte mit dem Namen `Tag-Workflow` auf **Ausführen**, wodurch der Konfigurationsbereich angezeigt wird.
 - d Geben Sie im Textbereich `Markierter Workflow` den Namen des in der vRealize Automation Code Stream-Pipeline zu verwendenden Workflows ein und wählen Sie ihn dann in der Liste aus.
 - e Geben Sie in den Bereichen `Tag` und `Wert` die Bezeichnung `CODESTREAM` in Großbuchstaben ein.
 - f Aktivieren Sie das Kontrollkästchen mit dem Namen **Globales Tag**.
 - g Klicken Sie auf **Ausführen**, wodurch das Tag mit dem Namen `CODESTREAM` an den Workflow angehängt wird, den Sie in Ihrer vRealize Automation Code Stream-Pipeline auswählen müssen.
 - h Klicken Sie im Navigationsbereich auf **Workflows** und bestätigen Sie, dass das Tag mit dem Namen `CODESTREAM` auf der Workflow-Karte angezeigt wird, die von Ihrer Pipeline ausgeführt werden soll.

Nachdem Sie sich bei vRealize Automation Code Stream angemeldet und eine vRO-Aufgabe zu Ihrer Pipeline hinzugefügt haben, wird der getaggte Workflow in der Workflow-Liste angezeigt.
- 2 Erstellen Sie in vRealize Automation Code Stream einen Endpoint für Ihre vRealize Orchestrator-Instanz.
 - a Klicken Sie auf **Endpoints > Neuer Endpoint**.
 - b Wählen Sie ein Projekt aus.
 - c Geben Sie einen relevanten Namen ein.

- d Geben Sie die URL des vRealize Orchestrator-Endpoints ein.

Verwenden Sie folgendes Format: **https://host-n-01-234.eng.vmware.com:8281**

Folgendes Format sollte nicht verwendet werden: https://host-n-01-234.eng.vmware.com:8281/vco/api

Bei der URL für eine vRealize Orchestrator-Instanz, die mit der vRealize Automation-Appliance integriert ist, handelt es sich um den FQDN für die Appliance ohne Port oder Pfad. Beispiel: **https://vra-appliance.yourdomain.local**

Für externe vRealize Orchestrator-Appliances ab vRealize Automation 8.x lautet der FQDN für die Appliance **https://vro-appliance.yourdomain.local**

Wenn beim Hinzufügen des Endpoints ein Problem auftritt, müssen Sie unter Umständen eine YAML-Konfiguration mit einem Fingerabdruck für das SHA-256-Zertifikat importieren und die Doppelpunkte entfernen. Beispiel: **B0:01:A2:72...** wird zu **B001A272...** Der YAML-Beispielcode ähnelt Folgendem:

```

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

Für externe vRealize Orchestrator-Appliances ab vRealize Automation 7.x lautet der FQDN für die Appliance **https://vro-appliance.yourdomain.local:8281/vco**

- e Klicken Sie auf **Zertifikat akzeptieren**, wenn die von Ihnen eingegebene URL ein Zertifikat benötigt.
- f Geben Sie den Benutzernamen und das Kennwort für den vRealize Orchestrator-Server an.

Wenn Sie einen nicht lokalen Benutzer für die Authentifizierung verwenden, müssen Sie den Domänenteil des Benutzernamens weglassen. Beispiel: Zur Authentifizierung mit **svc_vro@yourdomain.local** müssen Sie **svc_vro** im Textbereich **Username** eingeben.

- 3 Bereiten Sie Ihre Pipeline für die Ausführung der vRO-Aufgabe vor.
- Fügen Sie Ihrer Pipeline-Phase eine vRO-Aufgabe hinzu.
 - Geben Sie einen relevanten Namen ein.
 - Wählen Sie im Bereich „Workflow-Eigenschaften“ den vRealize Orchestrator-Endpoint aus.

- d Wählen Sie den Workflow aus, den Sie als CODESTREAM in vRealize Orchestrator markiert haben.

Wenn Sie einen von Ihnen erstellten benutzerdefinierten Workflow auswählen, müssen Sie möglicherweise die Eingabeparameterwerte eingeben.

- e Klicken Sie für **Aufgabe ausführen** auf **Bei Bedingung**.

The screenshot shows the configuration page for a task named 'vRO workflow'. At the top, there are tabs: 'Aufgabe :vRO workflow', 'Benachrichtigungen', 'Rollback', and 'AUFGABE VALIDIEREN'. The 'Aufgabe :vRO workflow' tab is active. Below the tabs, the configuration fields are as follows:

- Aufgabenname**: vRO workflow
- Typ**: vRO
- Bei Fehler fortfahren**: ☐
- Aufgabe ausführen**: ☐ Immer ☒ Bei Bedingung
- Bedingung**: A text input field with a blue border and an information icon on the right.
- Workflow-Eigenschaften**:
 - Endpoint**: vROEP
 - Workflow**: Test
- Ausgabeparameter**: Four buttons labeled 'status', 'workflowExecutionId', 'parameters', and 'properties'.

- f Geben Sie die Bedingungen ein, die beim Ausführen der Pipeline angewendet werden.

| Pipeline ausführen... | Bedingungen auswählen... |
|-----------------------|---|
| Bei Bedingung | <p>Führt die Pipeline-Aufgabe nur aus, wenn die Auswertung der definierten Bedingung „true“ ergibt. Wenn die Bedingung „false“ lautet, wird die Aufgabe übersprungen.</p> <p>Die vRO-Aufgabe ermöglicht es Ihnen, einen booleschen Ausdruck einzuschließen, der die folgenden Operanden und Operatoren verwendet.</p> <ul style="list-style-type: none"> ■ Pipeline-Variablen, wie beispielsweise <code>\${pipeline.variableName}</code>. Verwenden Sie bei der Eingabe von Variablen nur geschweifte Klammern. ■ Aufgabenausgabe-Variablen, wie beispielsweise <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code>. ■ Standard-Pipeline-Bindungsvariablen, wie beispielsweise <code>\${releasePipelineName}</code>. ■ Boolesche Werte, die der Groß-/Kleinschreibung unterliegen, wie beispielsweise <code>true</code>, <code>false</code>, <code>'true'</code>, <code>'false'</code>. ■ Ganzzahl- oder Dezimalwerte ohne Anführungszeichen. ■ Zeichenfolgenwerte, die mit einfachen oder doppelten Anführungszeichen wie <code>"test"</code> oder <code>'test'</code> verwendet werden. ■ Werte vom Typ „Zeichenfolge“ und „Numerisch“, wie beispielsweise <code>== Equals</code> und <code>!= Not Equals</code>. ■ Relationale Operatoren wie <code>></code>, <code>>=</code>, <code><</code> und <code><=</code>. ■ Boolesche Logik wie <code>&&</code> und <code> </code>. ■ Arithmetische Operatoren wie <code>+</code>, <code>-</code>, <code>*</code> und <code>/</code>. ■ Geschachtelte Ausdrücke mit runden Klammern. ■ Zeichenfolgen, die den Literalwert <code>ABCD</code> enthalten, werden als falsch ausgewertet und die Aufgabe wird übersprungen. ■ Unäre Operatoren werden nicht unterstützt. <p>Eine mögliche Beispielbedingung ist <code>\${Stage1.task1.output} == "Passed" \${pipeline.variableName} == 39</code></p> |
| Immer | Wenn Sie Immer auswählen, führt die Pipeline die Aufgabe ohne Bedingungen aus. |

- g Geben Sie eine Nachricht für die Begrüßung ein.
- h Klicken Sie auf **Aufgabe validieren** und beheben Sie alle auftretenden Fehler.
- 4 Speichern, aktivieren Sie Ihre Pipeline aus und führen sie sie aus.
- 5 Überprüfen Sie nach der Ausführung der Pipeline die Ergebnisse.
- Klicken Sie auf **Ausführungen**.
 - Klicken Sie auf die Pipeline.
 - Klicken Sie auf die Aufgabe.
 - Untersuchen Sie die Ergebnisse, den Eingabewert und die Eigenschaften.

Sie können die Ausführungs-ID des Workflows, die Person, die auf die Aufgabe reagiert hat, den Zeitpunkt der Reaktion und alle darin enthaltenen Kommentare identifizieren.

Ergebnisse

Herzlichen Glückwunsch! Sie haben einen vRealize Orchestrator-Workflow für die Verwendung in vRealize Automation Code Stream markiert und eine vRO-Aufgabe in Ihrer vRealize Automation Code Stream-Pipeline hinzugefügt, sodass ein Workflow ausgeführt wird, der eine Aktion in Ihrer DevOps-Umgebung automatisiert.

Beispiel: Ausgabeformat der vRO-Aufgabe

Das Ausgabeformat für eine vRO-Aufgabe ähnelt diesem Beispiel.

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

Nächste Schritte

Fügen Sie weiterhin vRO-Workflow-Aufgaben in Ihre Pipelines ein, damit Sie Aufgaben in Ihren Entwicklungs-, Test- und Produktionsumgebungen automatisieren können.

Auslösen von Pipelines in vRealize Automation Code Stream

7

Sie können vRealize Automation Code Stream eine Pipeline auslösen lassen, wenn bestimmte Ereignisse auftreten.

Beispiel:

- Der Docker-Auslöser kann eine Pipeline ausführen, wenn ein neues Artefakt erstellt oder aktualisiert wird.
- Der Auslöser für Git kann eine Pipeline auslösen, wenn Entwickler Code aktualisieren.
- Der Auslöser für Gerrit kann eine Pipeline auslösen, wenn Entwickler Code überprüfen.
- Der Befehl `curl` kann dazu verwendet werden, dass Jenkins die Pipeline nach Abschluss eines Builds auslöst.

Dieses Kapitel enthält die folgenden Themen:

- [Vorgehensweise zum Verwenden des Docker-Auslösers in vRealize Automation Code Stream zum Ausführen einer kontinuierlichen Bereitstellungs-Pipeline](#)
- [Vorgehensweise zum Verwenden des Git-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#)
- [Vorgehensweise zum Verwenden des Gerrit-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline](#)

Vorgehensweise zum Verwenden des Docker-Auslösers in vRealize Automation Code Stream zum Ausführen einer kontinuierlichen Bereitstellungs-Pipeline

Als vRealize Automation Code Stream-Administrator oder Entwickler können Sie den Docker-Auslöser in vRealize Automation Code Stream verwenden. Der Docker-Auslöser führt eine eigenständige Pipeline für die kontinuierliche Bereitstellung (Continuous Delivery, CD) aus, wenn ein Build-Artefakt erstellt oder aktualisiert wird. Der Docker-Auslöser führt die CD-Pipeline aus, die das neue oder aktualisierte Artefakt als Container-Image an das Docker-Hub-Repository überträgt. Die CD-Pipeline kann als Teil Ihrer automatisierten Builds ausgeführt werden.

Verwenden Sie den Docker-Auslöser beispielsweise zur kontinuierlichen Bereitstellung des aktualisierten Container-Images über die CD-Pipeline. Wenn Ihr Container-Image in die Docker-Registrierung eingecheckt wird, benachrichtigt der Webhook im Docker-Hub vRealize Automation Code Stream darüber, dass das Image geändert wurde. Diese Benachrichtigung löst aus, dass die CD-Pipeline mit dem aktualisierten Container-Image ausgeführt und das Image in das Docker-Hub-Repository hochgeladen wird.

Zur Verwendung des Docker-Images führen Sie mehrere Schritte in vRealize Automation Code Stream aus.

Tabelle 7-1. Verwenden des Docker-Triggers

| Vorgehensweise... | Weitere Informationen zu dieser Aktion... |
|---|--|
| Erstellen eines Docker-Registrierungs-Endpoints. | <p>Damit vRealize Automation Code Stream Ihre Pipeline auslöst, müssen Sie über einen Docker-Registrierungs-Endpoint verfügen. Wenn der Endpoint nicht vorhanden ist, können Sie beim Hinzufügen des Webhooks zum Docker-Auslöser eine Option zur Erstellung des Endpoints auswählen.</p> <p>Der Docker-Registrierungs-Endpoint enthält die URL für das Docker-Hub-Repository.</p> |
| Hinzufügen von Eingabeparametern zur Pipeline, die bei Ausführung der Pipeline automatisch Docker-Parameter einfügen. | <p>Sie können Docker-Parameter in die Pipeline einfügen. Parameter können den Namen, das Image, den Repository-Namen, den Repository-Namespace und das Tag des Docker-Ereignis-Besitzers enthalten.</p> <p>Sie können Eingabeparameter in Ihre CD-Pipeline einfügen, die vor dem Auslösen der Pipeline vom Docker-Webhook an die Pipeline übergeben werden.</p> |
| Erstellen eines Docker-Webhooks. | <p>Wenn Sie den Docker-Webhook in vRealize Automation Code Stream erstellen, wird im Docker-Hub ebenfalls ein entsprechender Webhook angelegt. Der Docker-Webhook in vRealize Automation Code Stream stellt über die im Webhook enthaltene URL eine Verbindung zum Docker-Hub her.</p> <p>Die Webhooks kommunizieren miteinander und lösen die Pipeline aus, wenn ein Artefakt im Docker-Hub erstellt oder aktualisiert wird.</p> <p>Wenn Sie den Docker-Webhook in vRealize Automation Code Stream aktualisieren oder löschen, wird der Webhook im Docker-Hub ebenfalls aktualisiert oder gelöscht.</p> |
| Konfigurieren und Hinzufügen einer Kubernetes-Aufgabe zur Pipeline. | <p>Wenn ein Artefakt im Docker-Hub-Repository erstellt oder aktualisiert wird, wird die Pipeline ausgelöst. Anschließend wird das Artefakt über die Pipeline dem Docker-Host in Ihrem Kubernetes-Cluster bereitgestellt.</p> |
| Einschließen einer lokalen YAML-Definition in die Aufgabe. | <p>Die auf die Bereitstellungsaufgabe angewendete YAML-Definition enthält das Docker-Container-Image und alle geheimen Schlüssel, die zum Abrufen des Images aus dem Repository für die Bereitstellung erforderlich sind.</p> |

Wenn ein Artefakt im Docker-Hub-Repository erstellt oder aktualisiert wird, benachrichtigt der Webhook im Docker-Hub den Webhook in vRealize Automation Code Stream, wodurch die Pipeline ausgelöst wird. Die folgenden Aktionen werden durchgeführt:

- 1 Der Docker-Hub sendet eine POST-Anfrage an die URL im Webhook.
- 2 vRealize Automation Code Stream führt den Docker-Auslöser aus.
- 3 Der Docker-Auslöser startet die CD-Pipeline.
- 4 Die CD-Pipeline überträgt das Artefakt an das Docker-Hub-Repository.
- 5 vRealize Automation Code Stream löst den zugehörigen Docker-Webhook aus, der eine CD-Pipeline ausführt, die das Artefakt auf Ihrem Docker-Host bereitstellt.

In diesem Beispiel erstellen Sie einen Docker-Endpoint und einen Docker-Webhook in vRealize Automation Code Stream, der Ihre Anwendung auf dem Kubernetes-Entwicklungscluster bereitstellt. Die Schritte enthalten den Beispielcode für die Nutzlast, die von Docker an die URL im Webhook gesendet wird, den verwendeten API-Code sowie den Authentifizierungscode mit dem sicheren Token.

Voraussetzungen

- Stellen Sie sicher, dass eine Pipeline für die kontinuierliche Bereitstellung (Continuous Delivery, CD) in Ihrer vRealize Automation Code Stream-Instanz vorhanden ist. Stellen Sie außerdem sicher, dass sie eine oder mehrere Kubernetes-Aufgaben enthält, die Ihre Anwendung bereitstellen. Weitere Informationen hierzu finden Sie unter [Kapitel 4 Planen eines nativen Builds, der Integration und Bereitstellung von Code in vRealize Automation Code Stream](#).
- Stellen Sie sicher, dass Sie auf einen vorhandenen Kubernetes-Cluster zugreifen können, auf dem die CD-Pipeline Ihre Anwendung für die Entwicklung bereitstellen kann.
- Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls nicht, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).

Verfahren

- 1 Erstellen eines Docker-Registrierungs-Endpoints.
 - a Klicken Sie auf **Endpoints**.
 - b Klicken Sie auf **Neuer Endpoint**.
 - c Beginnen Sie mit der Eingabe des Namens des vorhandenen Projekts.
 - d Wählen Sie **Docker-Registrierung** als Typ aus.
 - e Geben Sie einen relevanten Namen ein.
 - f Wählen Sie **Docker-Hub** als Servertyp aus.

- g Geben Sie die URL für das Docker-Hub-Repository ein.
- h Geben Sie den Namen und das Kennwort für den Zugriff auf das Repository ein.

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 Richten Sie die Eingabeeigenschaften in Ihrer CD-Pipeline so ein, dass Docker-Parameter bei Ausführung der Pipeline automatisch eingefügt werden.

sm-1 Aktiviert

Arbeitsbereich **Eingabe** Modell Ausgabe

Eingabeparameter ⓘ

Parameter automat... ☐ Gerrit ☐ Git ☒ Docker ☐ Keine

[HINZUFÜGEN](#) [EINGEFÜGTE PARAMETER HINZUFÜGEN](#) [ENTFERNEN](#)

| Mit Stern markier | Name |
|-------------------|----------------------------------|
| ⋮ ☆ | DOCKER_REGISTRY_EVENT_OWNER_NAME |
| ⋮ ☆ | DOCKER_REGISTRY_IMAGE |
| ⋮ ☆ | DOCKER_REGISTRY_REPO_NAME |
| ⋮ ☆ | DOCKER_REGISTRY_REPO_NAMESPACE |
| ⋮ ☆ | DOCKER_REGISTRY_TAG |

- 3 Erstellen eines Docker-Webhooks.
- Klicken Sie auf **Auslöser > Docker**.
 - Klicken Sie auf **Neuer Webhook für Docker**.
 - Wählen Sie ein Projekt aus.
 - Geben Sie einen relevanten Namen ein.
 - Wählen Sie den Docker-Registrierungs-Endpoint aus.

Wenn der Endpoint noch nicht vorhanden ist, klicken Sie auf **Endpoint erstellen** und erstellen Sie den Endpoint.

- Wählen Sie die Pipeline mit eingefügten Docker-Parametern aus, damit der Webhook ausgelöst wird. Weitere Informationen hierzu finden Sie unter [Schritt 2](#).

Wenn die Pipeline mit benutzerdefinierten hinzugefügten Eingabeparametern konfiguriert wurde, werden in der Liste der Eingabeparameter Parameter und Werte angezeigt. Sie können Werte für Eingabeparameter eingeben, die mit dem Auslöserereignis an die Pipeline übergeben werden. Alternativ können Sie auf die Eingabe von Werten verzichten oder die Standardwerte verwenden, sofern welche definiert sind.

Weitere Informationen zu Parametern auf der Registerkarte „Eingabe“ finden Sie unter [Planen eines nativen CICD-Builds in vRealize Automation Code Stream vor dem manuellen Hinzufügen von Aufgaben](#).

- g Geben Sie das API-Token ein.

Das CSP-API-Token authentifiziert Sie für externe API-Verbindungen mit vRealize Automation Code Stream. So rufen Sie das API-Token ab:

- 1 Klicken Sie auf **Token generieren**.
- 2 Geben Sie die E-Mail-Adresse ein, die mit Ihrem Benutzernamen und Kennwort verknüpft ist, und klicken Sie auf **Generieren**.

Das von Ihnen generierte Token ist sechs Monate lang gültig. Es wird auch als Aktualisierungstoken bezeichnet.

- Um das Token als Variable für die spätere Verwendung beizubehalten, klicken Sie auf **Variable erstellen**, geben Sie einen Namen für die Variable ein und klicken Sie auf **Speichern**.
- Um das Token als Textwert für die spätere Verwendung beizubehalten, klicken Sie auf **Kopieren** und fügen Sie das Token in eine Textdatei ein, um es lokal zu speichern.

Sie haben die Möglichkeit, eine Variable zu erstellen und das Token in einer Textdatei zur späteren Verwendung zu speichern.

- 3 Klicken Sie auf **Schließen**.

- h Geben Sie das Build-Image ein.

- i Geben Sie ein Tag ein.

Docker

Aktivität **Webhooks für Docker**

Webhook-URL [ⓘ] https://[REDACTED]/api/registry-webhook-listeners/1c9b3ae4-3f

Projekt test

Name * sm-1-Docker-WH

Beschreibung Docker webhook trigger for sm-1

Docker-Registrierung Docker-Register-Endpoint

Pipeline * sm-1 [ⓧ]

API-Token * [ⓧ] [TOKEN GENERIEREN](#)

Image [ⓘ] Image

Tag [ⓘ] Tags

[SPEICHERN](#) [ABBRECHEN](#)

- j Klicken Sie auf **Speichern**.

Die Webhook-Karte wird mit aktiviertem Docker-Webhook angezeigt. Wenn Sie einen Pseudo-Push zum Docker-Hub-Repository durchführen möchten, ohne den Docker-Webhook auszulösen und eine Pipeline auszuführen, klicken Sie auf **Deaktivieren**.

- 4 Konfigurieren Sie in der CD-Pipeline die Kubernetes-Bereitstellungsaufgabe.
 - a Wählen Sie in den Kubernetes-Aufgabeneigenschaften den Kubernetes-Entwicklungscluster aus.
 - b Wählen Sie die Aktion **Erstellen** aus.

- c Wählen Sie die **Lokale Definition** für die Nutzlastquelle aus.
- d Wählen Sie dann Ihre lokale YAML-Datei aus.

Beispiel: Der Docker-Hub sendet diese lokale YAML-Definition unter Umständen als Nutzlast an die URL im Webhook.

```
{
  "callback_url": "https://registry.hub.docker.com/u/svendowideit/testhook/hook/2141b5bi5i5b02bec211i4eeih0242eg11000a/",
  "push_data": {
    "images": [
      "27d47432a69bca5f2700e4dff7de0388ed65f9d3fb1ec645e2bc24c223dc1cc3",
      "51a9c7c1f8bb2fa19bcd09789a34e63f35abb80044bc10196e304f6634cc582c",
      "...",
    ],
    "pushed_at": 1.417566161e+09,
    "pusher": "trustedbuilder",
    "tag": "latest"
  },
  "repository": {
    "comment_count": 0,
    "date_created": 1.417494799e+09,
    "description": "",
    "dockerfile": "#\n# BUILD\u0009\u0009docker build -t svendowideit/apt-cacher .\n# RUN\u0009\u0009docker run -d -p 3142:3142 -name apt-cacher-run apt-cacher\n#\n# and then you can run containers with:\n#\n\u0009\u0009docker run -t -i -rm -e http_proxy http://192.168.1.2:3142/debian bash\n#\nFROM\u0009\u0009ubuntu\n\n\nVOLUME\u0009\u0009[/var/cache/apt-cacher-ng]\nRUN\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n\nEXPOSE\n\u0009\u000993142\nCMD\u0009\u0009chmod 777 /var/cache/apt-cacher-ng ; /etc/init.d/apt-cacher-ng start ; tail -f /var/log/apt-cacher-ng/*\n",
    "full_description": "Docker Hub based automated build from a GitHub repo",
    "is_official": false,
    "is_private": true,
    "is_trusted": true,
    "name": "testhook",
    "namespace": "svendowideit",
    "owner": "svendowideit",
    "repo_name": "svendowideit/testhook",
    "repo_url": "https://registry.hub.docker.com/u/svendowideit/testhook/",
    "star_count": 0,
    "status": "Active"
  }
}
```

Die API, die den Webhook im Docker-Hub erstellt, sieht folgendermaßen

aus: https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/

Der Text des JSON-Codes ähnelt Folgendem:

```
{
  "name": "demo_webhook",
```

```
"webhooks": [
{
"name": "demo_webhook",
"hook_url": "http://www.google.com"
}
]
}
```

Für den Empfang von Ereignissen vom Docker-Hub-Server verwendet das Authentifizierungsschema für den in vRealize Automation Code Stream erstellten Docker-Webhook einen Authentifizierungsmechanismus mittels einer Zulassungsliste mit einem zufälligen Zeichenfolgen-Token für den Webhook. Ereignisse werden auf Basis des sicheren Tokens gefiltert, den Sie an `hook_url` anhängen können.

vRealize Automation Code Stream kann jede Anfrage vom Docker-Hub-Server mithilfe des konfigurierten sicheren Tokens überprüfen. Beispiel: `hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""`

- 5 Erstellen Sie ein Docker-Artefakt in Ihrem Docker-Hub-Repository. Oder aktualisieren Sie ein vorhandenes Artefakt.
- 6 Um zu bestätigen, dass der Auslöser eingetreten ist, und die Aktivität auf dem Docker-Webhook anzuzeigen, klicken Sie auf **Auslöser > Docker > Aktivität**.

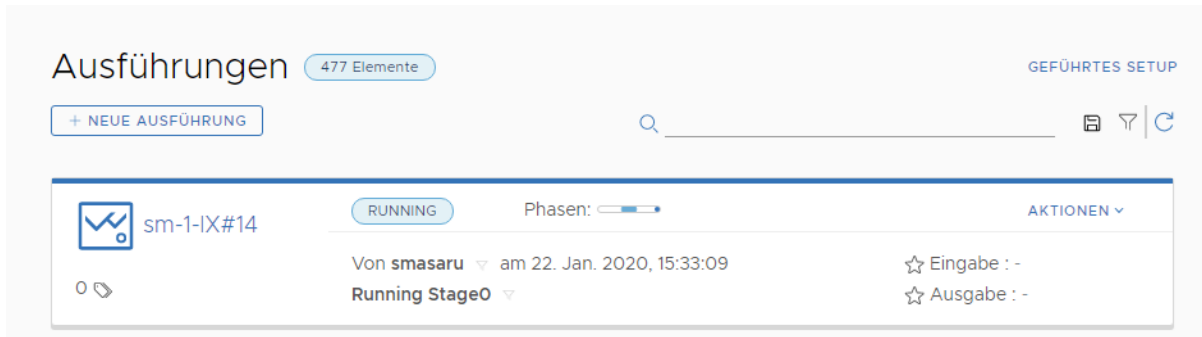
Docker

GUIDED SETUP

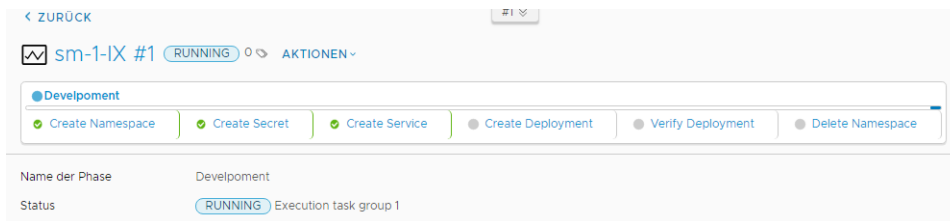
Activity
Webhooks for Docker

| 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 Klicken Sie auf **Ausführungen** und beobachten Sie Ihre Pipeline während der Ausführung.



- 8 Klicken Sie auf die laufende Phase und zeigen Sie die Aufgaben an, während die Pipeline ausgeführt wird.



Ergebnisse

Herzlichen Glückwunsch! Sie richten den Docker-Auslöser so ein, dass die CD-Pipeline kontinuierlich ausgeführt wird. Ihre Pipeline kann jetzt neue und aktualisierte Docker-Artefakte in das Docker-Hub-Repository hochladen.

Nächste Schritte

Stellen Sie sicher, dass das neue oder aktualisierte Artefakt auf dem Docker-Host im Kubernetes-Entwicklungscluster bereitgestellt wird.

Vorgehensweise zum Verwenden des Git-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline

Als vRealize Automation Code Stream-Administrator oder Entwickler können Sie vRealize Automation Code Stream mithilfe des Git-Auslösers in den Git-Lebenszyklus integrieren. Wenn Sie eine Codeänderung in GitHub, GitLab oder Bitbucket Enterprise vornehmen, kommuniziert das Ereignis mit vRealize Automation Code Stream über einen Webhook und löst die Ausführung einer Pipeline aus. Der Webhook ist mit GitLab, GitHub und lokalen Bitbucket-Unternehmensversionen kompatibel, wenn sowohl vRealize Automation Cloud Assembly als auch die Unternehmensversion im selben Netzwerk erreichbar sind.

Wenn Sie den Webhook für Git in vRealize Automation Code Stream hinzufügen, erstellt er auch einen Webhook im GitHub-, GitLab- oder Bitbucket-Repository. Wenn Sie den Webhook später aktualisieren oder löschen, wird der Webhook in GitHub, GitLab oder Bitbucket ebenfalls aktualisiert oder gelöscht.

Ihre Webhook-Definition muss einen Git-Endpoint auf dem Branch des Repositorys enthalten, das Sie überwachen möchten. vRealize Automation Code Stream verwendet den Git-Endpoint zum Erstellen des Webhooks. Wenn der Endpoint nicht vorhanden ist, können Sie ihn erstellen, wenn Sie den Webhook hinzufügen. In diesem Beispiel wird vorausgesetzt, dass Sie über einen vordefinierten Git-Endpoint in GitHub verfügen.

Sie können mehrere Webhooks für verschiedene Zweige erstellen, indem Sie denselben Git-Endpoint verwenden und verschiedene Werte für den Namen des Zweigs auf der Konfigurationsseite des Webhooks bereitstellen. Zum Erstellen eines weiteren Webhooks für einen anderen Zweig im selben Git-Repository muss der Git-Endpoint nicht mehrmals für mehrere Zweige geklont werden. Stattdessen geben Sie den Namen des Zweigs im Webhook an, wodurch Sie den Git-Endpoint wiederverwenden können. Wenn der Zweig im Git-Webhook mit dem Zweig im Endpoint übereinstimmt, müssen Sie den Namen des Zweigs nicht auf der Seite des Git-Webhooks angeben.

In diesem Beispiel wird die Verwendung des Git-Auslösers mit einem GitHub-Repository gezeigt. Zu den Voraussetzungen gehören jedoch Vorbereitungen, die bei Verwendung eines anderen Git-Servertyps erforderlich sind.

Voraussetzungen

- Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls nicht, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).
- Vergewissern Sie sich, dass Sie über einen Git-Endpoint auf dem GitHub-Branch verfügen, den Sie überwachen möchten. Weitere Informationen hierzu finden Sie unter [Wie integriere ich vRealize Automation Code Stream in Git?](#).
- Stellen Sie sicher, dass Sie über Rechte zum Erstellen eines Webhooks im Git-Repository verfügen.
- Wenn Sie einen Webhook in GitLab konfigurieren, ändern Sie die Standardnetzwerkeinstellungen in GitLab Enterprise, um ausgehende Anforderungen zu aktivieren und die Erstellung lokaler Webhooks zuzulassen.

Hinweis Diese Änderung ist nur für GitLab Enterprise erforderlich. Diese Einstellungen gelten nicht für GitHub oder Bitbucket.

- a Melden Sie sich bei der GitLab Enterprise-Instanz als Administrator an.
- b Navigieren Sie zu den Netzwerkeinstellungen, indem Sie eine URL wie `http://{gitlab-server}/admin/application_settings/network` verwenden.

- c Erweitern Sie **Ausgehende Anforderungen** und klicken Sie auf:
- Anforderung an das lokale Netzwerk aus Webhooks und Diensten zulassen.
 - Anforderung an das lokale Netzwerk aus Systemhook zulassen.
- Für die Pipelines, die Sie auslösen möchten, müssen Sie die Eingabeeigenschaften für das Einfügen von Git-Parametern bei der Ausführung der Pipeline festgelegt haben.

Build and Deploy Aktiviert

Arbeitsbereich **Eingabe** Modell Ausgabe

Eingabeparameter ⓘ

Parameter automatisch... ☐ Gerrit ☒ Git ☐ Docker ☐ Keine

HINZUFÜGEN EINGEFÜGTE PARAMETER HINZUFÜGEN/ENTFERNEN

| Mit Stern markiert ⓘ | Name |
|----------------------|-----------------------|
| ⋮ ☆ | GIT_BRANCH_NAME |
| ⋮ ☆ | GIT_CHANGE_SUBJECT |
| ⋮ ☆ | GIT_COMMIT_ID |
| ⋮ ☆ | GIT_EVENT_DESCRIPTION |
| ⋮ ☆ | GIT_EVENT_OWNER_NAME |
| ⋮ ☆ | GIT_EVENT_TIMESTAMP |
| ⋮ ☆ | GIT_REPO_NAME |
| ⋮ ☆ | GIT_SERVER_URL |

Informationen über Eingabeparameter finden Sie unter [Planen eines nativen CI/CD-Builds in vRealize Automation Code Stream](#) vor dem manuellen Hinzufügen von Aufgaben.

Verfahren

- 1 Klicken Sie in vRealize Automation Code Stream auf **Auslöser > Git**.
- 2 Klicken Sie auf die Registerkarte **Webhooks für Git** und dann auf **Neuer Webhook für Git**.
 - a Wählen Sie ein Projekt aus.
 - b Geben Sie einen Namen und eine Beschreibung für den Webhook ein.

- c Wählen Sie einen Git-Endpoint aus, der für den zu überwachenden Branch konfiguriert ist.

Wenn Sie Ihren Webhook erstellen, enthält die Webhook-Definition die aktuellen Endpoint-Details.

- Wenn Sie den Git-Typ, den Git-Servertyp oder die URL des Git-Repositorys zu einem späteren Zeitpunkt im Endpoint ändern, kann der Webhook keine Pipeline mehr auslösen, da er unter Verwendung der ursprünglichen Endpoint-Details auf das Git-Repository zugreift. Sie müssen den Webhook löschen und ihn erneut mit dem Endpoint erstellen.
- Wenn Sie den Authentifizierungstyp, den Benutzernamen oder das Kennwort zu einem späteren Zeitpunkt im Endpoint ändern, wird der Webhook weiterhin ausgeführt.

Weitere Informationen hierzu finden Sie unter [Wie integriere ich vRealize Automation Code Stream in Git?](#).

- d (Optional) Geben Sie den Branch ein, der vom Webhook überwacht werden soll.

Wenn Sie keinen Branch eingeben, überwacht der Webhook den für den Git-Endpoint konfigurierten Branch.

- e (Optional) Generieren Sie ein geheimes Token für den Webhook.

Bei Verwendung generiert vRealize Automation Code Stream ein zufälliges Zeichenfolgen-Token für den Webhook. Wenn der Webhook dann Git-Ereignisdaten empfängt, sendet er die Daten mit dem geheimen Token. vRealize Automation Code Stream verwendet die Informationen, um zu ermitteln, ob die Aufrufe von der erwarteten Quelle stammen, z. B. von der konfigurierten GitHub-Instanz, dem Repository und dem Branch. Das geheime Token bietet eine zusätzliche Sicherheitsebene, die verwendet wird, um zu überprüfen, ob die Git-Ereignisdaten aus der richtigen Quelle stammen.

f (Optional) Geben Sie Dateieinschlüsse oder -ausschlüsse als Bedingungen für den Auslöser an.

- Sie geben Dateieinschlüsse an, sodass Pipelines ausgelöst werden, wenn eine der Dateien in einem Commit mit den in den Einschlusspfaden oder dem Regex angegebenen Dateien übereinstimmt. Im Fall eines angegebenen Regex löst vRealize Automation Code Stream die Pipelines nur dann auf, wenn Dateinamen im Änderungssatz mit dem angegebenen Ausdruck übereinstimmen. Der Regex-Filter ist nützlich, wenn ein Auslöser für mehrere Pipelines in einem einzelnen Repository konfiguriert wird.
- Sie geben Dateiausschlüsse an, sodass die Pipelines nicht ausgelöst werden, wenn alle Dateien in einem Commit mit den angegebenen Dateien in den Ausschlusspfaden oder dem Regex übereinstimmen.
- Wenn diese Option aktiviert ist, wird durch die Priorisierung des Ausschlusses sichergestellt, dass Pipelines nicht ausgelöst werden, selbst wenn eine der Dateien in einem Commit die angegebenen Dateien in den Ausschlusspfaden oder im Regex übereinstimmt. Die Standardeinstellung ist „Aus“.

Wenn die Bedingungen sowohl für Einschluss als auch Ausschluss erfüllt sind, werden Pipelines nicht ausgelöst.

Im folgenden Beispiel sind sowohl Dateieinschlüsse als auch Dateiausschlüsse Bedingungen für den Auslöser.

The screenshot shows a configuration window titled 'Datei' with a sub-header 'Einschlüsse' (Inclusions) and 'Ausschlüsse' (Exclusions). Under 'Einschlüsse', there are two rules: one with 'PLAIN' type and path 'runtime/src/main/a.java', and another with 'REGEX' type and path '([a-z A-Z]+/[a-z A-Z])+'. Under 'Ausschlüsse', there are two rules: one with 'PLAIN' type and path 'runtime/pom.xml', and another with 'PLAIN' type and path 'runtime/demo.yaml'. At the bottom, there is a toggle switch labeled 'Ausschluss priorisieren' (Prioritize exclusion) which is currently turned off.

- Bei Dateieinschlüssen löst ein Commit mit einer Änderung in `runtime/src/main/a.java` oder einer beliebigen Java-Datei Pipelines aus, die in der Ereigniskonfiguration konfiguriert sind.
 - Bei Dateiausschlüssen werden bei einem Commit mit Änderungen nur in beiden Dateien die in den Ereigniskonfigurationen konfigurierten Pipelines nicht ausgelöst.
- g Wählen Sie für das Git-Ereignis eine **Push**- oder **Pull**-Anforderung aus.

- h Geben Sie das API-Token ein.

Das CSP-API-Token authentifiziert Sie für externe API-Verbindungen mit vRealize Automation Code Stream. So rufen Sie das API-Token ab:

- 1 Klicken Sie auf **Token generieren**.
- 2 Geben Sie die E-Mail-Adresse ein, die mit Ihrem Benutzernamen und Kennwort verknüpft ist, und klicken Sie auf **Generieren**.

Das von Ihnen generierte Token ist sechs Monate lang gültig. Es wird auch als Aktualisierungstoken bezeichnet.

- Um das Token als Variable für die spätere Verwendung beizubehalten, klicken Sie auf **Variable erstellen**, geben Sie einen Namen für die Variable ein und klicken Sie auf **Speichern**.
- Um das Token als Textwert für die spätere Verwendung beizubehalten, klicken Sie auf **Kopieren** und fügen Sie das Token in eine Textdatei ein, um es lokal zu speichern.

Sie haben die Möglichkeit, eine Variable zu erstellen und das Token in einer Textdatei zur späteren Verwendung zu speichern.

- 3 Klicken Sie auf **Schließen**.

- i Wählen Sie die Pipeline aus, die der Webhook auslösen soll.

Wenn die Pipeline mit benutzerdefinierten hinzugefügten Eingabeparametern konfiguriert wurde, werden in der Liste der Eingabeparameter Parameter und Werte angezeigt. Sie können Werte für Eingabeparameter eingeben, die mit dem Auslöserereignis an die Pipeline übergeben werden. Alternativ können Sie auf die Eingabe von Werten verzichten oder die Standardwerte verwenden, sofern welche definiert sind.

Informationen zum automatischen Einfügen von Eingabeparametern für Git-Auslöser finden Sie unter [Voraussetzungen](#).

- j Klicken Sie auf **Erstellen**.

Der Webhook wird als neue Karte angezeigt.

- 3 Klicken Sie auf die Webhook-Karte.

Wenn das Webhook-Datenformular erneut angezeigt wird, wird im oberen Bereich des Formulars eine Webhook-URL hinzugefügt. Der Git-Webhook wird über die Webhook-URL mit dem GitHub-Repository verbunden.

Git

Activity

Webhooks for Git

Webhook URL ⓘ

https://ca8b18e1-4b34-4389-9000-000000000000/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

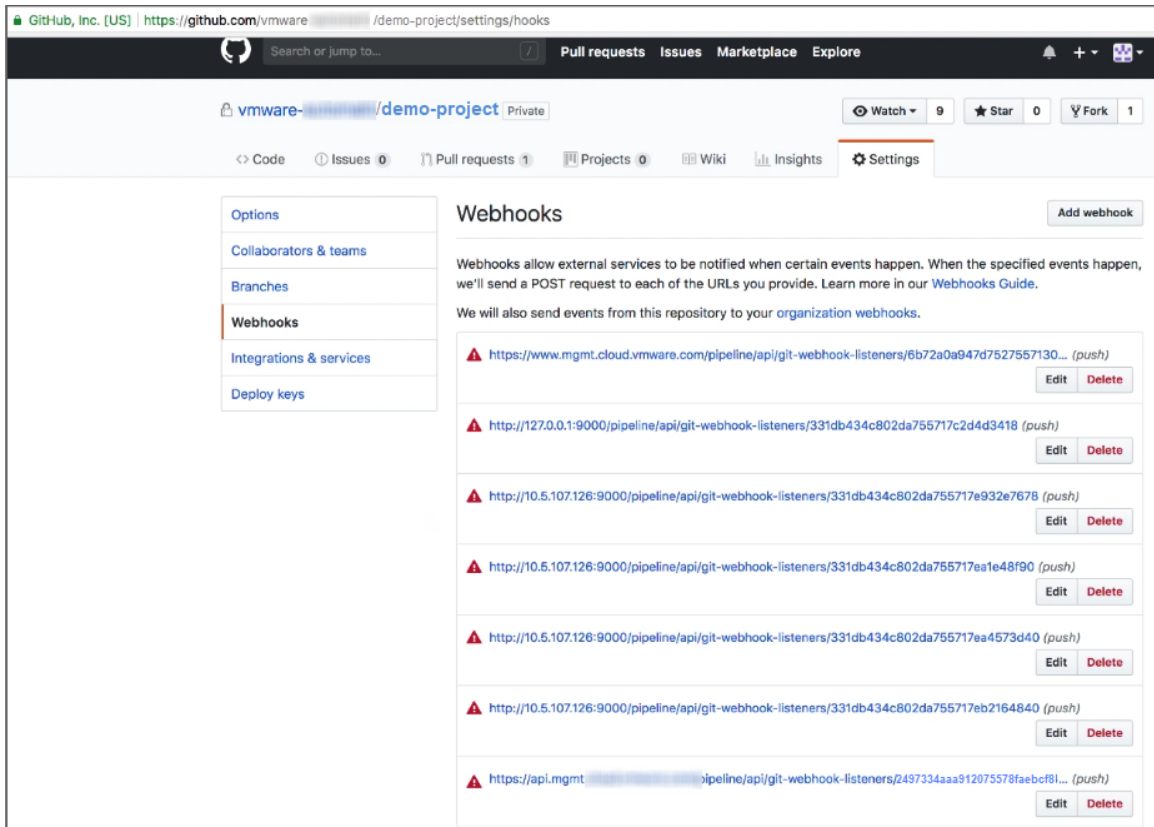
⌵

SAVE

CANCEL

- 4 Öffnen Sie in einem neuen Browserfenster das GitHub-Repository, das über den Webhook verbunden ist.
 - a Um den Webhook anzuzeigen, den Sie in vRealize Automation Code Stream hinzugefügt haben, klicken Sie auf der Registerkarte **Einstellungen** auf **Webhooks**.

Im unteren Bereich der Webhooks-Liste wird die gleiche Webhook-URL angezeigt.



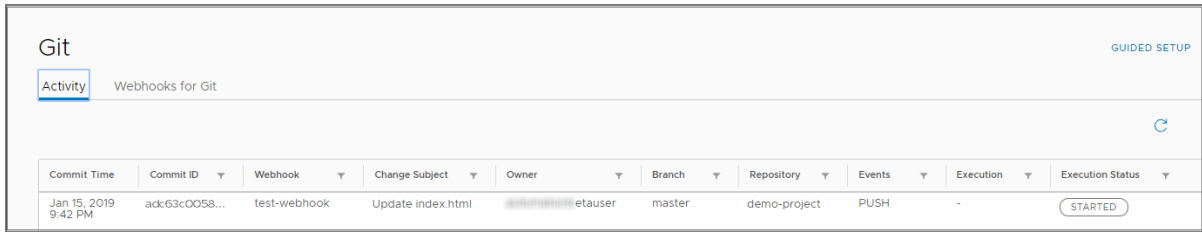
- b Wenn Sie den Code ändern möchten, klicken Sie auf die Registerkarte **Code** und wählen Sie eine Datei aus, die Sie im zu bearbeitenden Branch bearbeiten möchten. Übergeben Sie die Änderung.
 - c Um zu überprüfen, ob die Webhook-URL funktioniert, klicken Sie auf der Registerkarte **Einstellungen** erneut auf **Webhooks**.

Im unteren Bereich der Webhooks-Liste wird neben der Webhook-URL ein grünes Häkchen angezeigt.



- 5 Kehren Sie zu vRealize Automation Code Stream zurück, um die Aktivität auf dem Git-Webhook anzuzeigen. Klicken Sie auf **Auslöser > Git > Aktivität**.

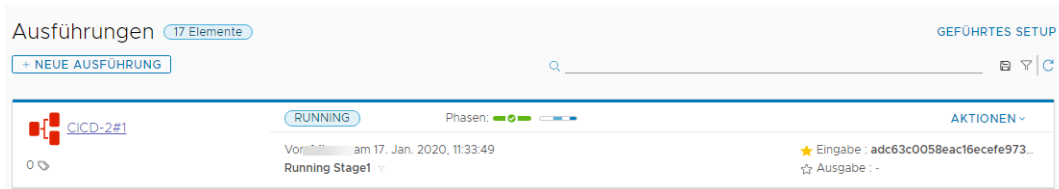
Stellen Sie unter „Ausführungsstatus“ sicher, dass die Pipeline-Ausführung gestartet wurde.



| Commit Time | Commit ID | Webhook | Change Subject | Owner | Branch | Repository | Events | Execution | Execution Status |
|----------------------|---------------|--------------|-------------------|----------|--------|--------------|--------|-----------|------------------|
| Jan 15, 2019 9:42 PM | adc63c0058... | test-webhook | Update index.html | metauser | master | demo-project | PUSH | - | STARTED |

- 6 Klicken Sie auf **Ausführungen**, um Ihre Pipeline während der Ausführung zu verfolgen.

Zum Überwachen der Pipeline-Ausführung können Sie die Schaltfläche zum Aktualisieren drücken.



| Pipeline | Status | Phasen | Aktionen |
|----------|---------|------------------------|--|
| CICD-2#1 | RUNNING | Phasen: [Progress Bar] | Eingabe: adc63c0058eac16ecef973... Ausgabe: - |

Ergebnisse

Herzlichen Glückwunsch! Sie haben erfolgreich einen Git-Auslöser zum Ausführen einer Pipeline verwendet.

Vorgehensweise zum Verwenden des Gerrit-Auslösers in vRealize Automation Code Stream zum Ausführen einer Pipeline

Als vRealize Automation Code Stream-Administrator oder -Entwickler können Sie vRealize Automation Code Stream in den Gerrit-Codeprüfungs-Lebenszyklus mithilfe des Gerrit-Auslösers integrieren. Das Ereignis löst die Ausführung einer Pipeline aus, wenn Sie einen Patch-Satz erstellen, Entwürfe veröffentlichen, Codeänderungen im Gerrit-Projekt zusammenführen oder Änderungen direkt in die Git-Verzweigung übertragen.

Wenn Sie den Auslöser für Gerrit hinzufügen, wählen Sie einen Gerrit-Listener sowie ein Gerrit-Projekt auf dem Gerrit-Server aus und konfigurieren Sie Gerrit-Ereignisse. In diesem Beispiel konfigurieren Sie zuerst einen Gerrit-Listener. Anschließend verwenden Sie diesen Listener in einem Gerrit-Auslöser mit zwei Ereignissen in drei verschiedenen Pipelines.

Voraussetzungen

- Vergewissern Sie sich, dass Sie Mitglied eines Projekts in vRealize Automation Code Stream sind. Falls nicht, bitten Sie einen vRealize Automation Code Stream-Administrator, Sie als Mitglied eines Projekts hinzuzufügen. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Hinzufügen eines Projekts in vRealize Automation Code Stream](#).
- Stellen Sie sicher, dass in vRealize Automation Code Stream ein Gerrit-Endpoint konfiguriert ist. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Integrieren von vRealize Automation Code Stream in Gerrit](#).
- Damit Pipelines ausgelöst werden, stellen Sie sicher, dass Sie die Eingabeeigenschaften der Pipeline als **Gerrit** festlegen, damit die Pipeline die Gerrit-Parameter als Eingaben empfangen kann, wenn die Pipeline ausgeführt wird.

Build and Deploy Aktiviert

Arbeitsbereich **Eingabe** Modell Ausgabe

Eingabeparameter ⓘ

Parameter automat... ☒ Gerrit ☐ Git ☐ Docker ☐ Keine

[HINZUFÜGEN](#) [EINGEFÜGTE PARAMETER HINZUFÜGEN](#) [ENTFERNEN](#)

| Mit Stern markiert ⓘ | Name |
|-------------------------------------|------------------------------|
| <input checked="" type="checkbox"/> | GERRIT_BRANCH |
| <input checked="" type="checkbox"/> | GERRIT_CHANGE_COMMIT_MESSAGE |
| <input checked="" type="checkbox"/> | GERRIT_CHANGE_FILELIST |
| <input checked="" type="checkbox"/> | GERRIT_CHANGE_ID |
| <input checked="" type="checkbox"/> | GERRIT_CHANGE_NUMBER |
| <input checked="" type="checkbox"/> | GERRIT_CHANGE_OWNER |
| <input checked="" type="checkbox"/> | GERRIT_CHANGE_OWNER_EMAIL |
| <input checked="" type="checkbox"/> | GERRIT_CHANGE_OWNER_NAME |
| <input checked="" type="checkbox"/> | GERRIT_CHANGE_OWNER_USERNAME |
| <input checked="" type="checkbox"/> | GERRIT_CHANGE_SUBJECT |

Informationen über Eingabeparameter finden Sie unter [Planen eines nativen CI/CD-Builds in vRealize Automation Code Stream](#) vor dem manuellen Hinzufügen von Aufgaben.

Verfahren

- 1 Klicken Sie in vRealize Automation Code Stream auf **Auslöser > Gerrit**.
- 2 (Optional) Klicken Sie auf die Registerkarte **Listener** und anschließend auf **Neuer Listener**.

Hinweis Wenn der Gerrit-Listener, den Sie für den Gerrit-Auslöser verwenden möchten, bereits definiert ist, überspringen Sie diesen Schritt.

- a Wählen Sie ein Projekt aus.
- b Geben Sie einen Namen für den Gerrit-Listener ein.
- c Wählen Sie einen Gerrit-Endpoint aus.

- d Geben Sie das API-Token ein.

Das CSP-API-Token authentifiziert Sie für externe API-Verbindungen mit vRealize Automation Code Stream. So rufen Sie das API-Token ab:

- 1 Klicken Sie auf **Token generieren**.
- 2 Geben Sie die E-Mail-Adresse ein, die mit Ihrem Benutzernamen und Kennwort verknüpft ist, und klicken Sie auf **Generieren**.

Das von Ihnen generierte Token ist sechs Monate lang gültig. Es wird auch als Aktualisierungstoken bezeichnet.

- Um das Token als Variable für die spätere Verwendung beizubehalten, klicken Sie auf **Variable erstellen**, geben Sie einen Namen für die Variable ein und klicken Sie auf **Speichern**.
- Um das Token als Textwert für die spätere Verwendung beizubehalten, klicken Sie auf **Kopieren** und fügen Sie das Token in eine Textdatei ein, um es lokal zu speichern.

Sie haben die Möglichkeit, eine Variable zu erstellen und das Token in einer Textdatei zur späteren Verwendung zu speichern.

- 3 Klicken Sie auf **Schließen**.

Wenn Sie eine Variable erstellt haben, zeigt das API-Token den Variablennamen an, den Sie unter Verwendung der Dollar-Bindung eingegeben haben. Wenn Sie das Token kopiert haben, zeigt das API-Token das maskierte Token an.

The screenshot shows the 'Gerrit' configuration page with the 'Listener' tab selected. The form contains the following fields and values:

- Projekt ***: test1
- Name ***: Gerrit-Demo-Listener
- Endpoint ***: corporate-gerrit
- API-Token ***: \${var.CSuser API Token } (with a green checkmark icon and a 'VARIABLE ERSTELLEN' button)

At the bottom of the form are three buttons: **ERSTELLEN**, **ÜBERPRÜFEN**, and **ABBRECHEN**.

- e Zum Überprüfen der Token- und Endpoint-Details klicken Sie auf **Überprüfen**.

Ihr Token läuft nach 90 Tagen ab.

- f Klicken Sie auf **Erstellen**.
- g Klicken Sie auf der Listener-Karte auf **Verbinden**.

Der Listener beginnt mit der Überwachung aller Aktivitäten auf dem Gerrit-Server und überwacht alle aktivierten Auslöser auf diesem Server. Um die Überwachung eines Auslösers auf diesem Server zu beenden, deaktivieren Sie den Auslöser.

Hinweis Zum Aktualisieren eines mit einem Listener verbundenen Gerrit-Endpoints müssen Sie den Listener trennen, bevor Sie den Endpoint aktualisieren.

- Klicken Sie auf **Konfigurieren > Auslöser > Gerrit**.
 - Klicken Sie auf die Registerkarte **Listener**.
 - Klicken Sie auf **Trennen** auf dem Listener, der mit dem zu aktualisierenden Endpoint verbunden ist.
-

- 3 Klicken Sie auf die Registerkarte **Auslöser** und anschließend auf **Neuer Auslöser**.

- 4 Wählen Sie ein Projekt auf dem Gerrit-Server aus.

- 5 Geben Sie einen Namen ein.

Der Name des Gerrit-Auslösers muss eindeutig sein.

- 6 Wählen Sie einen konfigurierten Gerrit-Listener aus.

vRealize Automation Code Stream verwendet den Gerrit-Listener, um eine Liste von Gerrit-Projekten bereitzustellen, die auf dem Server verfügbar sind.

- 7 Wählen Sie ein Projekt auf dem Gerrit-Server aus.

- 8 Geben Sie den Zweig im Repository ein, den der Gerrit-Listener überwachen wird.

- 9 (Optional) Geben Sie Dateieinschlüsse oder -ausschlüsse als Bedingungen für den Auslöser an.

- Sie geben Dateieinschlüsse an, die Pipelines auslösen. Wenn eine der Dateien in einem Commit mit den in den Einschlusspfaden oder dem Regex angegebenen Dateien übereinstimmt, werden die Pipelines ausgelöst. Wenn ein Regex angegeben ist, löst vRealize Automation Code Stream nur die Pipelines mit Dateinamen im Änderungssatz aus, die mit dem angegebenen Ausdruck übereinstimmen. Der Regex-Filter ist nützlich, wenn ein Auslöser für mehrere Pipelines in einem einzelnen Repository konfiguriert wird.
- Sie geben Dateiausschlüsse an, die die Auslösung von Pipelines verhindern. Wenn alle Dateien in einem Commit mit den in den Ausschlusspfaden oder dem Regex angegebenen Dateien übereinstimmt, werden die Pipelines nicht ausgelöst.
- **Priorisierung des Ausschlusses**, wenn diese Option aktiviert ist, wird sichergestellt, dass Pipelines nicht ausgelöst werden. Die Pipelines werden nicht ausgelöst, selbst wenn eine der Dateien in einem Commit mit den Dateien übereinstimmt, die in den Ausschlusspfaden oder dem Regex angegeben sind. Die Standardeinstellung für **Priorisierung des Ausschlusses** ist deaktiviert.

Wenn die Bedingungen sowohl den Dateieinschluss als auch den Dateiausschluss umfassen, werden Pipelines nicht ausgelöst.

Im folgenden Beispiel sind sowohl die Dateieinschlüsse als auch die Dateiausschlüsse Bedingungen für den Auslöser.

| Datei | | | |
|-------------------------|-------|--------------------------|-----|
| Einschlüsse | PLAIN | runtime/src/main/a.java | + |
| | REGEX | ([a-z A-Z]+/[a-z A-Z])+ | + + |
| Ausschlüsse | PLAIN | runtime/pom.xml | - |
| | PLAIN | runtime/demo.yaml | + + |
| Ausschluss priorisieren | | <input type="checkbox"/> | |

- Bei Dateieinschlüssen löst ein Commit mit einer Änderung in `runtime/src/main/a.java` oder einer beliebigen Java-Datei Pipelines aus, die in der Ereigniskonfiguration konfiguriert sind.
- Bei Dateiausschlüssen werden bei einem Commit mit Änderungen nur in beiden Dateien die in der Ereigniskonfiguration konfigurierten Pipelines nicht ausgelöst.

10 Klicken Sie auf **Neue Konfiguration**.

- Wählen Sie für ein Gerrit-Ereignis **Patch-Satz erstellt**, **Entwurf veröffentlicht** oder **Änderung zusammengeführt** aus. Oder wählen Sie für die direkte Weitergabe an Git mit Umgehung von Gerrit **Direkt an Git weitergeben** aus.
- Wählen Sie die Pipeline aus, die ausgelöst werden soll.

Wenn die Pipeline benutzerdefinierte hinzugefügte Eingabeparametern enthält, werden in der Liste der Eingabeparameter Parameter und Werte angezeigt. Sie können Werte für Eingabeparameter eingeben, die mit dem Auslöserereignis an die Pipeline übergeben werden sollen. Alternativ können Sie auf die Eingabe von Werten verzichten oder die Standardwerte verwenden.

Hinweis Wenn Standardwerte definiert sind:

- Alle für die Eingabeparameter eingegebenen Werte überschreiben die Standardwerte, die im Pipeline-Modell definiert sind.
 - Die Standardwerte in der Auslöserkonfiguration werden nicht geändert, wenn sich die Parameterwerte im Pipeline-Modell ändern.
-

Informationen zum automatischen Einfügen von Eingabeparametern für Gerrit-Auslöser finden Sie unter [Voraussetzungen](#).

- c Für **Patch-Satz erstellt**, **Entwurf veröffentlicht** und **Änderung zusammengeführt** werden einige Aktionen standardmäßig mit Beschriftungen angezeigt. Sie können die Bezeichnung ändern oder Kommentare hinzufügen. Wenn dann die Pipeline ausgeführt wird, wird die Bezeichnung oder der Kommentar auf der Registerkarte **Aktivität** als die **Durchgeführte Aktion** für die Pipeline angezeigt.
- d Klicken Sie auf **Speichern**.

Um mehrere Auslöserereignisse für mehrere Pipelines hinzuzufügen, klicken Sie erneut auf **Neue Konfiguration**.

Im folgenden Beispiel werden Ereignisse für drei Pipelines angezeigt:

- Wenn ein **Änderung zusammengeführt**-Ereignis im Gerrit-Projekt auftritt, wird die Pipeline mit der Bezeichnung **Gerrit-Pipeline** ausgelöst.
- Wenn ein **Patch-Satz erstellt**-Ereignis im Gerrit-Projekt auftritt, werden die Pipelines mit der Bezeichnung **Gerrit-Auslöser-Pipeline** und **Gerrit-Demo-Pipeline** ausgelöst.

Gerrit GEFÜHRTES SETUP

Aktivität **Auslöser** Listener

Projekt *

Name *

Gerrit-Listener *

Gerrit-Projekt *

Verzweigung *

Datei

Einschlüsse

Ausschlüsse

Ausschluss priorisieren ☐

[+ NEUE KONFIGURATION](#)

| | Ereignistyp | Pipeline | Bezeichnung |
|------------|------------------|-------------------------|-------------|
| | Change Merged | Gerrit-Pipeline | |
| | Patchset Created | Gerrit-Trigger-Pipeline | Verified |
| | Patchset Created | Gerrit-Demo-Pipeline | Verified |
| | | | |
| 3 Elemente | | | |

- 11 Klicken Sie auf **Erstellen**.

Der Gerrit-Auslöser wird auf der Registerkarte **Auslöser** als neue Karte angezeigt und ist standardmäßig als **Deaktiviert** festgelegt.

- 12 Klicken Sie auf der Auslöserkarte auf **Aktivieren**.

Nach der Aktivierung des Auslösers kann er den Gerrit-Listener verwenden, der Ereignisse überwacht, die auf dem Zweig des Gerrit-Projekts auftreten.

Um einen Auslöser zu erstellen, der dieselben Dateieinschluss- oder Dateiausschlussbedingungen hat, aber mit einem anderen Repository als dem, das Sie beim Erstellen des Auslösers verwendet haben, klicken Sie auf der Auslöserkarte auf **Aktionen > Klonen**. Klicken Sie dann im geklonten Auslöser auf **Öffnen** und ändern Sie die Parameter.

Ergebnisse

Herzlichen Glückwunsch! Sie haben erfolgreich einen Gerrit-Auslöser mit zwei Ereignissen in drei verschiedenen Pipelines konfiguriert.

Nächste Schritte

Nachdem Sie eine Codeänderung im Gerrit-Projekt ausgeführt haben, überprüfen Sie die Registerkarte **Aktivität** für das Gerrit-Ereignis in vRealize Automation Code Stream. Stellen Sie sicher, dass die Liste der Aktivitäten Einträge enthält, die jeder Pipeline-Ausführung in der Auslöserkonfiguration entsprechen.

Wenn ein Ereignis auftritt, können nur Pipelines im Gerrit-Auslöser ausgeführt werden, die sich auf den jeweiligen Ereignistyp beziehen. Wenn in diesem Beispiel ein Patch-Satz erstellt wird, werden nur die **Gerrit-Auslöser-Pipeline** und die **Gerrit-Demo-Pipeline** ausgeführt.

Mit den Informationen in den Spalten auf der Registerkarte **Aktivität** wird jedes Gerrit-Auslöserereignis beschrieben. Sie können die angezeigten Spalten auswählen, indem Sie auf das Spaltensymbol klicken, das unterhalb der Tabelle angezeigt wird.

- Die Spalten **Betreff ändern** und **Ausführung** sind leer, wenn der Auslöser eine direkte Git-Weitergabe war.
- Die Spalte **Gerrit-Auslöser** zeigt den Auslöser an, der das Ereignis erstellt hat.
- Die Spalte **Listener** ist standardmäßig deaktiviert. Wenn Sie sie auswählen, wird in der Spalte der Gerrit-Listener angezeigt, der das Ereignis empfangen hat. Ein einzelner Listener kann als mehreren Auslösern zugeordnet angezeigt werden.
- Die Spalte **Auslösertyp** ist standardmäßig deaktiviert. Wenn Sie diese Option auswählen, wird in der Spalte der Auslösertyp als AUTOMATISCH oder MANUELL angezeigt.
- Weitere Spalten sind **Commit-Zeit**, **Ändern#**, **Status**, **Nachricht**, **Durchgeführte Aktion**, **Benutzer**, **Gerrit-Projekt**, **Verzweigung** und **Ereignis**.

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: upryam@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: upryam@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: upryam@vm | test1 | Gerrit-Demo-Trigger | master | Patchset created |
| ⋮ | | | 1111Dummy | Gerrit-Trigger-Pipeline #1 | WAITING | Stage0.Task0: Execution Waiting for User Action. | | Orlando: upryam@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

Um die Aktivität für eine abgeschlossene oder fehlgeschlagene Pipeline-Ausführung zu steuern, klicken Sie auf die drei Punkte links neben jedem Eintrag auf dem Bildschirm „Aktivität“.

- Wenn die Pipeline aufgrund eines Fehlers im Pipeline-Modell oder eines anderen Problems nicht ausgeführt werden kann, korrigieren Sie den Fehler und wählen Sie **Erneut ausführen** aus, um die Pipeline erneut auszuführen.
- Wenn die Pipeline aufgrund eines Problems mit der Netzwerkkonnektivität oder eines anderen Problems nicht ausgeführt werden kann, wählen Sie **Fortsetzen** aus, wodurch die Ausführung derselben Pipeline neu gestartet und Laufzeit eingespart wird.
- Verwenden Sie **Ausführung anzeigen**, wodurch die Pipeline-Ausführungsansicht geöffnet wird. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Ausführen einer Pipeline und Anzeigen von Ergebnissen](#).
- Verwenden Sie die Option **Löschen**, um den Eintrag aus dem Aktivitätsbildschirm zu löschen.

Wenn ein Gerrit-Ereignis eine Pipeline nicht auslösen kann, klicken Sie auf **Manuell auslösen**, wählen Sie dann den Gerrit-Auslöser aus, geben Sie die Änderungs-ID ein und klicken Sie auf **Ausführen**.

Überwachen von Pipelines in vRealize Automation Code Stream



Als vRealize Automation Code Stream-Administrator oder Entwickler benötigen Sie Einblick in die Leistung Ihrer Pipelines in vRealize Automation Code Stream. Sie müssen wissen, wie effektiv Ihre Pipelines Code von der Entwicklung, über Tests und die Produktion freigibt.

Um Einblick zu erhalten, verwenden Sie vRealize Automation Code Stream-Dashboards, um die Trends und Ergebnisse einer Pipeline-Ausführung zu überwachen. Sie können die Standard-Pipeline-Dashboards verwenden, um eine einzelne Pipeline zu überwachen oder benutzerdefinierte Dashboards zum Überwachen mehrerer Pipelines zu erstellen.

- Pipeline-Metriken enthalten Statistiken wie die mittleren Zeiten, die im Pipeline-Dashboard verfügbar sind.
- Wenn Sie Metriken für mehrere Pipelines anzeigen möchten, verwenden Sie die benutzerdefinierten Dashboards.

Dieses Kapitel enthält die folgenden Themen:

- [Überblick über das Pipeline-Dashboard in vRealize Automation Code Stream](#)
- [Vorgehensweise zum Verwenden benutzerdefinierter Dashboards zum Verfolgen von wichtigen Leistungsindikatoren für meine Pipeline in vRealize Automation Code Stream](#)

Überblick über das Pipeline-Dashboard in vRealize Automation Code Stream

Ein Pipeline-Dashboard ist eine Ansicht der Ergebnisse für eine bestimmte Pipeline, die ausgeführt wurde, zum Beispiel Trends, Top-Fehler und erfolgreiche Änderungen. vRealize Automation Code Stream erstellt das Pipeline-Dashboard, wenn Sie eine Pipeline erstellen.

Das Dashboard enthält die Widgets, die die Ergebnisse der Pipeline-Ausführung anzeigen.

Widget „Anzahl der Pipeline-Ausführungsstatus“

Sie können die Gesamtanzahl der Ausführungen einer Pipeline über einen bestimmten Zeitraum anzeigen, die nach folgenden Status gruppiert sind: „Abgeschlossen“, „Fehlgeschlagen“ oder „Abgebrochen“. Um zu sehen, wie sich der Pipeline-Ausführungsstatus über einen längeren oder kürzeren Zeitraum hinweg geändert hat, ändern Sie die Anzeigedauer.

Widget „Pipeline-Ausführungsstatistik“

Die Statistik der Pipeline-Ausführung enthält die mittleren Zeiten für die Wiederherstellung, die Bereitstellung oder das Fehlschlagen einer Pipeline im Laufe der Zeit.

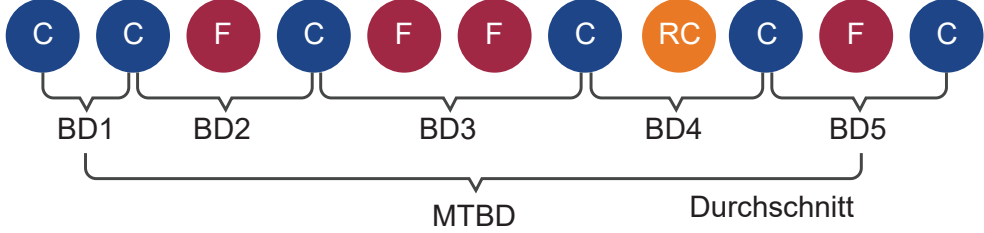
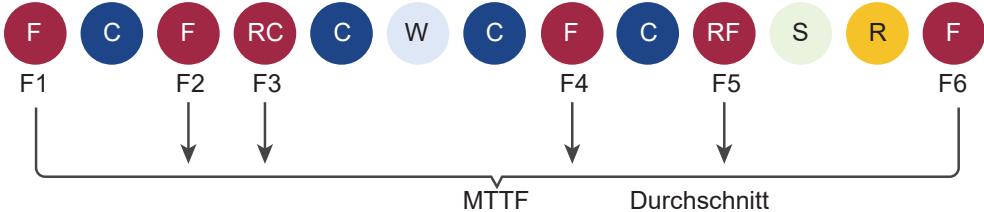
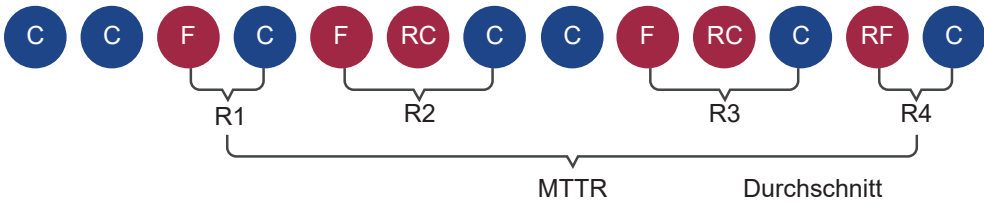
Die folgenden Zustände gelten für alle Pipeline-Ausführungen:

- Abgeschlossen
- Fehlgeschlagen
- Im Wartezustand
- Wird ausgeführt
- Abgebrochen
- In der Warteschlange
- Nicht gestartet
- Rollback wird durchgeführt
- Rollback abgeschlossen
- Rollback fehlgeschlagen
- Angehalten

Tabelle 8-1. Messen der mittleren Zeiten

| Gemessenes Element | Bedeutung |
|---|---|
| Durchschnittliche CI | Durchschnittliche Zeit, die in der Phase der kontinuierlichen Integration verbracht wird, gemessen anhand der Zeit im CI-Aufgabentyp. |
| Mittlere Zeit bis zur Bereitstellung (MTTD) | Durchschnittliche Dauer aller ABGESCHLOSSENEN Ausführungen über einen bestimmten Zeitraum. D1, D2 usw. ist die Zeitdauer für die Bereitstellung jeder ABGESCHLOSSENEN Ausführung. |

Tabelle 8-1. Messen der mittleren Zeiten (Fortsetzung)

| Gemessenes Element | Bedeutung |
|--|---|
| Mittlere Zeit zwischen Bereitstellungen (MTBD) | <p>Durchschnittliche Zeit, die zwischen erfolgreichen Bereitstellungen über einen bestimmten Zeitraum verstrichen ist. Die Zeit, die zwischen zwei aufeinanderfolgenden ABGESCHLOSSENEN Ausführungen verstrichen ist, ist die Zeit zwischen erfolgreichen Bereitstellungen, z. B. BD1, BD2 usw. MTBD gibt an, wie oft eine Produktionsumgebung aktualisiert wird.</p>  |
| Mittlere Zeit bis zum Ausfall (MTTF) | <p>Durchschnittliche Dauer von Ausführungen, die mit den Status FEHLGESCHLAGEN, ROLLBACK ABGESCHLOSSEN oder ROLLBACK FEHLGESCHLAGEN über einen bestimmten Zeitraum enden. F1, F2 usw. ist die Zeitdauer, in der eine Ausführung mit den Status FEHLGESCHLAGEN, ROLLBACK ABGESCHLOSSEN oder ROLLBACK FEHLGESCHLAGEN endet.</p>  |
| Mittlere Zeit bis zur Wiederherstellung (MTTR) | <p>Durchschnittliche Zeit bis zur Wiederherstellung nach einem Fehler über einen bestimmten Zeitraum. Die Zeit bis zur Wiederherstellung nach einem Fehler ist die Zeit, die zwischen einer Ausführung mit dem endgültigen Status FEHLGESCHLAGEN, ROLLBACK ABGESCHLOSSEN oder ROLLBACK FEHLGESCHLAGEN und der nächsten sofortigen erfolgreichen Ausführung mit dem Status ABGESCHLOSSEN verstrichen ist. R1, R2 usw. gibt die Zeitdauer für die Wiederherstellung nach jeder Ausführung mit dem Status FEHLGESCHLAGEN oder ROLLBACK FEHLGESCHLAGEN an.</p>  |

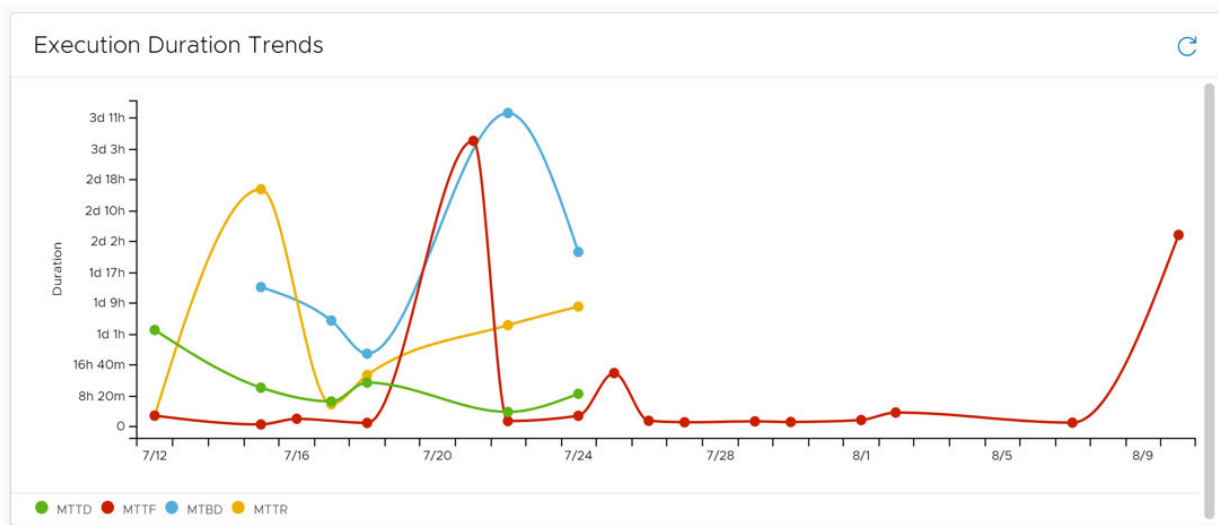
Widgets „Wichtigste fehlgeschlagene Phasen“ und „Aufgaben“

Zwei Widgets zeigen die wichtigsten fehlgeschlagenen Phasen und Aufgaben in einer Pipeline an. Jede Messung gibt die Anzahl und der Prozentsatz der Fehler für Entwicklungs- und Post-Entwicklungsumgebungen für jede Pipeline und jedes Projekt an, die für eine Woche oder einen Monat gemittelt wurden. Sie sehen dann die wichtigsten Fehler bei der Behebung von Problemen im Prozess zur Automatisierung der Freigabe.

Sie können beispielsweise die Anzeige für eine bestimmte Dauer konfigurieren, wie die letzten sieben Tage, und die wichtigsten fehlgeschlagenen Aufgaben während dieses Zeitraums vermerken. Wenn Sie eine Änderung in Ihrer Umgebung oder Pipeline vornehmen und die Pipeline erneut ausführen, sollten Sie die wichtigsten fehlgeschlagenen Aufgaben über eine längere Dauer überprüfen, beispielsweise für die letzten 14 Tage. Möglicherweise haben sich die wichtigsten fehlgeschlagenen Aufgaben geändert. An diesem Ergebnis können Sie ablesen, dass die Änderung in Ihrem Prozess zur Automatisierung der Freigabe die Erfolgsrate Ihrer Pipeline-Ausführung verbessert hat.

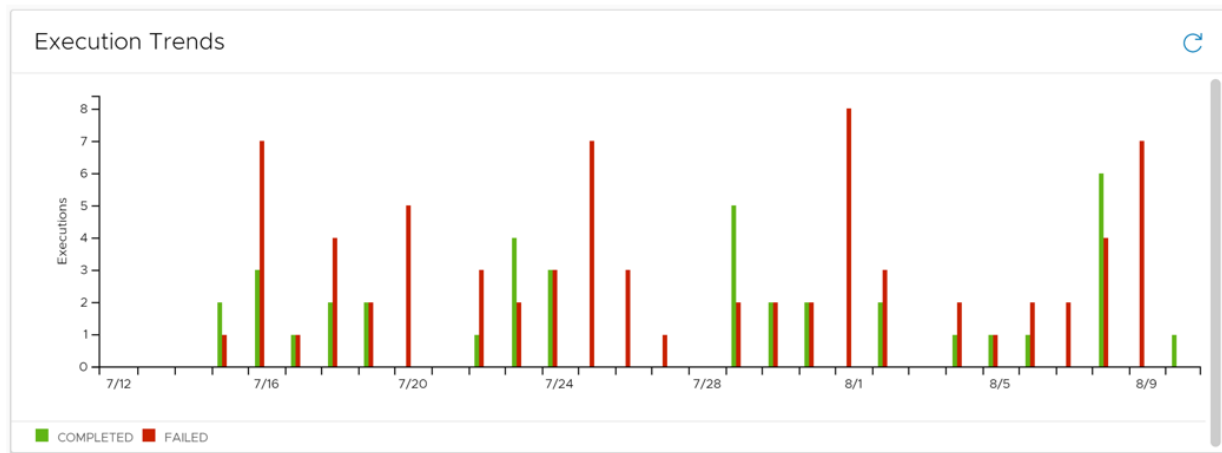
Widget „Trends bei der Pipeline-Ausführungsdauer“

Trends der Pipeline-Ausführungsdauer zeigen MTDD, MTTF, MTBD und MTTR über einen bestimmten Zeitraum an.



Widget „Pipeline-Ausführungstrends“

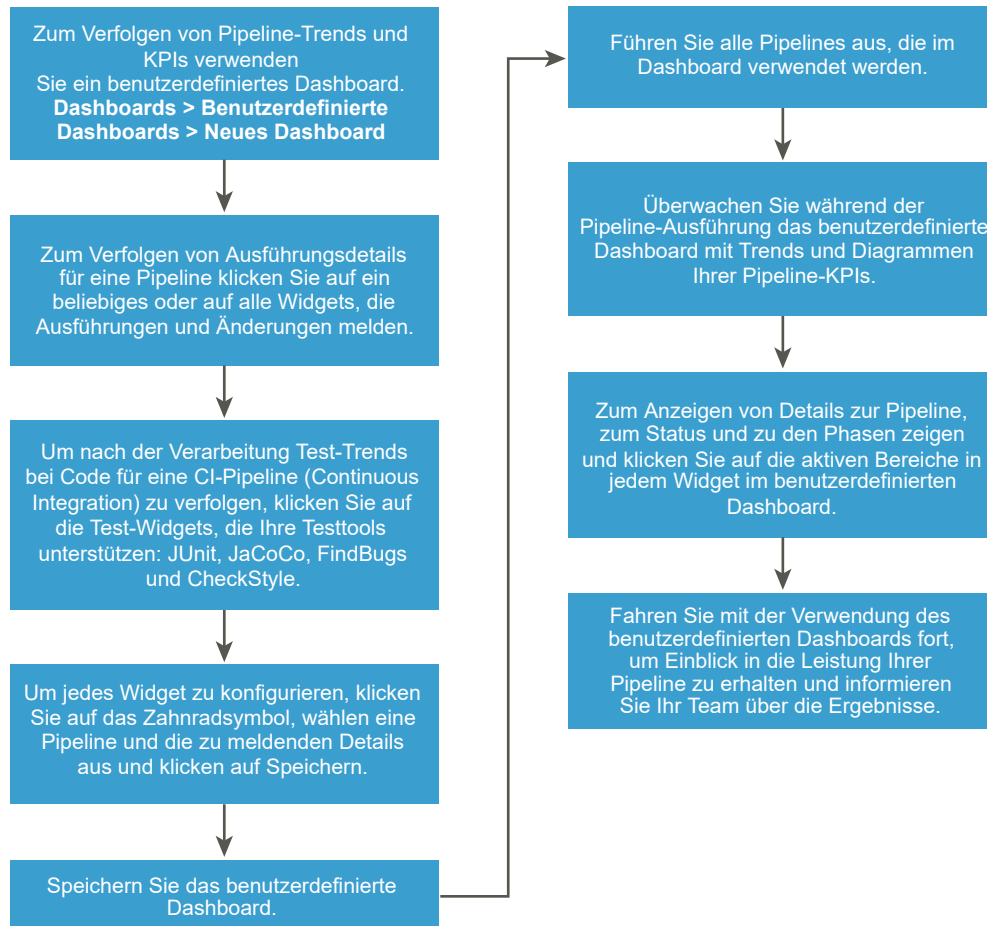
Pipeline-Ausführungstrends zeigen die gesamten täglichen Ausführungen einer Pipeline an, nach Status über einen bestimmten Zeitraum gruppiert. Mit Ausnahme des aktuellen Tages zeigen die meisten täglichen Aggregationszahlen nur die Ausführungen ABGESCHLOSSEN und FEHLGESCHLAGEN an.



Vorgehensweise zum Verwenden benutzerdefinierter Dashboards zum Verfolgen von wichtigen Leistungsindikatoren für meine Pipeline in vRealize Automation Code Stream

Als vRealize Automation Code Stream-Administrator oder -Entwickler erstellen Sie das benutzerdefinierte Dashboard, um die Ergebnisse anzuzeigen, die Sie für eine oder mehrere ausgeführte Pipelines anzeigen möchten. Sie können zum Beispiel ein projektweites Dashboard mit KPIs und Metriken erstellen, die aus mehreren Pipelines erfasst werden. Wenn eine Ausführungswarnung oder ein Fehler gemeldet wird, können Sie das Dashboard verwenden, um den Fehler zu beheben.

Um Trends und wichtige Leistungsindikatoren für Ihre Pipelines mithilfe eines benutzerdefinierten Dashboards nachzuverfolgen, fügen Sie dem Dashboard Widgets hinzu und konfigurieren Sie sie so, dass sie über Ihre Pipelines berichten.



Voraussetzungen

- Stellen Sie sicher, dass eine oder mehrere Pipelines vorhanden sind. Klicken Sie auf der Benutzeroberfläche auf **Pipelines**.
- Stellen Sie für die zu überwachenden Pipelines sicher, dass sie erfolgreich ausgeführt wurden. Klicken Sie auf **Ausführungen**.

Verfahren

- 1 Um ein benutzerdefiniertes Dashboard zu erstellen, klicken Sie auf **Dashboards > Benutzerdefinierte Dashboards > Neues Dashboard**.

- 2 Um das Dashboard so anzupassen, dass es über bestimmte Trends und wichtige Leistungsindikatoren für Ihre Pipeline berichtet, klicken Sie auf ein Widget.

Um beispielsweise Details über den Pipeline-Status, die Phasen, die Aufgaben, die Ausführungsdauer und den Benutzer anzuzeigen, der Sie ausgeführt hat, klicken Sie auf das Widget **Ausführungsdetails**. Bei einer CI-Pipeline können Sie die Trends bei der Nachbearbeitung mithilfe der Widgets für JUnit, JaCoCo, FindBugs und CheckStyle verfolgen.

| Ausführung# | Status | Statusmeldung | Alle Aufgaben | TaskID (StageID) | Dauer |
|-------------|-----------|--|---------------|------------------|------------------------------------|
| #22 | WAITING | Stage0.Task0: Execution Waiting for User Action. | | | 2 Stunden, 34 Minuten, 25 Sekunden |
| #21 | COMPLETED | Execution Completed. | | | 17 Sekunden |

Elemente pro Seite: 10 | 1 - 10 von 22 Elementen

- 3 Konfigurieren Sie jedes Widget, das Sie hinzufügen.
 - a Klicken Sie im Widget auf das Zahnradsymbol.
 - b Wählen Sie eine Pipeline aus, legen Sie die verfügbaren Optionen fest und wählen Sie die anzuzeigenden Spalten aus.
 - c Klicken Sie zum Speichern der Widget-Konfiguration auf **Speichern**.
 - d Um das benutzerdefinierte Dashboard zu speichern, klicken Sie auf **Speichern** und dann auf **Schließen**.
- 4 Um weitere Informationen zur Pipeline anzuzeigen, klicken Sie auf die aktiven Bereiche auf den Widgets.

Klicken Sie beispielsweise im Widget **Ausführungsdetails** auf einen Eintrag in der Spalte „Status“, um weitere Informationen zur Pipeline-Ausführung anzuzeigen. Klicken Sie alternativ dazu im Widget **Letzte erfolgreiche Änderung** auf den aktiven Link, um eine Zusammenfassung des Pipeline-Status und der Aufgabe anzuzeigen.

Ergebnisse

Herzlichen Glückwunsch! Sie haben ein benutzerdefiniertes Dashboard erstellt, das Trends und KPIs für Ihre Pipelines überwacht.

Nächste Schritte

Überwachen Sie weiterhin die Leistung Ihrer Pipelines in vRealize Automation Code Stream und teilen Sie die Ergebnisse mit Ihren Vorgesetzten und Teams, um den Prozess zur Freigabe Ihrer Anwendungen weiter zu verbessern.

Weitere Informationen zu Code Stream

9

Es gibt viele Möglichkeiten für vRealize Automation Code Stream-Administratoren und Entwickler, mehr über vRealize Automation Code Stream und dessen Verwendungszweck zu erfahren.

Sie können diese Dokumentation verwenden, um weitere Informationen zu Pipelines und deren Ausführungen, zum Hinzufügen von Endpoints und Projekten usw. zu erhalten.

Sie machen sich mit den von Rollen bereitgestellten Berechtigungen vertraut. Sie erhalten Informationen zur Verwendung beschränkter Ressourcen sowie zur Anforderung von Genehmigungen für Pipelines. Weitere Informationen hierzu finden Sie unter [Vorgehensweise zum Verwalten des Benutzerzugriffs und der Genehmigungen in vRealize Automation Code Stream](#).

Nutzen Sie die Vorzüge der Suchfunktion, indem Sie herausfinden, wo sich bestimmte Aufträge oder Komponenten in Ihren Pipelines, Ausführungen oder Endpoints befinden.

Dieses Kapitel enthält die folgenden Themen:

- [Definition der Suchfunktion in vRealize Automation Code Stream](#)
- [Weitere Ressourcen für vRealize Automation Code Stream-Administratoren und -Entwickler](#)

Definition der Suchfunktion in vRealize Automation Code Stream

Sie verwenden die Suchfunktion, um herauszufinden, wo sich bestimmte Elemente oder Komponenten befinden. Sie möchten gegebenenfalls nach aktivierten oder deaktivierten Pipelines suchen, da deaktivierte Pipelines beispielsweise nicht ausgeführt werden können.

Elemente, nach denen gesucht werden kann

Durchsuchbare Elemente:

- Projekte
- Endpoints
- Pipelines
- Ausführungen
- Pipeline-Dashboards, benutzerdefinierte Dashboards

- Gerrit-Auslöser und -Server
- Git-Webhooks
- Docker-Webhooks

Die Suche mit spaltenbasierten Filtern kann durchgeführt werden in:

- Benutzervorgängen
- Variablen
- Aktivität für Gerrit, Git und Docker auslösen

Sie können die Suche mit spaltenbasierten Filtern auf der Seite **Aktivität** für jeden Auslöser durchführen.

Funktionsweise der Suche

Die Suchkriterien variieren je nach der Seite, auf der Sie sich befinden. Jede Seite verfügt über unterschiedliche Suchkriterien.

| Suchorte | Bei der Suche zu verwendende Kriterien |
|-------------------------------|--|
| Pipeline-Dashboards | Projekt, Name, Beschreibung, Tags, Verknüpfung |
| Benutzerdefinierte Dashboards | Projekt, Name, Beschreibung, Verknüpfung (UUID eines Elements auf dem Dashboard) |
| Ausführungen | Name, Kommentare, Grund, Tags, Index, Status, Projekt, Anzeigen, Ausgeführt von, Von mir ausgeführt, Link (UUID der Ausführung) und Eingabeparameter, Ausgabeparameter oder Statusmeldung unter Verwendung dieses Formats: <key>:<value> |
| Pipelines | Name, Beschreibung, Zustand, Tags, Erstellt von, Von mir erstellt, Aktualisiert von, Von mir aktualisiert, Projekt |
| Projekte | Name, Beschreibung |
| Endpoints | Name, Beschreibung, Typ, Aktualisiert von, Projekt |
| Gerrit-Auslöser | Name, Status, Projekt |
| Gerrit-Server | Name, Server-URL, Projekt |
| Git-Webhooks | Name, Servertyp, Repository, Branch, Projekt |

Dabei gilt:

- Verknüpfung ist die UUID einer Pipeline, einer Ausführung oder eines Widgets in einem Dashboard.
- Eingabeparameter, Ausgabeparameter und Statusmeldungsnotation und Beispiele sind:
 - Notation: `input.<inputKey>:<inputValue>`
Beispiel: `input.GERRIT_CHANGE_OWNER_EMAIL:joe_user`
 - Notation: `output.<outputKey>:<outputValue>`

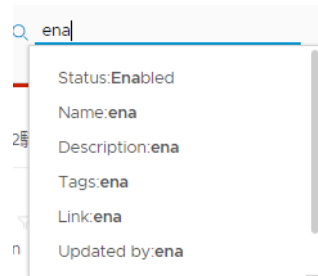
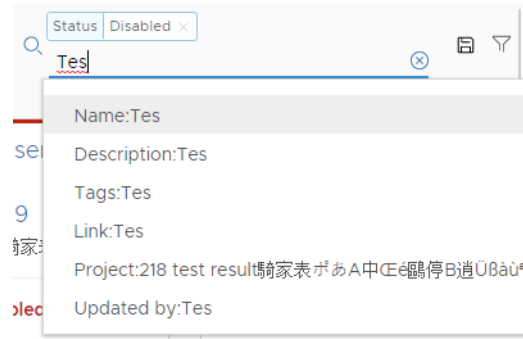
Beispiel: **output.BuildNo:29**

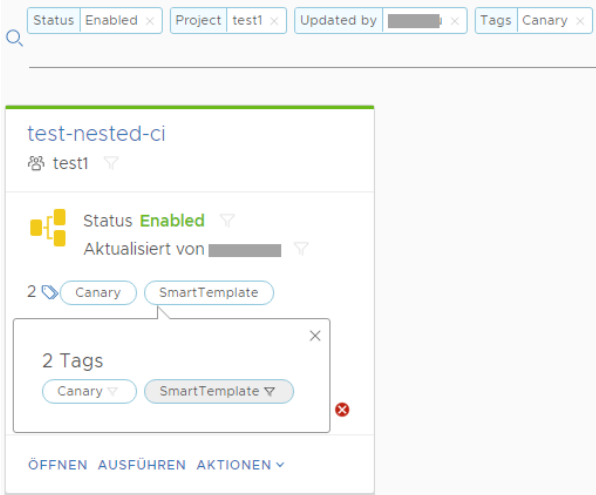
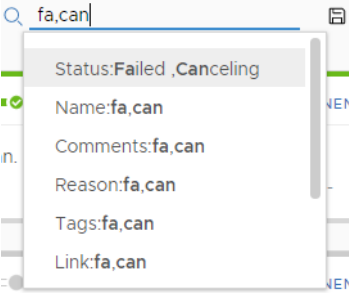
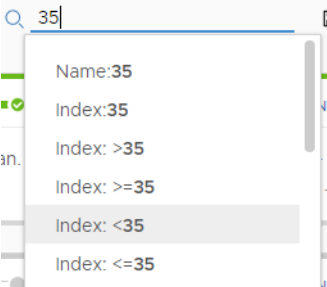
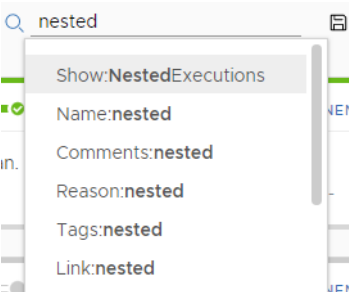
- Notation: `statusMessage:<value>`

Beispiel: **statusMessage:Execution failed**

- Der Status oder Zustand richtet sich nach der Suchseite.
 - Für Ausführungen sind folgende Werte möglich: „Abgeschlossen“, „Fehlgeschlagen“, „Rollback fehlgeschlagen“ oder „Abgebrochen“.
 - Zu den möglichen Zustandswerten für Pipelines gehören: aktiviert, deaktiviert oder freigegeben.
 - Zu den möglichen Statuswerten für Auslöser gehören: aktiviert oder deaktiviert.
- „Ausgeführt“, „Erstellt“ oder „Vor mir aktualisiert“ bezieht sich auf den angemeldeten Benutzer.

Das Suchfeld wird oben rechts auf jeder gültigen Seite angezeigt. Wenn Sie mit der Eingabe im leeren Suchfeld beginnen, erkennt vRealize Automation Code Stream den Kontext der Seite und schlägt Optionen für die Suche vor.

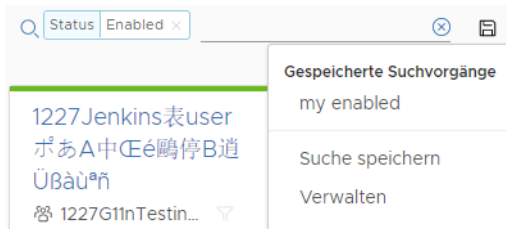
| Bei der Suche zu verwendende Methoden | Eingabemethode |
|--|--|
| <p>Geben Sie einen Teil des Suchparameters ein.</p> <p>Geben Sie beispielsweise ena ein, um einen Statusfilter hinzuzufügen, der alle aktivierten Pipelines auflistet.</p> |  |
| <p>Um die Anzahl der gefundenen Elemente zu reduzieren, fügen Sie einen Filter hinzu.</p> <p>Geben Sie beispielsweise tes ein, um einen Namensfilter hinzuzufügen. Der Filter funktioniert wie AND mit dem vorhandenen Filter Status: deaktiviert, um nur die deaktivierten Pipelines mit tes im Namen anzuzeigen.</p> <p>Wenn Sie einen weiteren Filter hinzufügen, werden die folgenden Optionen angezeigt: Name, Beschreibung, Tags, Link, Projekt und Aktualisiert von.</p> |  |

| Bei der Suche zu verwendende Methoden | Eingabemethode |
|---|--|
| <p>Um die Anzahl der angezeigten Elemente zu reduzieren, klicken Sie auf das Filtersymbol für Eigenschaften einer Pipeline oder einer Pipeline-Ausführung.</p> <ul style="list-style-type: none"> Für Pipelines haben Status, Tags, Projekt und Aktualisiert von jeweils ein Filtersymbol. Für Ausführungen haben Tags, Ausgeführt von und Statusmeldung jeweils ein Filtersymbol. <p>Klicken Sie beispielsweise auf der Pipeline-Karte auf das Symbol, um den Filter für das SmartTemplate-Tag zu den vorhandenen Filtern hinzuzufügen für: Status:Enabled, Project:test, Updated by:user und Tags:Canary.</p> |  |
| <p>Verwenden Sie ein Kommatrennzeichen, um alle Elemente in zwei Ausführungszuständen einzubeziehen.</p> <p>Geben Sie beispielsweise fa,can zum Erstellen eines Statusfilters ein, der als OR fungiert, um alle fehlgeschlagenen oder abgebrochenen Ausführungen aufzulisten</p> |  |
| <p>Geben Sie eine Zahl ein, um alle Elemente in einem Indexbereich einzubeziehen.</p> <p>Geben Sie beispielsweise 35 ein und wählen Sie < aus, um alle Ausführungen mit einem Indexwert kleiner als 35 aufzulisten.</p> |  |
| <p>Als Aufgaben modellierte Pipelines werden zu verschachtelten Ausführungen und werden standardmäßig nicht mit allen Ausführungen aufgelistet.</p> <p>Zur Anzeige verschachtelter Ausführungen geben Sie Verschachtelt ein und wählen den Filter Anzeigen aus.</p> |  |

Speichern einer Favoritensuche

Sie können die Favoritensuche zur Verwendung auf jeder Seite speichern, indem Sie neben dem Suchbereich auf das Festplattensymbol klicken.

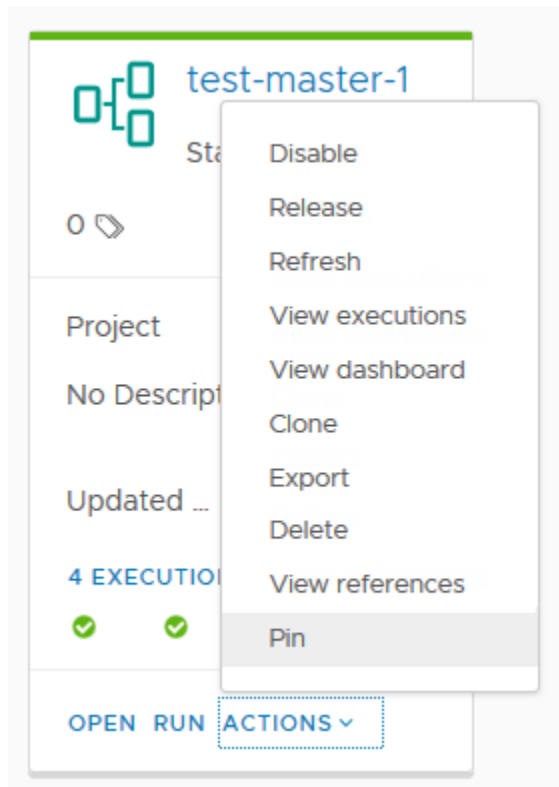
- Sie speichern eine Suche, indem Sie die Parameter für die Suche eingeben und auf das Symbol klicken, um der Suche einen Namen (z. B. **my enabled**) zu geben.
- Nach dem Speichern einer Suche klicken Sie auf das Symbol, um auf die Suche zuzugreifen. Sie können auch **Verwalten** auswählen, um die Suche in der Liste der gespeicherten Suchläufe umzubenennen, zu löschen oder zu verschieben.



Suchläufe sind an Ihren Benutzernamen gebunden und werden nur auf den Seiten angezeigt, für die die Suche gilt. Wenn Sie beispielsweise einen Suchlauf mit der Bezeichnung **my enabled** für **Status: aktiviert** auf der Seite „Pipelines“ gespeichert haben, steht der Suchlauf mit der Bezeichnung **my enabled** auf der Seite „Gerrit-Auslöser“ nicht zur Verfügung, obwohl **Status: aktiviert** ein gültiger Suchlauf für einen Auslöser ist.

Speichern einer bevorzugten Pipeline

Wenn Sie über eine bevorzugte Pipeline oder ein Dashboard verfügen, können Sie diese(s) so fixieren, dass sie bzw. es immer oben auf der Seite „Pipelines“ oder „Dashboards“ angezeigt wird. Klicken Sie auf der Pipeline-Karte auf **Aktionen > Anheften**.



Weitere Ressourcen für vRealize Automation Code Stream-Administratoren und -Entwickler

Als vRealize Automation Code Stream-Administrator oder Entwickler erhalten Sie weitere Informationen zu vRealize Automation Code Stream.

Tabelle 9-1. Weitere Ressourcen für Administratoren

| Weitere Informationen über ... | Bieten die folgenden Ressourcen... |
|--|---|
| <p>Andere Möglichkeiten zur Verwendung von vRealize Automation Code Stream durch Administratoren:</p> <ul style="list-style-type: none"> ■ Konfiguration der Pipelines zur Automatisierung der Tests und der Veröffentlichung von nativen Cloud-Anwendungen ■ Automatisierung und Tests des Quellcodes von Entwicklern – vom Test bis zur Produktion ■ Konfiguration der Pipelines für Entwickler, um Änderungen zu testen, bevor diese an die primären Verzweigung übergeben werden ■ Erfassung wichtiger Pipeline-Metriken | <p>vRealize Automation Code Stream</p> <ul style="list-style-type: none"> ■ Dokumentation zu vRealize Automation ■ vRealize Automation-Produkt-Webseite <p>VMware interaktiv</p> <ul style="list-style-type: none"> ■ Verwenden Sie die vRealize Automation Community. ■ Verwenden der VMware Learning Zone ■ Durchsuchen der VMware-Blogs. ■ Durchführen der praktischen Übungen von VMware. |

Tabelle 9-2. Weitere Ressourcen für Entwickler

| Weitere Informationen über ... | Bieten die folgenden Ressourcen... |
|---|---|
| <p>Andere Möglichkeiten, wie Entwickler vRealize Automation Code Stream verwenden können:</p> <ul style="list-style-type: none"> ■ Verwendung öffentlicher und privater Registrierungs-Images, um Umgebungen für neue Anwendungen oder Dienste zu erstellen ■ Einrichtung von Entwicklungsumgebungen, sodass Sie Verzweigungen aus dem neuesten stabilen Build erstellen können ■ Aktualisierung der Entwicklungsumgebungen mit den neuesten Codeänderungen und Artefakten ■ Testen nicht übergebener Codeänderungen für die neuesten stabilen Builds anderer abhängiger Dienste ■ Erhalten einer Benachrichtigung, wenn eine Änderung, die an eine primäre CI/CD-Pipeline übergeben wurde, andere Dienste unterbricht | <p>vRealize Automation Code Stream</p> <ul style="list-style-type: none"> ■ Dokumentation zu vRealize Automation ■ vRealize Automation-Produkt-Webseite <p>VMware interaktiv</p> <ul style="list-style-type: none"> ■ Verwenden Sie die vRealize Automation Community. ■ Verwenden der VMware Learning Zone ■ Durchsuchen der VMware-Blogs. ■ Durchführen der praktischen Übungen von VMware. |