

# Utilisation et gestion de vRealize Automation Code Stream

14 décembre 2022

vRealize Automation 8.2

Vous trouverez la documentation technique la plus récente sur le site Web de VMware, à l'adresse :

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

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

**VMware France SAS.**  
Tour Franklin  
100-101 Terrasse Boieldieu  
92042 Paris La Défense 8 Cedex  
France  
[www.vmware.com/fr](http://www.vmware.com/fr)

Copyright © 2022 VMware, Inc. Tous droits réservés. [Informations relatives aux copyrights et marques commerciales.](#)

# Table des matières

<b>1</b>	<b>Présentation et fonctionnement de vRealize Automation Code Stream</b>	<b>5</b>
<b>2</b>	<b>Configuration pour la modélisation du processus de publication</b>	<b>11</b>
	Ajout d'un projet	15
	Gestion de l'accès et des approbations utilisateur	16
	Opérations d'utilisateur et approbations	25
<b>3</b>	<b>Création et utilisation de pipelines</b>	<b>28</b>
	Comment exécuter un pipeline et afficher les résultats	31
	Types de tâche disponibles	36
	Utilisation des liaisons de variables dans les pipelines	41
	Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline	51
	Quelles variables et expressions peuvent être utilisées lors de la liaison de tâches de pipeline	53
	Comment puis-je envoyer des notifications sur mon pipeline	72
	Création d'un ticket JIRA en cas d'échec d'une tâche de pipeline	75
	Comment restaurer un déploiement	78
<b>4</b>	<b>Planification de la génération, de l'intégration et de la livraison de votre code en mode natif</b>	<b>84</b>
	Planification d'une build native CICD avant d'utiliser le modèle de pipeline intelligent	84
	Planification d'une build native CI avant d'utiliser le modèle de pipeline intelligent	89
	Planification d'une build native CD avant d'utiliser le modèle de pipeline intelligent	90
	Planification d'une build native CICD avant ajout de tâches manuel	91
	Planification de la restauration	96
<b>5</b>	<b>Didacticiels</b>	<b>99</b>
	Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline	100
	Automatisation de la publication d'une application déployée à partir d'un modèle de cloud YAML	106
	Automatisation de la publication d'une application sur un cluster Kubernetes	113
	Déploiement d'une application vers un déploiement Bleu-vert	121
	Intégration d'outils de génération, de test et de déploiement personnalisés	126
	Utilisation d'une API REST pour l'intégration avec d'autres applications	136
<b>6</b>	<b>Connexion à des points de terminaison</b>	<b>140</b>
	Présentation des points de terminaison	140
	Intégration à Jenkins	142

Intégration à Git 148

Intégration à Gerrit 151

Intégration à vRealize Orchestrator 153

## **7 Déclenchement de pipelines 158**

Utilisation du déclencheur Docker pour exécuter un pipeline de livraison continue 158

Utilisation du déclencheur Git pour exécuter un pipeline 166

Utilisation du déclencheur Gerrit pour exécuter un pipeline 173

## **8 Surveillance des pipelines 181**

Suivi des indicateurs de performance clés d'un pipeline 181

## **9 En savoir plus 185**

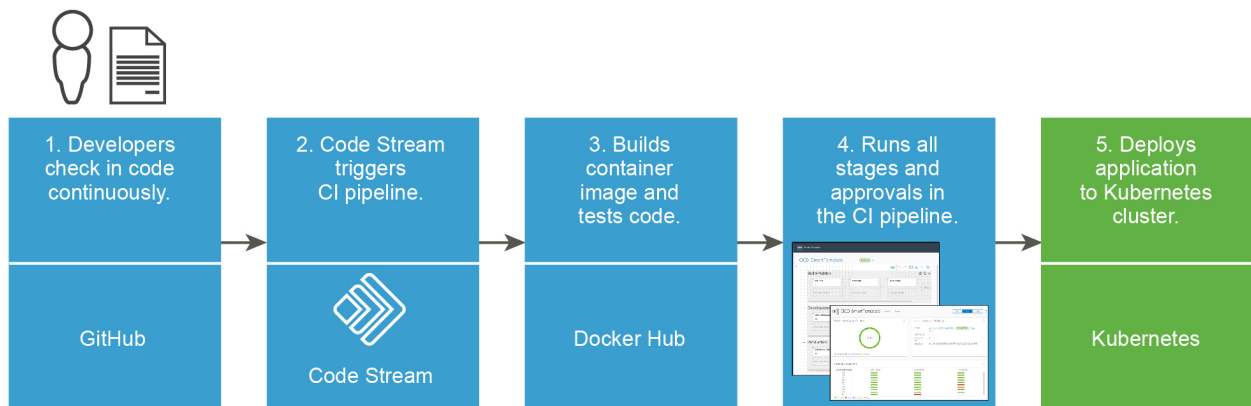
Présentation de la recherche 185

Ressources supplémentaires pour les administrateurs et les développeurs 190

# Présentation et fonctionnement de vRealize Automation Code Stream

1

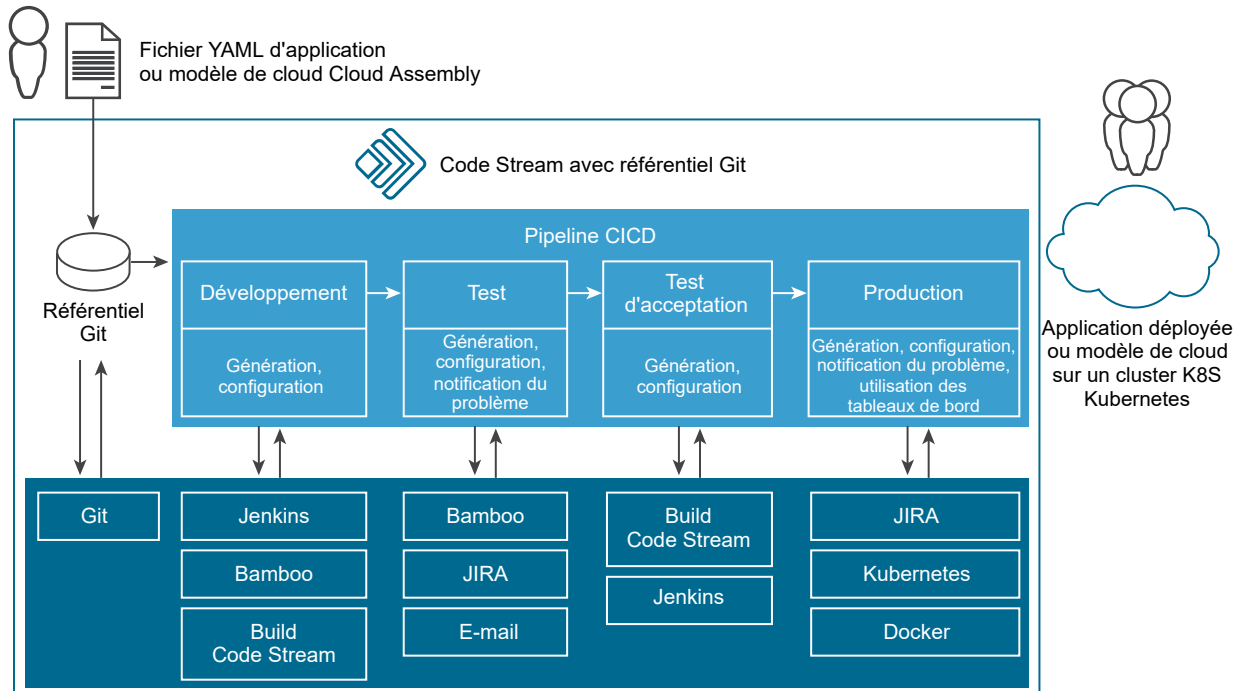
vRealize Automation Code Stream™ est un outil d'intégration continue et de prestation continue (CICD) utilisé pour générer des pipelines qui modélisent le processus de publication logicielle dans votre cycle de vie DevOps. En créant des pipelines, vous générez l'infrastructure de code nécessaire à la fourniture rapide et continue de votre logiciel.



Lorsque vous utilisez vRealize Automation Code Stream pour fournir votre logiciel, vous intégrez deux des parties les plus importantes de votre cycle de vie DevOps : votre processus de publication et vos outils de développement. Après la configuration initiale, où vRealize Automation Code Stream est intégré à vos outils de développement existants, les pipelines automatisent l'intégralité de votre cycle de vie DevOps.

À partir de vRealize Automation 8.2, les Blueprints sont appelés VMware Cloud Templates.

Vous pouvez créer un pipeline qui génère, teste et publie votre logiciel. vRealize Automation Code Stream utilise ce pipeline assurer l'évolution de votre logiciel du référentiel de code source à la production, en passant par les tests.



Pour en savoir plus sur la planification de pipelines d'intégration continue et de livraison continue, reportez-vous à la section [Chapitre 4 Planification de la génération, de l'intégration et de la livraison de votre code en mode natif dans vRealize Automation Code Stream](#).

## Utilisation de vRealize Automation Code Stream par les administrateurs de vRealize Automation Code Stream

En tant qu'administrateur, vous créez des points de terminaison et vous vous assurez que des instances opérationnelles sont disponibles pour les développeurs. Vous pouvez créer, déclencher et gérer des pipelines, et bien plus encore. Vous disposez du rôle `Administrator`, comme décrit dans [Gestion de l'accès et des approbations utilisateur dans vRealize Automation Code Stream](#).

**Tableau 1-1. Prise en charge des développeurs par les administrateurs de vRealize Automation Code Stream**

Pour aider les développeurs...	Voici ce que vous pouvez faire...
Vous fournissez et gérez des environnements.	<p>Vous créez des environnements pour que les développeurs testent et déploient leur code.</p> <ul style="list-style-type: none"> <li>■ Vous suivez l'état des environnement et envoyez des notifications par e-mail.</li> <li>■ Vous assurez la production vos développeurs en veillant à ce que leurs environnements fonctionnent en permanence.</li> </ul> <p>Pour en savoir plus, reportez-vous à la section <a href="#">Plus de ressources pour les administrateurs et les développeurs vRealize Automation Code Stream</a>.</p> <p>Reportez-vous également à la section <a href="#">Chapitre 5 Didacticiels d'utilisation de vRealize Automation Code Stream</a>.</p>
Fournissez des points de terminaison.	Vous vous assurez que les développeurs disposent d'instances opérationnelles de points de terminaison qui peuvent se connecter à leurs pipelines.
Vous autorisez les intégrations à d'autres services.	<p>Vous vous assurez que les intégrations à d'autres services fonctionnent.</p> <p>Pour en savoir plus, reportez-vous à la <a href="#">documentation vRealize Automation</a>.</p>
Créer des pipelines.	<p>Vous créez des pipelines qui modélisent les processus de publication.</p> <p>Pour en savoir plus, reportez-vous à la section <a href="#">Chapitre 3 Création et utilisation de pipelines dans vRealize Automation Code Stream</a>.</p>
Déclencher des pipelines.	<p>Vous vous assurez que les pipelines s'exécutent lorsque des événements se produisent.</p> <ul style="list-style-type: none"> <li>■ Pour déclencher un pipeline de livraison continue (CD) autonome lors de la création ou de la mise à jour d'un artefact de build, utilisez le déclencheur Docker.</li> <li>■ Pour déclencher un pipeline lorsqu'un développeur valide les modifications apportées à son code, vous utilisez le déclencheur Git.</li> <li>■ Pour déclencher un pipeline lorsque les développeurs révisent le code, la fusion, etc., vous utilisez le déclencheur Gerrit.</li> <li>■ Pour exécuter un pipeline de livraison continue (CD) autonome lors de la création ou de la mise à jour d'un artefact de build, utilisez le déclencheur Docker.</li> </ul> <p>Pour en savoir plus, reportez-vous à la section <a href="#">Chapitre 7 Déclenchement des pipelines dans vRealize Automation Code Stream</a>.</p>
Vous gérez les pipelines et les approbations.	<p>Vous réalisez un suivi continu des pipelines.</p> <ul style="list-style-type: none"> <li>■ Vous affichez l'état des pipelines et identifiez les utilisateurs qui ont exécuté les pipelines.</li> <li>■ Vous affichez les approbations sur les exécutions de pipeline et gérez les approbations pour les exécutions de pipelines actifs et inactifs.</li> </ul> <p>Pour en savoir plus, reportez-vous à la section <a href="#">Description des opérations et des approbations d'utilisateur dans vRealize Automation Code Stream</a>.</p> <p>Reportez-vous également à la section <a href="#">Suivi des indicateurs de performance clés d'un pipeline dans vRealize Automation Code Stream</a>.</p>

**Tableau 1-1. Prise en charge des développeurs par les administrateurs de vRealize Automation Code Stream (suite)**

Pour aider les développeurs...	Voici ce que vous pouvez faire...
Vous surveillez les environnements des développeurs.	<p>Vous créez des tableaux de bord personnalisés qui surveillent l'état des pipelines, les tendances, les mesures et les indicateurs clés. Utilisez les tableaux de bord personnalisés pour surveiller les pipelines qui aboutissent ou qui échouent dans les environnements des développeurs. Vous pouvez également identifier et signaler les ressources sous-utilisées, et libérez des ressources.</p> <p>Vous pouvez également afficher les éléments suivants :</p> <ul style="list-style-type: none"> <li>■ Durée d'exécution d'un pipeline avant aboutissement.</li> <li>■ Délai d'attente d'un pipeline avant approbation et notification de l'utilisateur qui doit l'approuver.</li> <li>■ Étapes et tâches qui échouent le plus souvent.</li> <li>■ Étapes et tâches dont l'exécution est la plus longue.</li> <li>■ Publications en cours des équipes de développement.</li> <li>■ Applications dont le déploiement et la publication ont abouti.</li> </ul> <p>Pour en savoir plus, reportez-vous à la section <a href="#">Chapitre 8 Surveillance des pipelines dans vRealize Automation Code Stream</a>.</p>
Résoudre des problèmes.	<p>Vous dépannez et résolvez les échecs de pipeline dans les environnements des développeurs.</p> <ul style="list-style-type: none"> <li>■ Identifier et résoudre les problèmes dans les environnements d'intégration et de livraison continues (CICD).</li> <li>■ Vous utilisez les tableaux de bord de pipeline et créez des tableaux de bord personnalisés pour en savoir plus. Reportez-vous à la section <a href="#">Chapitre 8 Surveillance des pipelines dans vRealize Automation Code Stream</a>.</li> </ul> <p>Reportez-vous également à la section <a href="#">Chapitre 2 Configuration de vRealize Automation Code Stream pour modéliser le processus de publication</a>.</p>

vRealize Automation Code Stream fait partie de vRealize Automation. vRealize Automation Code Stream s'intègre aux services suivants :

- Utilisez vRealize Automation Cloud Assembly pour déployer des modèles de cloud.
- Utilisez vRealize Automation Service Broker pour obtenir des modèles de cloud du catalogue.

Pour découvrir les autres actions possibles, consultez la [documentation de VMware vRealize Automation](#).

## Utilisation de vRealize Automation Code Stream par les développeurs

En tant que développeur, vous utilisez vRealize Automation Code Stream pour générer et exécuter des pipelines, et surveiller l'activité des pipelines sur les tableaux de bord. Vous disposez du rôle `User`, comme décrit dans [Gestion de l'accès et des approbations utilisateur dans vRealize Automation Code Stream](#).



Après avoir exécuté un pipeline, vous souhaitez savoir :


- Si votre code a passé toutes les étapes du pipeline : examinez les résultats sur la page **Exécutions**.
- Les mesures à prendre en cas d'échec du pipeline et les causes de l'échec : examinez les principales erreurs sur la page **Tableaux de bord**.

**Tableau 1-2. Utilisation de vRealize Automation Code Stream par les développeurs**

Pour intégrer et publier votre code...	Voici ce que vous faites...
Générer des pipelines.	Vous testez et déployez votre code. Vous mettez à jour votre code en cas d'échec d'un pipeline.
Connecter votre pipeline à des points de terminaison.	Vous connectez les tâches de votre pipeline à des points de terminaison, tels qu'un référentiel GitHub.
Exécuter des pipelines.	Vous ajoutez une tâche d'approbation par opérations d'utilisateur afin qu'un autre utilisateur puisse approuver votre pipeline à des points spécifiques.
Afficher les tableaux de bord.	Vous affichez les résultats sur le tableau de bord de pipeline. Vous pouvez afficher les tendances, l'historique, les échecs, etc.

Pour plus d'informations sur la mise en route, reportez-vous à [Démarrage de VMware Code Stream](#).

## Rechercher des informations supplémentaires dans le panneau de support intégré au produit

Si vous ne trouvez pas les informations dont vous avez besoin ici, vous pouvez obtenir de l'aide supplémentaire dans le produit. 

- Cliquez sur les points d'aide thématique et les info-bulles de l'interface utilisateur, et lisez-les pour obtenir des informations contextuelles pertinentes, à l'emplacement et au moment où vous en avez besoin.
- Ouvrez le panneau de support du produit et lisez les rubriques qui s'affichent pour la page de l'interface utilisateur active. Vous pouvez également effectuer une recherche dans le panneau pour obtenir des réponses aux questions.

Pour en savoir plus sur les Webhooks

Vous pouvez créer plusieurs Webhooks pour différentes branches en utilisant le même point de terminaison Git et en fournissant des valeurs différentes pour le nom de la branche sur la page de configuration du Webhook. Pour créer un autre Webhook pour une autre branche dans le même référentiel Git, vous n'avez pas besoin de cloner le point de terminaison Git plusieurs fois pour

plusieurs branches. Vous fournissez plutôt le nom de la branche dans le Webhook, ce qui vous permet de réutiliser le point de terminaison Git. Si la branche dans le Webhook Git est la même que celle du point de terminaison, vous n'avez pas besoin de fournir un nom de branche dans la page Git du Webhook.

# Configuration de vRealize Automation Code Stream pour modéliser le processus de publication

## 2

Pour modéliser votre processus de publication, vous créez un pipeline qui représente les étapes, tâches et approbations que vous utilisez généralement pour libérer votre logiciel. vRealize Automation Code Stream automatise ensuite le processus qui crée, teste, approuve et déploie votre code.

À présent que vous disposez de tout ce qui est nécessaire pour modéliser votre processus de publication logicielle, voici comment le faire dans vRealize Automation Code Stream.

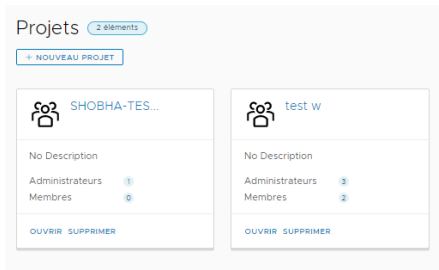
### Conditions préalables

- Vérifiez si des points de terminaison sont déjà disponibles. Dans vRealize Automation Code Stream, cliquez sur **Points de terminaison**.
- Découvrez les méthodes natives de création et de déploiement de votre code. Reportez-vous à la section [Chapitre 4 Planification de la génération, de l'intégration et de la livraison de votre code en mode natif dans vRealize Automation Code Stream](#).
- Déterminez si certaines des ressources que vous utiliserez dans votre pipeline doivent être marquées comme étant limitées. Reportez-vous à la section [Gestion de l'accès et des approbations utilisateur dans vRealize Automation Code Stream](#).
- Si vous disposez du rôle d'utilisateur ou de visualisation au lieu du rôle d'administrateur, identifiez l'administrateur de votre instance de vRealize Automation Code Stream.

### Procédure

- 1 Examinez les projets disponibles dans vRealize Automation Code Stream et sélectionnez celui qui vous convient.
  - Si aucun projet n'est répertorié, demandez à un administrateur de vRealize Automation Code Stream de créer un projet et de vous rendre membre du projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).

- Si vous n'êtes membre d'aucun projet répertorié, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet.



- 2 Ajoutez les nouveaux points de terminaison dont vous avez besoin pour votre pipeline.

Par exemple, vous pouvez avoir besoin de Git, Jenkins, Code Stream Build, Kubernetes et JIRA.

- 3 Créez des variables afin de pouvoir réutiliser des valeurs dans les tâches de votre pipeline.

Utilisez des variables restreintes pour limiter les ressources utilisées dans vos pipelines, comme une machine hôte. Vous pouvez empêcher le pipeline de poursuivre son exécution jusqu'à ce qu'un autre utilisateur l'approuve explicitement.

Les administrateurs peuvent créer des variables secrètes et des variables restreintes. Les utilisateurs peuvent créer des variables secrètes.

Vous pouvez réutiliser une variable autant de fois que vous le souhaitez sur plusieurs pipelines. Par exemple, une variable qui définit une machine hôte peut être définie sous la forme suivante : `HostIPAddress`. Ensuite, pour utiliser la variable dans une tâche de pipeline, saisissez `${var.HostIPAddress}`.

Project	Nom	Type	Valeur
0709-AWS-w2 顧客表がおA 中広@顧客@通U8&u^n	2000000000000	SECRET	*****
0709-AWS-w2 顧客表がおA 中広@顧客@通U8&u^n	20	RESTRICTED	*****
0709-AWS-w2 顧客表がおA 中広@顧客@通U8&u^n	2	SECRET	*****

- 4 Si vous êtes administrateur, marquez les points de terminaison et variables essentiels à votre entreprise comme ressources limitées.

Lorsqu'un utilisateur qui n'est pas un administrateur tente d'exécuter un pipeline qui inclut une ressource restreinte, le pipeline s'arrête sur la tâche qui utilise la ressource restreinte. Ensuite, un administrateur doit reprendre l'exécution du pipeline.

## 5 Planifiez la stratégie de build pour votre pipeline CICD, CI ou CD natif.

Avant de créer un pipeline intégrant continuellement (CI) et déployant continuellement (CD) votre code, planifiez votre stratégie de build. Le plan de build vous aide à déterminer ce dont vRealize Automation Code Stream a besoin afin de pouvoir générer, intégrer, tester et déployer votre code en mode natif.

### Pour créer une build vRealize

#### Automation Code Stream native...

Utilisez l'un des modèles de pipeline intelligents.

#### Résultats dans cette stratégie de build...

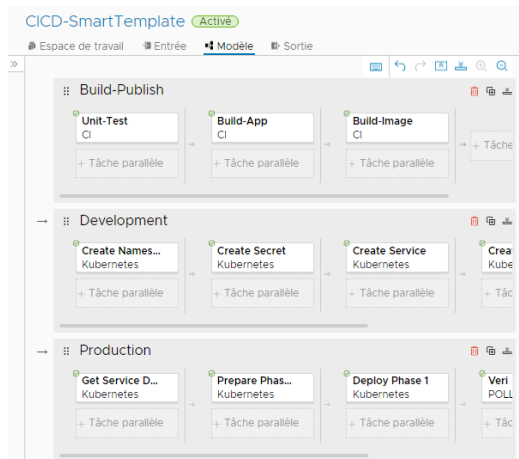
- Crée toutes les étapes et tâches pour vous.
- Clone le référentiel source.
- Génère et teste votre code.
- Place votre code dans un conteneur en vue de son déploiement.
- Remplit les étapes des tâches du pipeline en fonction de vos sélections.

Ajoutez des étapes et des tâches manuellement.

Ajoutez des étapes et des tâches, puis entrez les informations destinées à les remplir.

## 6 Créez votre pipeline à l'aide d'un modèle de cloud intelligent ou ajoutez manuellement des étapes et des tâches au pipeline.

Ensuite, vous marquez toutes les ressources comme étant restreintes. Ajoutez des approbations si nécessaire. Appliquez des variables régulières, restreintes ou secrètes. Ajoutez des liaisons entre les tâches.



## 7 Validez, activez et exécutez votre pipeline.

## 8 Affichez les exécutions de pipeline.

The screenshot displays the 'Exécutions' (Executions) page with 468 elements. It lists several pipeline runs:

- Demo-Jenkins-K... #93**: COMPLETED. Steps: 4/4. Par: sestervil. le 19 janv. 2020 à 16:17:14. Entrée: 6d82d079a8b8921a911. Sortie: -.
- Demo-Jenkins-K... #13**: COMPLETED. Steps: 4/4. Par: sestervil. le 19 janv. 2020 à 16:14:35. Entrée: 6d82d079a8b8921a9. Sortie: -.
- Demo-Jenkins-K... #92**: COMPLETED. Steps: 4/4. Par: sestervil. le 19 janv. 2020 à 16:11:06. Entrée: 8b3a29fdf\_. Sortie: -.
- Demo-CICD-Simp#48**: FAILED. Steps: 4/4. Par: sestervil. le 19 janv. 2020 à 16:09:20. Entrée: -. Sortie: -. Production.Deploy Phase 1: Failed script execution: Failed to execu...

## 9 Pour suivre l'état et les KPI, utilisez les tableaux de bord de pipeline et créez des tableaux de bord personnalisés.

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

- Dernière exécution réussie**: Demo-Jenkins-K8s #110 COMPLETED il y a 10 minutes. Exécuté par: pmartini. Durée: 25 secondes. Entrée: 8b3a29fdf\_. Sortie: -.
- Nombre d'états**: 7 derniers jours. Total: 107. Legend: COMPLETED (green), FAILED (red), WAITING (yellow).
- Statistiques d'ex...**: 14 derniers jours. Metrics: MTBD (1 minute), MTTF (3 jours), MTBD (47 minutes), MTTR (15 minutes).
- Exécutions récentes**: Table showing recent executions with columns for Execution #/Steps, Dev, and status.

Exécution #/Étapes	Dev	Status
#110	✓	COMPLETED
#109	✓	COMPLETED
#107	✓	COMPLETED
#104	✓	COMPLETED
#103	✓	COMPLETED
#102	✓	COMPLETED
#101	✓	COMPLETED
#100	✓	COMPLETED
#99	✓	COMPLETED
#98	✓	COMPLETED

Legend: COMPLETED (green), FAILED (red), RUNNING (blue), WAITING (yellow), CANCELED (pink), ROLLBACK\_COMPLETED (red), ROLLBACK\_FAILED (red).

### Résultats

Vous avez créé un pipeline que vous pouvez utiliser dans le projet sélectionné.

Vous pouvez également exporter votre fichier YAML de pipeline pour l'importer et le réutiliser dans d'autres projets.

### Étape suivante

Découvrez les cas d'utilisation que vous souhaitez peut-être appliquer dans votre environnement. Reportez-vous à la section [Chapitre 5 Didacticiels d'utilisation de vRealize Automation Code Stream](#).

## Ajout d'un projet dans vRealize Automation Code Stream

Vous créez un projet auquel vous ajoutez des administrateurs et des membres de manière à ce que les membres du projet puissent utiliser des fonctionnalités (créer un pipeline ou ajouter un point de terminaison, par exemple). Pour créer, supprimer ou mettre à jour un projet pour une équipe de développement, vous devez être un administrateur de vRealize Automation Code Stream.

Vous ne pouvez créer un pipeline que s'il existe un projet. Lorsque vous créez un pipeline, vous devez sélectionner le projet auquel l'associer afin de regrouper toutes les informations du pipeline. Les définitions des points de terminaison et des variables dépendent également d'un projet existant.

### Conditions préalables

- Vérifiez que vous disposez du rôle d'administrateur de vRealize Automation Code Stream. Reportez-vous à la section [Présentation des rôles dans vRealize Automation Code Stream](#).  
  
Si vous ne disposez pas du rôle d'administrateur de vRealize Automation Code Stream, mais que vous êtes un administrateur dans vRealize Automation Cloud Assembly, vous pouvez utiliser l'interface utilisateur de vRealize Automation Cloud Assembly pour créer, mettre à jour ou supprimer des projets. Reportez-vous à [Ajout d'un projet pour mon équipe de développement vRealize Automation Cloud Assembly](#)
- Si vous ajoutez des groupes Active Directory à des projets, vérifiez que vous avez configuré des groupes Active Directory pour votre organisation. Reportez-vous à la section [Activation de groupes Active Directory dans vRealize Automation pour les projets](#). Si les groupes ne sont pas synchronisés, ils ne sont pas disponibles lorsque vous tentez de les ajouter à un projet.

### Procédure

- 1 Sélectionnez **Projets**, puis cliquez sur **Nouveau projet**.
- 2 Entrez le nom du projet.
- 3 Cliquez sur **Créer**.
- 4 Sélectionnez la fiche du projet que vous venez de créer et cliquez sur **Ouvrir**.
- 5 Cliquez sur l'onglet **Utilisateurs** et ajoutez des utilisateurs avec les rôles qui leur sont attribués.
  - Tout administrateur de projet peut ajouter des membres.
  - Tout membre du projet disposant d'un rôle de service peut utiliser des services.
  - L'observateur de projet peut voir des projets, mais ne peut pas les créer, les mettre à jour ou les supprimer.

Pour plus d'informations sur les rôles de projet, consultez [Gestion de l'accès et des approbations utilisateur dans vRealize Automation Code Stream](#).
- 6 Cliquez sur **Enregistrer**.

### Étape suivante

Ajoutez des points de terminaison et des pipelines qui utilisent le projet. Reportez-vous au [Chapitre 6 Connexion de vRealize Automation Code Stream à des points de terminaison](#) et au [Chapitre 3 Création et utilisation de pipelines dans vRealize Automation Code Stream](#).

## Gestion de l'accès et des approbations utilisateur dans vRealize Automation Code Stream

vRealize Automation Code Stream propose plusieurs manières de s'assurer que les utilisateurs disposent de l'autorisation appropriée pour utiliser les pipelines conduisant à la publication de vos applications logicielles.

Chaque membre d'une équipe se voit attribuer un rôle qui lui confère des autorisations spécifiques sur les pipelines, les points de terminaison et les tableaux de bord, et lui offre la possibilité de marquer des ressources comme étant limitées.

Les opérations et approbations d'utilisateur vous permettent de contrôler à quel moment un pipeline s'exécute et doit s'arrêter pour une approbation. Votre rôle détermine si vous pouvez reprendre un pipeline et exécuter des pipelines qui incluent des variables ou des points de terminaison restreints.

Utilisez des variables secrètes pour masquer et chiffrer des informations sensibles. Utilisez une variable limitée pour les chaînes, les mots de passe et les URL qui doivent être masqués et chiffrés, ainsi que pour en restreindre l'utilisation dans les exécutions. Par exemple, utilisez une variable secrète pour un mot de passe ou une URL. Vous pouvez utiliser des variables secrètes et restreintes pour n'importe quel type de tâche dans votre pipeline.

## Présentation des rôles dans vRealize Automation Code Stream

En fonction de votre rôle dans vRealize Automation Code Stream, vous pouvez effectuer certaines actions et accéder à certaines zones. Par exemple, selon votre rôle, vous pouvez créer, mettre à jour et exécuter des pipelines. Vous pouvez également être seulement autorisé à afficher les pipelines.

Toutes les actions sauf limitées signifie que ce rôle a l'autorisation d'effectuer des actions de création, de lecture, de mise à jour et de suppression sur des entités, à l'exception des variables et des points de terminaison limités.



**Tableau 2-1. Autorisations d'accès au niveau du service et du projet dans vRealize Automation Code Stream**

Rôles vRealize Automation Code Stream					
Niveaux d'accès	Administrateur de Code Stream	Développeur Code Stream	Exécuteur Code Stream	Visualisation de Code Stream	Utilisateur de Code Stream
Accès au niveau du service vRealize Automation Code Stream	Toutes les actions	Toutes les actions sauf limitées	Actions de l'exécuteur	Lecture seule	aucune
Accès au niveau du projet : administrateur du projet	Toutes les actions	Toutes les actions	Toutes les actions	Toutes les actions	Toutes les actions
Accès au niveau du projet : membre du projet	Toutes les actions	Toutes les actions sauf limitées	Toutes les actions sauf limitées	Toutes les actions sauf limitées	Toutes les actions sauf limitées
Accès au niveau du projet : observateur du projet	Toutes les actions	Toutes les actions sauf limitées	Actions de l'exécuteur	Lecture seule	Lecture seule

Les utilisateurs qui disposent du rôle d'administrateur de projet peuvent effectuer toutes les actions sur les projets pour lesquels ils sont un administrateur de projet.

Un administrateur de projet peut créer, lire, mettre à jour et supprimer des pipelines, des variables, des points de terminaison, des tableaux de bord et des déclencheurs, mais aussi démarrer un pipeline qui inclut des points de terminaison ou des variables limités si ces ressources se trouvent dans le projet dans lequel l'utilisateur est un administrateur de projet.

Les utilisateurs qui disposent du rôle d'observateur de service peuvent afficher toutes les informations qui sont disponibles pour l'administrateur. Ils ne peuvent effectuer aucune action, sauf si un administrateur leur accorde un rôle d'administrateur de projet ou de membre du projet. Si l'utilisateur est affilié à un projet, il dispose des autorisations associées à ce rôle. Les autorisations de l'observateur de projet ne peuvent pas être étendues de la même manière que pour le rôle d'administrateur ou de membre. Ce rôle est en lecture seule dans tous les projets.

Si vous disposez d'autorisations de lecture dans un projet, vous pouvez toujours afficher des ressources limitées.

- Pour afficher les points de terminaison limités (qui affichent une icône de cadenas sur la carte du point de terminaison), cliquez sur **Configurer > Points de terminaison**.
- Pour afficher les variables limitées et secrètes (qui s'affichent comme étant LIMITÉE ou SECRÈTE dans la colonne **Type**), cliquez sur **Configurer > Variable**.

Tableau 2-2. Fonctionnalités des rôles de service vRealize Automation Code Stream

Contexte de l'interface utilisateur	Capacités	Rôle d'administrateur de Code Stream	Rôle de développeur de Code Stream	Rôle d'exécuteur de Code Stream	Rôle d'observateur de Code Stream	Rôle d'utilisateur de Code Stream
Pipelines						
	Afficher les pipelines	Oui	Oui	Oui	Oui	
	Créer des pipelines	Oui	Oui			
	Exécuter des pipelines	Oui	Oui	Oui		
	Exécuter les pipelines qui incluent des variables ou des points de terminaison limités	Oui				
	Mettre à jour des pipelines	Oui	Oui			
	Supprimer des pipelines	Oui	Oui			
Exécutions de pipeline						
	Afficher les exécutions de pipeline	Oui	Oui	Oui	Oui	
	Reprendre, suspendre et annuler les exécutions de pipeline	Oui	Oui	Oui		
	Reprendre les pipelines qui s'arrêtent pour approbation sur les ressources limitées	Oui				
Intégrations personnalisées						
	Créer des intégrations personnalisées	Oui	Oui			
	Lire les intégrations personnalisées	Oui	Oui	Oui	Oui	
	Mettre à jour les intégrations personnalisées	Oui	Oui			
Points de terminaison						
	Afficher les exécutions	Oui	Oui	Oui	Oui	

Tableau 2-2. Fonctionnalités des rôles de service vRealize Automation Code Stream (suite)

Contexte de l'interface utilisateur	Capacités	Rôle d'administrateur de Code Stream	Rôle de développeur de Code Stream	Rôle d'exécuteur de Code Stream	Rôle d'observateur de Code Stream	Rôle d'utilisateur de Code Stream
	Créer des exécutions	Oui	Oui			
	Mettre à jour les exécutions	Oui	Oui			
	Supprimer les exécutions	Oui	Oui			
<b>Marquer les ressources comme étant limitées</b>						
	Marquer une variable ou un point de terminaison comme étant limité(e)	Oui				
<b>Tableaux de bord</b>						
	Afficher les tableaux de bord	Oui	Oui	Oui	Oui	
	Créer des tableaux de bord	Oui	Oui			
	Mettre à jour les tableaux de bord	Oui	Oui			
	Supprimer les tableaux de bord	Oui	Oui			

## Rôles personnalisés et autorisations dans vRealize Automation Code Stream

Vous pouvez créer des rôles personnalisés dans vRealize Automation Cloud Assembly qui étendent des privilèges aux utilisateurs qui utilisent des pipelines. Lorsque vous créez un rôle personnalisé pour des pipelines vRealize Automation Code Stream, vous sélectionnez une ou plusieurs autorisations de **Pipeline**.

Sélectionnez le nombre minimal d'autorisations de **Pipeline** requises pour les utilisateurs auxquels ce rôle personnalisé sera attribué.

Lorsqu'un projet est attribué à un utilisateur, qu'un rôle lui est donné et qu'un rôle personnalisé incluant une ou plusieurs autorisations de **Pipeline** lui est attribué, cet utilisateur peut effectuer toutes les actions permises par les autorisations. Par exemple, il peut créer des variables limitées, gérer des pipelines limités, créer et gérer des intégrations personnalisées, etc.

Tableau 2-3. Autorisations de pipeline que vous pouvez attribuer à des rôles personnalisés

Autorisation de pipeline	Administrateur de Code Stream	Développeur Code Stream	Exécuteur Code Stream	Visualisation de Code Stream	Utilisateur de Code Stream	Administrateur de projet	Membre du projet	Observateur de projet
Gérer les pipelines	Oui	Oui				Oui	Oui	
Gérer les pipelines limités	Oui					Oui		
Gérer les intégrations personnalisées	Oui	Oui						
Exécuter les pipelines	Oui	Oui	Oui			Oui	Oui	
Exécuter les pipelines limités	Oui					Oui		
Gérer les exécutions	Oui					Oui		
Lecture. Cette autorisation n'est pas visible.	Oui	Oui	Oui	Oui		Oui	Oui	Oui

Tableau 2-4. Utilisation des autorisations de pipeline avec des rôles personnalisés

Autorisation	Actions possibles
Gérer les pipelines	<ul style="list-style-type: none"> <li>■ Créer, mettre à jour, supprimer et cloner des pipelines.</li> <li>■ Publier et annuler la publication des pipelines pour VMware Service Broker.</li> <li>■ Créer, mettre à jour et supprimer des points de terminaison.</li> <li>■ Créer, mettre à jour et supprimer des variables régulières et secrètes.</li> <li>■ Créer, cloner, mettre à jour et supprimer un écouteur Gerrit.</li> <li>■ Connecter et déconnecter un écouteur Gerrit.</li> <li>■ Créer, cloner, mettre à jour et supprimer un déclencheur Gerrit.</li> <li>■ Créer, mettre à jour et supprimer un Webhook Git.</li> <li>■ Créer, mettre à jour et supprimer un Webhook Docker.</li> <li>■ Utiliser des modèles de pipeline intelligents pour créer des pipelines.</li> <li>■ Importer des pipelines à partir de YAML et les exporter dans YAML.</li> <li>■ Créer, mettre à jour et supprimer des tableaux de bord personnalisés.</li> <li>■ Lire toutes les intégrations personnalisées.</li> <li>■ Lire tous les points de terminaison et variables limités, mais pas afficher leurs valeurs.</li> </ul>
Gérer les pipelines limités	<ul style="list-style-type: none"> <li>■ Créer, mettre à jour et supprimer des points de terminaison.</li> <li>■ Marquer des points de terminaison comme étant limités, mettre à jour les points de terminaison limités et les supprimer.</li> <li>■ Créer, mettre à jour et supprimer des variables régulières et secrètes.</li> <li>■ Créer, mettre à jour et supprimer des variables limitées.</li> <li>■ Toutes les autorisations que vous pouvez effectuer avec l'autorisation Gérer les pipelines.</li> </ul>
Gérer les intégrations personnalisées	<ul style="list-style-type: none"> <li>■ Créer et mettre à jour des intégrations personnalisées.</li> <li>■ Gérer la version et publier des intégrations personnalisées.</li> <li>■ Supprimer et déconseiller des versions d'intégrations personnalisées.</li> <li>■ Supprimer des intégrations personnalisées.</li> </ul>
Exécuter les pipelines	<ul style="list-style-type: none"> <li>■ Exécuter des pipelines.</li> <li>■ Suspendre, reprendre et annuler les exécutions de pipeline.</li> <li>■ Relancer des exécutions de pipeline.</li> <li>■ Reprendre, relancer et déclencher manuellement un événement déclencheur Gerrit.</li> <li>■ Approuver une opération d'utilisateur et effectuer des approbations par lot sur les opérations d'utilisateur.</li> </ul>

Tableau 2-4. Utilisation des autorisations de pipeline avec des rôles personnalisés (suite)

Autorisation	Actions possibles
Exécuter les pipelines limités	<ul style="list-style-type: none"> <li>■ Exécuter des pipelines.</li> <li>■ Suspendre, reprendre, annuler et supprimer des exécutions de pipeline.</li> <li>■ Relancer des exécutions de pipeline.</li> <li>■ Synchroniser l'exécution d'un pipeline en cours d'exécution.</li> <li>■ Forcer la suppression de l'exécution d'un pipeline en cours d'exécution.</li> <li>■ Reprendre, relancer, supprimer et déclencher manuellement un événement déclencheur Gerrit.</li> <li>■ Résoudre les éléments limités et poursuivre l'exécution du pipeline.</li> <li>■ Changer le contexte utilisateur et poursuivre l'exécution du pipeline après l'approbation d'une tâche d'opération d'utilisateur.</li> <li>■ Toutes les autorisations que vous pouvez effectuer avec l'autorisation Exécuter les pipelines.</li> </ul>
Gérer les exécutions	<ul style="list-style-type: none"> <li>■ Exécuter des pipelines.</li> <li>■ Suspendre, reprendre, annuler et supprimer des exécutions de pipeline.</li> <li>■ Relancer des exécutions de pipeline.</li> <li>■ Reprendre, relancer, supprimer et déclencher manuellement un événement déclencheur Gerrit.</li> <li>■ Toutes les autorisations que vous pouvez effectuer avec l'autorisation Exécuter les pipelines.</li> </ul>

Les rôles personnalisés peuvent inclure des combinaisons d'autorisations. Ces autorisations sont organisées en groupes de capacités, qui permettent aux utilisateurs de gérer ou d'exécuter des pipelines, avec et sans ressources limitées. Ces autorisations représentent toutes les fonctionnalités que chaque rôle peut effectuer dans vRealize Automation Code Stream.

Par exemple, si vous créez un rôle personnalisé et que vous incluez l'autorisation appelée **Gérer les pipelines limités**, les utilisateurs disposant du rôle de développeur de vRealize Automation Code Stream peuvent :

- Créer, mettre à jour et supprimer des points de terminaison.
- Marquer des points de terminaison comme étant limités, mettre à jour les points de terminaison limités et les supprimer.
- Créer, mettre à jour et supprimer des variables régulières et secrètes.
- Créer, mettre à jour et supprimer des variables limitées.

Tableau 2-5. Exemples de combinaisons d'autorisations de pipeline dans les rôles personnalisés

Nombre d'autorisations attribuées au rôle personnalisé	Exemples d'autorisations combinées	Utilisation de ce tableau de bord
Une seule autorisation	<b>Exécuter les pipelines</b>	
Deux autorisations	<b>Gérer les pipelines et Exécuter les pipelines</b>	
Trois autorisations	<b>Gérer les pipelines, Exécuter les pipelines et Exécuter les pipelines limités</b>	
	<b>Gérer les pipelines, Gérer les intégrations personnalisées et Exécuter les pipelines limités</b>	Cette combinaison peut s'appliquer à un rôle de développeur de vRealize Automation Code Stream, mais elle est limitée aux projets dont l'utilisateur est membre.
	<b>Gérer les pipelines, Gérer les intégrations personnalisées et Gérer les exécutions</b>	Cette combinaison peut s'appliquer à un administrateur de vRealize Automation Code Stream, mais elle limitée aux projets dont l'utilisateur est membre.
	<b>Gérer les pipelines, Gérer les pipelines limités et Gérer les intégrations personnalisées</b>	Avec cette combinaison, un utilisateur dispose d'autorisations complètes et peut créer et supprimer tous les éléments dans vRealize Automation Code Stream.

## Si vous disposez du rôle d'administrateur

En tant qu'administrateur, vous pouvez créer des intégrations personnalisées, des points de terminaison, des variables, des déclencheurs, des pipelines et des tableaux de bord.

Les projets permettent aux pipelines d'accéder aux ressources de l'infrastructure. Les administrateurs créent des projets afin que les utilisateurs puissent regrouper des pipelines, des points de terminaison et des tableaux de bord. Les utilisateurs sélectionnent ensuite le projet dans leurs pipelines. Chaque projet inclut un administrateur et les utilisateurs auxquels des rôles ont été attribués.

Avec le rôle d'administrateur, vous pouvez marquer des points de terminaison et des variables comme étant des ressources limitées et vous pouvez exécuter des pipelines qui utilisent des ressources limitées. Si un utilisateur non administrateur exécute un pipeline qui inclut un point de terminaison ou une variable limité(e), le pipeline s'arrête au niveau de la tâche où la variable limitée est utilisée et un administrateur doit reprendre l'exécution du pipeline.

En tant qu'administrateur, vous pouvez également demander que les pipelines soient publiés dans vRealize Automation Service Broker.

## Si vous disposez du rôle de développeur

Vous pouvez utiliser des pipelines comme un administrateur, sauf que vous ne pouvez pas utiliser de variables ou de points de terminaison limités.

Si vous exécutez un pipeline qui utilise des variables ou des points de terminaison restreints, le pipeline ne s'exécute que jusqu'à la tâche qui utilise la ressource restreinte. Ensuite, le pipeline s'arrête et un administrateur de vRealize Automation Code Stream ou un administrateur de projet doit reprendre l'exécution du pipeline.

## Si vous disposez du rôle d'utilisateur

Vous pouvez accéder à vRealize Automation Code Stream, mais ne disposez d'aucun privilège associé à d'autres rôles.

## Si vous disposez du rôle de visualisation

Vous pouvez afficher les mêmes ressources que l'administrateur (par exemple, les pipelines, les points de terminaison, les exécutions de pipeline, les tableaux de bord, les intégrations personnalisées et les déclencheurs), mais vous ne pouvez pas les créer, les mettre à jour ou les supprimer. Pour effectuer des actions, le rôle d'administrateur de projet ou de membre de projet doit également être attribué au rôle d'observateur.

Les utilisateurs disposant du rôle d'observateur peuvent afficher les projets. Ils peuvent également voir les points de terminaison limités et les variables limitées, mais ne peuvent pas voir les informations détaillées les concernant.

## Si vous disposez du rôle d'exécuteur

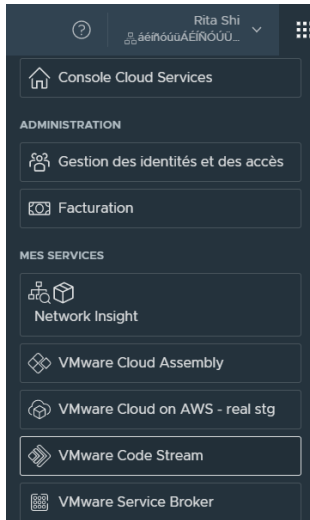
Vous pouvez exécuter des pipelines et effectuer des actions sur les tâches d'opération d'utilisateur. Vous pouvez également reprendre, suspendre et annuler les exécutions de pipeline, mais vous ne pouvez pas modifier les pipelines.

## Comment attribuer et mettre à jour des rôles

Pour attribuer et mettre à jour des rôles pour d'autres utilisateurs, vous devez être administrateur.

- 1 Pour afficher les utilisateurs actifs et leurs rôles, dans vRealize Automation, cliquez sur les neuf points dans le coin supérieur droit.
- 2 Cliquez sur **Gestion des identités et des accès**.





- 3 Pour afficher le nom et les rôles des utilisateurs, cliquez sur **Utilisateurs actifs**.



- 4 Pour ajouter des rôles pour un utilisateur ou modifier ses rôles, cochez la case en regard du nom d'utilisateur et cliquez sur **Modifier les rôles**.
- 5 Lorsque vous ajoutez ou modifiez des rôles d'utilisateur, vous pouvez également ajouter un accès aux services.
- 6 Pour enregistrer vos modifications, cliquez sur **Enregistrer**.

## Description des opérations et des approbations d'utilisateur dans vRealize Automation Code Stream

La zone Opérations d'utilisateur affiche les exécutions de pipeline requérant une approbation. L'utilisateur qui est requis en tant qu'approbateur peut approuver ou rejeter l'exécution du pipeline.

Lorsque vous créez un pipeline, il est possible que vous deviez ajouter une approbation à un pipeline si :

- Un membre de l'équipe doit vérifier votre code.
- Un autre utilisateur doit vérifier un artefact de build.
- Vous devez vous assurer que tous les tests sont terminés.
- Une tâche utilise une ressource marquée comme étant limitée par un administrateur et la tâche a besoin d'une approbation.
- Le pipeline va publier le logiciel en production.

L'approbateur requis pour une tâche de pipeline doit disposer de l'autorisation et de l'expertise pour déterminer s'il doit l'approuver.

Lorsque vous ajoutez une tâche d'opération utilisateur, vous pouvez définir le délai d'expiration en jours, heures ou minutes. Par exemple, vous avez besoin que l'utilisateur requis approuve le pipeline dans 30 minutes. Si l'approbation n'est pas obtenue dans les 30 minutes qui suivent, le pipeline échoue comme prévu.

Une fois que l'utilisateur requis a approuvé la tâche d'opération de l'utilisateur :

- L'exécution du pipeline en attente peut continuer.
- Lorsque le pipeline se poursuit, toutes les demandes précédentes en attente d'approbation de cette tâche d'opération utilisateur sont annulées.

**User Operations** GUIDED SETUP

Active Items Inactive Items

✓ APPROVE ✗ REJECT

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

1 Items per page 20 1 - 7 of 7 items

Dans la zone Opérations d'utilisateur, les éléments à approuver ou à rejeter s'affichent comme des éléments actifs ou inactifs. Chaque élément est mappé à une tâche d'opération d'utilisateur dans un pipeline.

- Les **Éléments actifs** attendent que l'approbateur révise la tâche, puis l'approuve ou la refuse. Si vous êtes un utilisateur figurant dans la liste d'approbatrices, vous pouvez développer la ligne d'opération de l'utilisateur et cliquer sur **Accepter** ou **Rejeter**.

- Les **Éléments inactifs** ont été approuvés ou rejetés. Si un utilisateur a rejeté l'opération de l'utilisateur, ou si l'approbation sur la tâche a expiré, elle ne peut plus être approuvée.

N° d'index est une chaîne de six caractères alphanumériques unique que vous pouvez utiliser comme filtre pour rechercher une approbation particulière.

Les approbations de pipeline s'affichent également dans la zone **Exécutions**.

- Les pipelines en attente d'approbation indiquent leur état en attente.
- Les autres états incluent les suivants : En file d'attente, Terminé, Échec.
- Si votre pipeline est en attente, l'approbateur requis doit approuver votre tâche de pipeline.

# Création et utilisation de pipelines dans vRealize Automation Code Stream

## 3

Vous pouvez utiliser vRealize Automation Code Stream pour modéliser votre processus de génération, de test et de déploiement. Avec vRealize Automation Code Stream, vous pouvez configurer l'infrastructure qui prend en charge votre cycle de publication et créer des pipelines qui modélisent vos activités de publication logicielle. vRealize Automation Code Stream crée votre logiciel à partir du code de développement, lui fait passer les tests requis, puis le déploie sur vos instances de production.

Chaque pipeline inclut des étapes et des tâches. Les étapes représentent vos phases de développement et les tâches effectuent les actions requises pour que votre application logicielle franchisse toutes les étapes.

## Présentation des pipelines dans vRealize Automation Code Stream

Un pipeline est un modèle d'intégration continue et de livraison continue de votre processus de publication logicielle. Il publie votre logiciel depuis le code source jusqu'à la production, en lui faisant passer les tests requis. Il inclut une séquence d'étapes avec des tâches qui représentent les activités de votre cycle de publication logicielle. Votre application logicielle passe d'une étape à la suivante via le pipeline.

Vous ajoutez des points de terminaison afin que les tâches de votre pipeline puissent se connecter à des sources de données, des référentiels ou des systèmes de notification.

## Création de pipelines

Pour créer un pipeline, vous pouvez commencer par un canevas vide, utiliser un modèle de pipeline intelligent ou importer du code YAML.

- Utilisez le canevas vide. Pour obtenir un exemple, reportez-vous à la section [Planification d'une build native CI/CD dans vRealize Automation Code Stream avant d'ajouter des tâches manuellement](#).
- Utilisez un modèle de pipeline intelligent. Pour obtenir un exemple, reportez-vous à la section [Chapitre 4 Planification de la génération, de l'intégration et de la livraison de votre code en mode natif dans vRealize Automation Code Stream](#).

- Importez le code YAML. Cliquez sur **Pipelines > Importer**. Dans la boîte de dialogue **Importer**, sélectionnez le fichier YAML ou entrez le code YAML, puis cliquez sur **Importer**.

Lorsque vous utilisez le canevas vide pour créer un pipeline, vous pouvez ajouter des étapes, des tâches et des approbations. Le pipeline automatise le processus de création, de test, de déploiement et de libération de votre publication. Les tâches définies à chaque étape exécutent des actions qui génèrent, testent et publient le code à l'étape en cours.

**Tableau 3-1. Exemples d'étapes et d'utilisation de pipeline**

Exemples d'étapes	Exemples de ce que vous pouvez faire
Développement	<p>À une étape de développement, vous pouvez provisionner une machine, récupérer un artefact ou ajouter une tâche de génération pour créer un hôte Docker à utiliser pour l'intégration continue de votre code, etc.</p> <p>Par exemple :</p> <ul style="list-style-type: none"> <li>■ Pour planifier et créer une build d'intégration continue (CI), qui fournit votre code à l'aide de la fonctionnalité de build native dans vRealize Automation Code Stream, reportez-vous à la section <a href="#">Planification d'une build native CI dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent</a>.</li> </ul>
Test	<p>À une étape de test, vous pouvez ajouter une tâche Jenkins pour tester votre application logicielle et inclure des outils de test post-traitement, tels que JUnit et JaCoCo, etc.</p> <p>Par exemple :</p> <ul style="list-style-type: none"> <li>■ Intégrez vRealize Automation Code Stream à Jenkins et exécutez un travail Jenkins dans votre pipeline afin de générer et tester votre code source. Reportez-vous à la section <a href="#">Intégration de vRealize Automation Code Stream à Jenkins</a>.</li> <li>■ Créez des scripts personnalisés qui étendent la capacité de vRealize Automation Code Stream afin d'y intégrer vos propres outils de génération, de test et de déploiement. Reportez-vous à la section <a href="#">Intégration d'outils de génération, de test et de déploiement propres avec vRealize Automation Code Stream</a>.</li> <li>■ Suivez les tendances d'un pipeline d'intégration continue (CI) en matière de post-traitement. Reportez-vous à la section <a href="#">Suivi des indicateurs de performance clés d'un pipeline dans vRealize Automation Code Stream</a>.</li> </ul>
Production	<p>À une étape de production, vous pouvez intégrer un modèle de cloud dans vRealize Automation Cloud Assembly pour provisionner votre infrastructure, déployer votre logiciel sur un cluster Kubernetes, etc.</p> <p>Par exemple :</p> <ul style="list-style-type: none"> <li>■ Pour obtenir des exemples d'étapes de développement et de production qui permettent de déployer une application logicielle dans votre propre modèle de déploiement Bleu-vert, reportez-vous à la section <a href="#">Déploiement d'une application dans vRealize Automation Code Stream vers un déploiement Bleu-vert</a>.</li> <li>■ Pour intégrer un modèle de cloud à votre pipeline, reportez-vous à la section <a href="#">Automatisation de la publication d'une application déployée à partir d'un modèle de cloud YAML dans vRealize Automation Code Stream</a>. Vous pouvez également ajouter une tâche de déploiement qui exécute un script pour déployer l'application.</li> <li>■ Pour automatiser le déploiement de vos applications logicielles sur un cluster Kubernetes, reportez-vous à la section <a href="#">Automatisation de la publication d'une application dans vRealize Automation Code Stream sur un cluster Kubernetes</a>.</li> <li>■ Pour intégrer du code dans votre pipeline et déployer votre image de build, reportez-vous à la section <a href="#">Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans vRealize Automation Code Stream</a>.</li> </ul>

Vous pouvez exporter votre pipeline en tant que fichier YAML. Cliquez sur **Pipelines**, puis sur une fiche de pipeline et enfin sur **Actions > Exporter**.

## Approbation des pipelines

Vous pouvez obtenir l'approbation d'un autre membre de l'équipe à des points spécifiques de votre pipeline.

- Pour exiger une approbation sur un pipeline en incluant une tâche d'opération utilisateur dans un pipeline, reportez-vous à la section [Comment exécuter un pipeline et afficher les résultats](#). Cette tâche envoie une notification par e-mail à l'utilisateur qui doit la vérifier. Le réviseur doit approuver ou rejeter l'approbation pour que l'exécution du pipeline puisse se poursuivre. Si le délai d'expiration de la tâche Opération de l'utilisateur est défini en jours, heures ou minutes, l'utilisateur requis doit approuver le pipeline avant l'expiration de la tâche. Sinon, le pipeline échoue comme prévu.
- À chaque étape d'un pipeline, si une tâche ou une étape échoue, vous pouvez demander à vRealize Automation Code Stream de créer un ticket JIRA. Reportez-vous à la section [Création d'un ticket JIRA dans vRealize Automation Code Stream en cas d'échec d'une tâche de pipeline](#).

## Déclenchement de pipelines

Des pipelines peuvent se déclencher lorsque les développeurs archivent ou révisent le code, ou encore lorsqu'un artefact de build est créé ou mis à jour.

- Pour intégrer vRealize Automation Code Stream au cycle de vie Git et déclencher un pipeline lorsque les développeurs mettent à jour le code, utilisez le déclencheur Git. Reportez-vous à la section [Utilisation du déclencheur Git dans vRealize Automation Code Stream pour exécuter un pipeline](#).
- Pour intégrer vRealize Automation Code Stream au cycle de vie de révision du code Gerrit et déclencher un pipeline en cas de révision du code, utilisez le déclencheur Gerrit. Reportez-vous à la section [Utilisation du déclencheur Gerrit dans vRealize Automation Code Stream pour exécuter un pipeline](#).
- Pour déclencher un pipeline lors de la création ou de la mise à jour d'un artefact de build Docker, utilisez le déclencheur Docker. Reportez-vous à la section [Utilisation du déclencheur Docker dans vRealize Automation Code Stream pour exécuter un pipeline de livraison continue](#).

Pour plus d'informations sur les déclencheurs pris en charge par vRealize Automation Code Stream, reportez-vous à la section [Chapitre 7 Déclenchement des pipelines dans vRealize Automation Code Stream](#).

Ce chapitre contient les rubriques suivantes :

- [Comment exécuter un pipeline et afficher les résultats](#)

- Types de tâche disponibles dans vRealize Automation Code Stream
- Utilisation des liaisons de variables dans les pipelines vRealize Automation Code Stream
- Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline dans vRealize Automation Code Stream
- Quelles variables et expressions peuvent être utilisées lors de la liaison de tâches de pipeline dans vRealize Automation Code Stream
- Envoi de notifications relatives à un pipeline dans vRealize Automation Code Stream
- Création d'un ticket JIRA dans vRealize Automation Code Stream en cas d'échec d'une tâche de pipeline
- Restauration d'un déploiement dans vRealize Automation Code Stream

## Comment exécuter un pipeline et afficher les résultats

Vous pouvez exécuter un pipeline à partir de la fiche de pipeline, en mode d'édition de pipeline et à partir de l'exécution de pipeline. Vous pouvez également utiliser les déclencheurs disponibles pour que vRealize Automation Code Stream exécute un pipeline lorsque certains événements se produisent.

Lorsque toutes les étapes et tâches du pipeline sont valides, le pipeline est prêt à être publié, exécuté ou déclenché.

Pour exécuter ou déclencher votre pipeline à l'aide de vRealize Automation Code Stream, vous pouvez activer et exécuter le pipeline à partir de la fiche de pipeline ou pendant que vous vous trouvez dans le pipeline. Ensuite, vous pouvez afficher l'exécution du pipeline pour confirmer que le pipeline a créé, testé et déployé votre code.

Lorsqu'une exécution de pipeline est en cours, vous pouvez supprimer l'exécution si vous êtes un administrateur ou un utilisateur non-administrateur.

- Administrateur : pour supprimer l'exécution d'un pipeline en cours d'exécution, cliquez sur **Exécutions**. Sur l'exécution à supprimer, cliquez sur **Actions > Supprimer**.
- Utilisateur non-administrateur : pour supprimer l'exécution d'un pipeline en cours d'exécution, cliquez sur **Exécutions**, puis cliquez sur **Alt Maj d**.

Lorsqu'une exécution de pipeline est en cours et semble bloquée, un administrateur peut actualiser l'exécution à partir de la page Exécutions ou de la page Détails de l'exécution.

- Page Exécutions : cliquez sur **Exécutions**. Sur l'exécution à actualiser, cliquez sur **Actions > Synchroniser**.
- Page Détails de l'exécution : cliquez sur **Exécutions**, cliquez sur le lien vers les détails de l'exécution, puis sur **Actions > Synchroniser**.

Pour exécuter un pipeline lorsque des événements spécifiques se produisent, utilisez les déclencheurs.

- Le déclencheur Git peut exécuter un pipeline lorsque les développeurs mettent à jour le code.

- Le déclencheur Gerrit peut exécuter un pipeline lors de révisions de code.
- Le déclencheur Docker peut exécuter un pipeline lorsqu'un artefact est créé dans un registre Docker.
- La commande `curl` peut demander à Jenkins d'exécuter un pipeline à la fin d'une build Jenkins.

Pour plus d'informations sur l'utilisation de déclencheurs, reportez-vous à la section [Chapitre 7 Déclenchement des pipelines dans vRealize Automation Code Stream](#).

La procédure suivante indique comment exécuter un pipeline à partir de la fiche de pipeline, afficher les exécutions, afficher les détails de l'exécution et utiliser les actions. Elle indique également comment libérer un pipeline afin de pouvoir l'ajouter à vRealize Automation Service Broker.

#### Conditions préalables

- Vérifiez qu'un ou plusieurs pipelines sont créés. Reportez-vous aux exemples de la section [Chapitre 5 Didacticiels d'utilisation de vRealize Automation Code Stream](#).

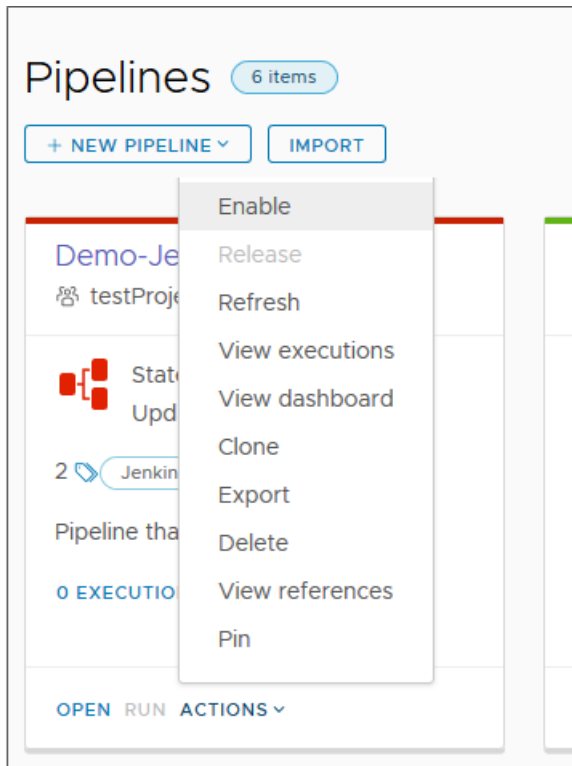


## Procédure

### 1 Activez votre pipeline.

Pour pouvoir exécuter ou libérer un pipeline, vous devez d'abord l'activer.

- a Cliquez sur **Pipelines**.
- b Sur votre fiche de pipeline, cliquez sur **Actions > Activer**.



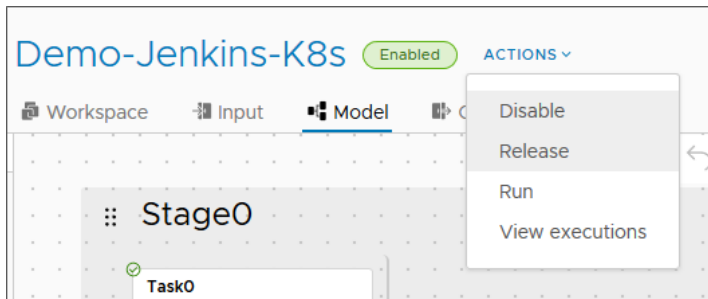
Vous pouvez également activer le pipeline lorsque vous vous trouvez dans celui-ci. Si le pipeline est déjà activé, l'option **Exécuter** est active et le menu **Actions** affiche **Désactiver**.

## 2 (Facultatif) Publiez votre pipeline.

Si vous souhaitez mettre à disposition votre pipeline en tant qu'élément du catalogue dans vRealize Automation Service Broker, vous devez le publier dans vRealize Automation Code Stream.

- a Cliquez sur **Pipelines**.
- b Sur votre fiche de pipeline, cliquez sur **Actions > Publier**.

Vous pouvez également publier le pipeline lorsque vous vous trouvez dans celui-ci.



Après avoir publié le pipeline, vous ouvrez vRealize Automation Service Broker pour ajouter le pipeline en tant qu'élément du catalogue et l'exécuter. Reportez-vous à la section [Ajouter des pipelines vRealize Automation Code Stream au catalogue vRealize Automation Service Broker](#).

**Note** Si le pipeline nécessite plus de 120 minutes pour s'exécuter, fournissez un temps d'exécution approximatif en tant que valeur de délai d'expiration de la demande. Pour définir ou vérifier le délai d'expiration de la demande d'un projet, ouvrez vRealize Automation Service Broker en tant qu'administrateur et sélectionnez **Infrastructure > Projets**. Cliquez sur le nom de votre projet, puis cliquez sur **Provisionnement**.

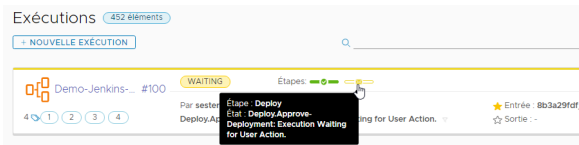
Si la valeur de délai d'expiration de la demande n'est pas définie, une exécution qui nécessite plus de 120 minutes pour s'exécuter apparaît comme ayant échoué avec une erreur d'expiration de la demande de rappel. Toutefois, l'exécution du pipeline n'est pas affectée.

- 3 Sur la fiche de pipeline, cliquez sur **Exécuter**.
- 4 Pour afficher le pipeline au fil de son exécution, cliquez sur **Exécutions**.

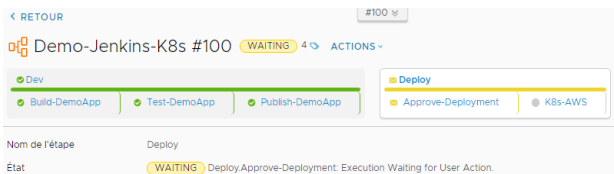
Le pipeline exécute chaque étape dans l'ordre, et l'exécution du pipeline affiche une icône d'état pour chaque étape. Si le pipeline inclut une tâche d'opération utilisateur, un utilisateur doit approuver la tâche pour que le pipeline continue de s'exécuter. Lorsqu'une tâche d'opération utilisateur est utilisée, l'exécution du pipeline s'interrompt jusqu'à ce que l'utilisateur requis approuve la tâche.

Par exemple, vous pouvez utiliser la tâche d'opération utilisateur pour approuver le déploiement de code dans un environnement de production.

Si le délai d'expiration de la tâche Opération de l'utilisateur est défini en jours, heures ou minutes, l'utilisateur requis doit approuver le pipeline avant l'expiration de la tâche. Sinon, le pipeline échoue comme prévu.



- 5 Pour savoir quelle étape du pipeline est en attente d'approbation de l'utilisateur, cliquez sur l'icône d'état de l'étape.

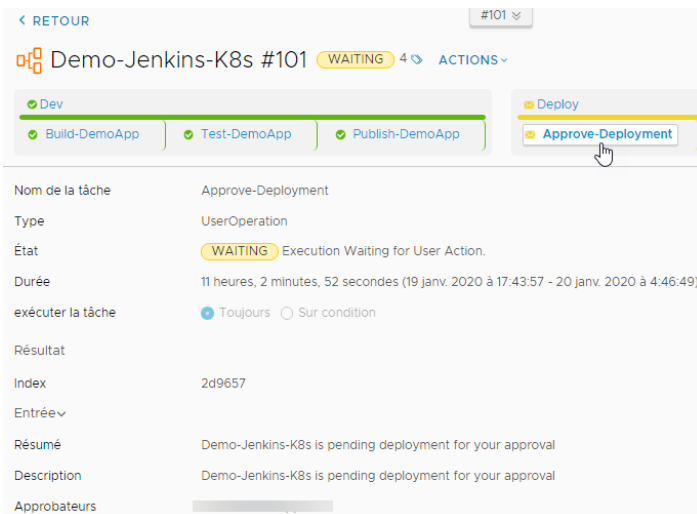


- 6 Pour afficher les détails de la tâche, cliquez sur celle-ci.

Une fois que l'utilisateur requis a approuvé la tâche, un utilisateur disposant du rôle approprié doit reprendre le pipeline. Pour connaître les rôles requis, reportez-vous à [Gestion de l'accès et des approbations utilisateur dans vRealize Automation Code Stream](#).

Si une exécution échoue, vous devez identifier et corriger la cause de l'échec. Ensuite, accédez à l'exécution, puis cliquez sur **Actions > Exécuter à nouveau**.

Vous pouvez reprendre les exécutions de pipeline principal et les exécutions imbriquées.



- 7 À partir de l'exécution du pipeline, vous pouvez cliquer sur **Actions** pour afficher le pipeline et sélectionner une action telle que **Pause**, **Annuler** et plus encore. Lorsqu'une exécution de pipeline est en cours, si vous êtes administrateur vous pouvez supprimer ou synchroniser l'exécution du pipeline. Si vous êtes un utilisateur non-administrateur, vous pouvez supprimer un pipeline en cours d'exécution.

- 8 Pour naviguer facilement entre les exécutions et afficher les détails d'une tâche, cliquez sur **Exécutions**, puis cliquez sur une exécution de pipeline. Ensuite, cliquez sur l'onglet en haut de l'exécution, puis sélectionnez une exécution.



### Résultats

Félicitations ! Vous avez exécuté un pipeline, examiné l'exécution du pipeline et affiché une tâche d'opération utilisateur qui exigeait une approbation pour que le pipeline continue de s'exécuter. Vous avez également utilisé le menu **Actions** dans l'exécution du pipeline pour revenir au modèle de pipeline afin d'y apporter toute modification requise.

### Étape suivante

Pour plus d'informations sur l'utilisation de vRealize Automation Code Stream pour automatiser votre cycle de publication logicielle, reportez-vous à la section [Chapitre 5 Didacticiels d'utilisation de vRealize Automation Code Stream](#).

## Types de tâche disponibles dans vRealize Automation Code Stream

Vous pouvez configurer votre pipeline pour effectuer certaines actions en y ajoutant des types de tâche spécifiques. Chaque type de tâche s'intègre à une autre application pour permettre à votre pipeline d'accomplir ce pour quoi vous l'avez conçu.

Si vous avez besoin d'extraire des artefacts d'un référentiel pour le déploiement, d'exécuter un script distant ou de demander l'approbation d'un membre de l'équipe pour que votre pipeline s'exécute, vRealize Automation Code Stream fournit le type de tâche qu'il vous faut.

Avant d'utiliser ce type de tâche dans votre pipeline, vérifiez que le point de terminaison correspondant est disponible.

Tableau 3-2. Obtenir une approbation ou définir un point de décision

Type de tâche	Effets	Exemples et détails
<b>Opération de l'utilisateur</b>	Active une approbation requise qui contrôle le moment où un pipeline s'exécute et doit s'arrêter pour approbation.	Reportez-vous à la section <a href="#">Comment exécuter un pipeline et afficher les résultats</a> , et <a href="#">Gestion de l'accès et des approbations utilisateur</a> dans vRealize Automation Code Stream.
<b>Condition</b>	Ajoute un point de décision qui détermine si le pipeline continue à s'exécuter, ou s'arrête, en fonction des expressions de condition. Lorsque la condition est true, le pipeline exécute des tâches successives. Lorsqu'elle est false, le pipeline s'arrête.	Reportez-vous à la section <a href="#">Utilisation des liaisons de variables dans une tâche de condition</a> pour l'exécution ou l'arrêt d'un pipeline dans vRealize Automation Code Stream.

Tableau 3-3. Automatiser l'intégration et le déploiement continus

Type de tâche	Effets	Exemples et détails
<b>Modèle de cloud</b>	Déploie un modèle de cloud d'automatisation à partir de GitHub et provisionne une application, puis automatise l'intégration continue et la livraison continue (CICD) de ce modèle de cloud pour votre déploiement.	<p>Reportez-vous à la section <a href="#">Automatisation de la publication d'une application déployée à partir d'un modèle de cloud YAML dans vRealize Automation Code Stream</a>.</p> <p>Lorsque vous sélectionnez <b>Créer</b> ou <b>Mettre à jour</b> dans la tâche de modèle de cloud et que vous sélectionnez <b>Modèle de cloud</b> et <b>Version</b>, les paramètres du modèle de cloud s'affichent. Vous pouvez ajouter ces éléments, qui adaptent les liaisons de variables aux zones de texte d'entrée dans la tâche de modèle de cloud :</p> <ul style="list-style-type: none"> <li>■ Entier</li> <li>■ Chaîne d'énumération</li> <li>■ Valeur booléenne</li> <li>■ Variable de tableau</li> </ul> <p>Lorsque vous utilisez une liaison de variables dans les champs d'entrée, vous devez tenir compte des exceptions suivantes. Pour les énumérations, vous devez sélectionner une valeur d'énumération à partir d'un ensemble fixe. Pour les valeurs booléennes, vous devez entrer la valeur dans la zone de texte saisie.</p> <p>Le paramètre du modèle de cloud apparaît dans la tâche de modèle de cloud lorsqu'un modèle de cloud de vRealize Automation Cloud Assembly inclut des variables d'entrée. Par exemple, si un modèle de cloud dispose d'un type d'entrée <code>Integer</code>, vous pouvez entrer l'entier directement ou sous la forme d'une variable à l'aide d'une liaison de variables.</p>
<b>CI</b>	Permet l'intégration continue de votre code dans votre pipeline en extrayant une image de build Docker à partir d'un point de terminaison de registre et en la déployant sur un cluster Kubernetes.	Reportez-vous à la section <a href="#">Planification d'une build native CICD dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent</a> .
<b>Personnalisé</b>	Intègre vRealize Automation Code Stream à vos propres outils de génération, de test et de déploiement.	Reportez-vous à la section <a href="#">Intégration d'outils de génération, de test et de déploiement propres avec vRealize Automation Code Stream</a> .

Tableau 3-3. Automatiser l'intégration et le déploiement continu (suite)

Type de tâche	Effets	Exemples et détails
Kubernetes	Automatisez le déploiement de vos applications logicielles sur des clusters Kubernetes sur AWS.	Reportez-vous à la section <a href="#">Automatisation de la publication d'une application dans vRealize Automation Code Stream sur un cluster Kubernetes</a> .
Pipeline	Imbrique un pipeline dans un pipeline principal. Lorsqu'un pipeline est imbriqué, il se comporte comme une tâche dans le pipeline principal.  Dans l'onglet Tâche du pipeline principal, vous pouvez facilement accéder au pipeline imbriqué en cliquant sur le lien y donnant accès. Le pipeline imbriqué s'ouvre dans un nouvel onglet du navigateur.	Pour rechercher des pipelines imbriqués dans <b>Exécutions</b> , entrez <b>imbriqué</b> dans la zone de recherche.

Tableau 3-4. Intégrer des applications de développement, de test et de déploiement

Type de tâche	Effets	Exemples et détails
Bamboo	Interagit avec un serveur d'intégration continue (CI) Bamboo, qui génère, teste et intègre en permanence des logiciels en préparation du déploiement, et déclenche des builds de code lorsque les développeurs valident les modifications. Il expose les emplacements d'artefact que la build Bamboo produit afin que la tâche puisse générer les paramètres que d'autres tâches utiliseront pour la génération et le déploiement.	Connectez-vous à un point de terminaison de serveur Bamboo et démarrez un plan de build Bamboo à partir de votre pipeline.
Jenkins	Déclenche des tâches Jenkins qui génèrent et testent votre code source, exécute des cas de test et peut utiliser des scripts personnalisés.	Reportez-vous à la section <a href="#">Intégration de vRealize Automation Code Stream à Jenkins</a> .
TFS	Vous permet de connecter votre pipeline à Team Foundation Server pour gérer et appeler des projets de build, y compris des tâches configurées qui génèrent et testent votre code.	vRealize Automation Code Stream prend en charge Team Foundation Server 2013 et 2015.
vRO	Étend la capacité de vRealize Automation Code Stream en exécutant des workflows prédéfinis ou personnalisés dans vRealize Orchestrator.	Reportez-vous à la section <a href="#">Intégration de vRealize Automation Code Stream à vRealize Orchestrator</a> .

Tableau 3-5. Intégrer d'autres applications via une API

Type de tâche	Effets	Exemples et détails
REST	Intègre vRealize Automation Code Stream à d'autres applications qui utilisent une REST API afin de pouvoir développer et fournir en continu des applications logicielles qui interagissent entre elles.	Reportez-vous à la section <a href="#">Utilisation d'une API REST pour intégrer vRealize Automation Code Stream à d'autres applications</a> .
Interrogation	Appelle une REST API et l'interroge jusqu'à ce que la tâche de pipeline réponde aux critères de sortie et se termine.	Reportez-vous à la section <a href="#">Utilisation d'une API REST pour intégrer vRealize Automation Code Stream à d'autres applications</a> .

Tableau 3-6. Exécuter des scripts distants et définis par l'utilisateur

Type de tâche	Effets	Exemples et détails
PowerShell	<p>Permet au type de tâche de script PowerShell d'exécuter des commandes de script sur un hôte distant. Par exemple, un script peut automatiser des tâches de test et exécuter des types administratifs de commandes.</p> <p>Le script peut être distant ou défini par l'utilisateur. Il peut se connecter via HTTP ou HTTPS, et peut utiliser TLS.</p> <p>Le service nommé <code>winrm</code> doit être configuré sur l'hôte Windows et <code>winrm</code> doit être configuré pour <code>MaxShellsPerUser</code> et <code>MaxMemoryPerShellMB</code>.</p>	<p>Lorsque vous configurez <code>MaxShellsPerUser</code> et <code>MaxMemoryPerShellMB</code> :</p> <ul style="list-style-type: none"> <li>■ La valeur acceptable pour <code>MaxShellsPerUser</code> est de 500 pour 50 pipelines simultanés, avec 5 tâches PowerShell par pipeline. Pour définir la valeur, exécutez : <code>winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'</code></li> <li>■ La valeur de mémoire acceptable pour <code>MaxMemoryPerShellMB</code> est de 2 048. Pour définir la valeur, exécutez : <code>winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'</code></li> </ul> <p>Le script écrit la sortie dans un fichier de réponse qu'un autre pipeline peut utiliser.</p>
SSH	<p>Permet au type de tâche de script shell bash d'exécuter des commandes de script sur un hôte distant. Par exemple, un script peut automatiser des tâches de test et exécuter des types administratifs de commandes.</p> <p>Le script peut être distant ou défini par l'utilisateur. Il peut se connecter via HTTP ou HTTPS, et nécessite une clé privée ou un mot de passe.</p> <p>Le service SSH doit être configuré sur l'hôte Linux et la configuration SSHD de <code>MaxSessions</code> doit être définie sur 50.</p>	<p>Le script peut être distant ou défini par l'utilisateur. Par exemple, un script peut ressembler à ce qui suit :</p> <pre>message="Hello World" echo \$message</pre> <p>Le script écrit la sortie dans un fichier de réponse qu'un autre pipeline peut utiliser.</p>



## Utilisation des liaisons de variables dans les pipelines vRealize Automation Code Stream

La liaison d'une tâche de pipeline signifie que vous créez une dépendance pour la tâche lorsque le pipeline s'exécute. Vous pouvez créer une liaison pour une tâche de pipeline de plusieurs manières. Vous pouvez lier une tâche à une autre tâche, la lier à une variable et une expression ou la lier à une condition.

### Application de liaisons dollar aux variables de modèle de cloud dans une tâche de modèle de cloud

Vous pouvez appliquer des liaisons dollar aux variables de modèle de cloud dans une tâche de modèle de cloud de pipeline vRealize Automation Code Stream. La façon dont vous modifiez les variables dans vRealize Automation Code Stream dépend de la façon dont les propriétés de la variable sont codées dans le modèle de cloud.

Si vous avez besoin d'utiliser des liaisons dollar dans une tâche de modèle de cloud, mais que la version actuelle du modèle de cloud que vous utilisez dans la tâche de modèle de cloud ne l'autorise pas, modifiez le modèle de cloud dans vRealize Automation Cloud Assembly et déployez une nouvelle version. Ensuite, utilisez la nouvelle version du modèle de cloud dans votre tâche de modèle de cloud et ajoutez les liaisons dollar aux emplacements souhaités.

Vous devez disposer des autorisations appropriées pour appliquer des liaisons dollar aux types de propriété fournis par le modèle de cloud vRealize Automation Cloud Assembly.

- Vous devez avoir le même rôle que la personne qui a créé le déploiement du modèle de cloud dans vRealize Automation Cloud Assembly.
- La personne qui modélise le pipeline et la personne qui exécute le pipeline peuvent être deux utilisateurs différents et peuvent avoir des rôles différents.
- Si un développeur dispose du rôle d'exécuteur de vRealize Automation Code Stream et modélise le pipeline, le développeur doit également disposer du même rôle dans vRealize Automation Cloud Assembly que la personne qui a déployé le modèle de cloud. Par exemple, le rôle requis peut être Administrateur de vRealize Automation Cloud Assembly.
- Seule la personne qui modélise le pipeline a l'autorisation de créer le pipeline et le déploiement.

Pour utiliser un jeton d'API dans la tâche de modèle de cloud :

- La personne qui modélise le pipeline peut donner un jeton d'API à un autre utilisateur disposant du rôle Exécuteur de vRealize Automation Code Stream. Ensuite, lorsque l'exécuteur exécute le pipeline, il utilise le jeton d'API et les informations d'identification créées par le jeton d'API.
- Lorsqu'un utilisateur entre un jeton d'API dans la tâche de modèle de cloud, il crée les informations d'identification nécessaires pour l'exécution du pipeline.
- Pour chiffrer la valeur du jeton d'API, cliquez sur **Créer une variable**.

- Si vous ne créez pas de variable pour le jeton d'API et que vous l'utilisez dans la tâche de modèle de cloud, la valeur du jeton d'API s'affiche en texte brut.

Pour appliquer des liaisons dollar à des variables de modèle de cloud dans une tâche de modèle de cloud, procédez comme suit.

Démarrez avec un modèle de cloud dont les propriétés de variable d'entrée sont définies, telles que `integerVar`, `stringVar`, `flavorVar`, `BooleanVar`, `objectVar` et `arrayVar`. Les propriétés d'image sont définies dans la section `resources`. Les propriétés peuvent être similaires à ce qui suit dans le code du modèle de cloud :

```
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}'
```

Vous pouvez utiliser les variables à symbole dollar (\$) pour `image` et `flavor`. Par exemple :

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

Pour utiliser un modèle de cloud dans un pipeline vRealize Automation Code Stream et y ajouter des liaisons dollar, procédez comme suit.

- 1 Dans vRealize Automation Code Stream, cliquez sur **Pipelines > Canevas vide**.
- 2 Ajoutez une tâche de **modèle de cloud** au pipeline.
- 3 Dans la tâche de modèle de cloud, pour la **Source du modèle de cloud**, sélectionnez **Modèles de cloud Cloud Assembly**, entrez le nom du modèle de cloud et sélectionnez sa version.
- 4 Notez que vous pouvez entrer un jeton d'API qui peut fournir des informations d'identification pour que le pipeline s'exécute et que vous pouvez utiliser **Créer une variable** pour créer une variable qui chiffre le jeton d'API dans la tâche de modèle de cloud.
- 5 Dans le tableau **Paramètre et valeur** qui s'affiche, notez les valeurs des paramètres. La valeur par défaut pour `flavor` est `small` et la valeur par défaut pour `l'image` est `ubuntu`.
- 6 Supposons que vous devez modifier le modèle de cloud dans vRealize Automation Cloud Assembly. Par exemple :
  - a Définissez `flavor` pour utiliser une propriété de type `array`. vRealize Automation Cloud Assembly autorise les valeurs séparées par des virgules pour `flavor` lorsque le type est **array**.
  - b Cliquez sur **Déployer**.
  - c Sur la page Type de déploiement, entrez un nom de déploiement et sélectionnez la version du modèle de cloud.
  - d Sur la page Entrées de déploiement, vous pouvez définir une ou plusieurs valeurs pour `flavor`.
  - e Notez que la page Entrées de déploiement inclut toutes les variables définies dans votre code de modèle de cloud et qui apparaîtront comme définies dans le code du modèle de cloud. Par exemple : `Integer Var`, `String Var`, `Flavor Var`, `Boolean Var`, `Object Var` et `Array Var`. `String Var` et `Flavor Var` sont des valeurs de chaîne et `Boolean Var` est une case à cocher.
  - f Cliquez sur **Déployer**.
- 7 Dans vRealize Automation Code Stream, sélectionnez la nouvelle version du modèle de cloud et entrez des valeurs dans la table **Paramètre et valeur**. Les modèles de cloud prennent en charge les types de paramètres suivants et la capacité pour vRealize Automation Code Stream à autoriser les liaisons de variables dollar à ces modèles de cloud, ce qui entraîne de légères

différences dans l'interface utilisateur de la tâche du modèle de cloud vRealize Automation Code Stream et l'interface du modèle de cloud vRealize Automation Cloud Assembly. Selon la façon dont un modèle de cloud est codé, vous pouvez être autorisé ou non à entrer des valeurs dans la tâche.

- a Pour **flavorVar**, si le modèle de cloud a défini le type comme chaîne ou tableau, entrez une chaîne ou un tableau de valeurs séparées par des virgules. Exemple de tableau : **test, test**.
- b Pour **BooleanVar**, dans le menu déroulant, sélectionnez **true** ou **false**. Pour utiliser une liaison de variables, entrez **\$** et sélectionnez une liaison de variables dans la liste.

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

Output Parameter

status deploymentName deploymentId deploymentName

var  
input  
comments  
requestBy  
executionIndex  
executionId  
executionUrl  
name  
description  
Stage0

- c Pour **objectVar**, entrez la valeur entre accolades et guillemets droits en suivant ce format : **{"bkix" : "bkix" : }**.
  - d La variable **objectVar** est transmise au modèle de cloud et peut être utilisée de différentes façons selon le modèle de cloud. Elle autorise un format de chaîne pour un objet JSON et vous pouvez ajouter des paires clé-valeur sous forme de valeurs séparées par des virgules dans la table clé-valeur. Vous pouvez entrer du texte brut pour un objet JSON ou une paire clé-valeur comme format de chaîne normal pour JSON.
  - e Pour **arrayVar**, entrez la valeur d'entrée séparée par des virgules sous la forme d'un tableau en suivant ce format : **["1" , "2"]**.
- 8 Dans le pipeline, vous pouvez lier un paramètre d'entrée à un tableau.
- a Cliquez sur l'onglet **Entrée**.
  - b Entrez un nom pour l'entrée. Par exemple, **arrayInput**.
  - c Dans la table **Paramètre et valeur**, cliquez sur **arrayVar** et entrez **\${input.arrayInput}**.
  - d Après avoir enregistré et activé le pipeline, lors de son exécution, vous devez fournir une valeur d'entrée de tableau. Par exemple, entrez **["1" , "2"]**, puis cliquez sur **Exécuter**.

Vous pouvez désormais utiliser les liaisons de variables à symbole dollar (\$) dans un modèle de cloud pour une tâche de modèle de cloud de pipeline vRealize Automation Code Stream.

## Transmission d'un paramètre à un pipeline lors de son exécution

Vous pouvez ajouter des paramètres d'entrée à votre pipeline pour que vRealize Automation Code Stream les transmette au pipeline. Ensuite, lorsque le pipeline s'exécute, un utilisateur doit entrer la valeur du paramètre d'entrée. Vous pouvez ajouter des paramètres de sortie à votre pipeline afin que les tâches de pipeline puissent utiliser la valeur de sortie d'une tâche. vRealize Automation Code Stream prend en charge l'utilisation de paramètres de plusieurs manières qui répondent à vos propres besoins en matière de pipeline.

Par exemple, pour inviter un utilisateur à saisir l'URL de son serveur Git lorsqu'un pipeline avec une tâche REST s'exécute, vous pouvez lier la tâche REST à l'URL d'un serveur Git.

Pour créer la liaison de variables, ajoutez une variable de liaison d'une URL à la tâche REST. Lorsque le pipeline s'exécute et qu'il atteint la tâche REST, un utilisateur doit entrer son URL vers le serveur Git. Voici comment créer la liaison :

- 1 Dans votre pipeline, cliquez sur l'onglet **Entrée**.
- 2 Pour définir le paramètre, dans **Paramètres d'insertion automatique**, cliquez sur **Git**.  
La liste des paramètres Git s'affiche et inclut **GIT\_SERVER\_URL**. Si vous avez besoin d'utiliser une valeur par défaut pour l'URL du serveur Git, vous devez modifier ce paramètre.
- 3 Cliquez sur **Modèle**, puis cliquez sur votre tâche REST.
- 4 Dans l'onglet **Tâche**, dans la zone **URL**, entrez \$, puis sélectionnez **entrée** et **GIT\_SERVER\_URL**.

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

Task name ⓘ \*   Task3

Type \*   REST

Continue on failure   ☐

Execute task   ☒ Always   ☐ On condition

REST Request

Action \*   GET

URL \$ \*   \${input.GIT\_SERVER\_URL}

Agent endpoint

Headers

Output Parameters

status   responseHeaders   responseBody   responseJson   responseCode

L'entrée ressemble à ce qui suit : `${input.GIT_SERVER_URL}`

- 5 Pour vérifier l'intégrité de la liaison de variables pour la tâche, cliquez sur **Valider la tâche**.  
vRealize Automation Code Stream indique que la tâche a été correctement validée.
- 6 Lorsque le pipeline exécute la tâche REST, un utilisateur doit entrer l'URL du serveur Git. Dans le cas contraire, la tâche ne termine pas son exécution.

## Liaison de deux tâches de pipeline par la création de paramètres d'entrée et de sortie

Lorsque vous liez deux tâches entre elles, vous ajoutez une variable de liaison à la configuration d'entrée de la tâche réceptrice. Ensuite, lorsque le pipeline s'exécute, un utilisateur remplace la variable de liaison par l'entrée requise.

Pour lier des tâches de pipeline entre elles, vous utilisez la variable de signe dollar (\$) dans les paramètres d'entrée et de sortie. Voici comment procéder.

Supposons que vous ayez besoin de votre pipeline pour appeler une URL dans une tâche REST et pour envoyer une réponse. Pour ce faire, vous devez inclure les paramètres d'entrée et de sortie dans votre tâche REST. Vous avez également besoin d'un utilisateur pour approuver la tâche, de sorte que vous incluez également une tâche Opérations d'utilisateur pour qu'un autre utilisateur l'approuve lorsque le pipeline s'exécute. Cet exemple vous présente comment utiliser des expressions dans les paramètres d'entrée et de sortie et faire en sorte que le pipeline attende l'approbation sur la tâche.

- 1 Dans votre pipeline, cliquez sur l'onglet **Entrée**.

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

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

- 2 Laissez les **Paramètres d'insertion automatique** sur **Aucun**.
- 3 Cliquez sur **Ajouter**, entrez le nom, la valeur et la description du paramètre, puis cliquez sur **OK**. Par exemple :
  - a Entrez un nom d'URL.
  - b Entrez la valeur : {Stage0.Task3.input.http://www.docs.vmware.com}
  - c Entrez une description.
- 4 Cliquez sur l'onglet **Sortie**, cliquez sur **Ajouter**, puis entrez le nom et le mappage du paramètre de sortie.

### Add Pipeline Output Parameter

Name \*

Reference \$ \* 

responseHeaders  
responseBody  
responseJson  
responseCode

- Entrez un nom de paramètre de sortie unique.
- Cliquez dans la zone de **Référence** et entrez \$.
- Entrez le mappage de sortie de la tâche en sélectionnant les options lorsqu'elles s'affichent. Sélectionnez **Stage0**, sélectionnez **Task3**, sélectionnez **sortie** et sélectionnez **responseCode**. Ensuite, cliquez sur **OK**.

rest-ix-1 Enabled ACTIONS ▾

Workspace
Input
Model
Output

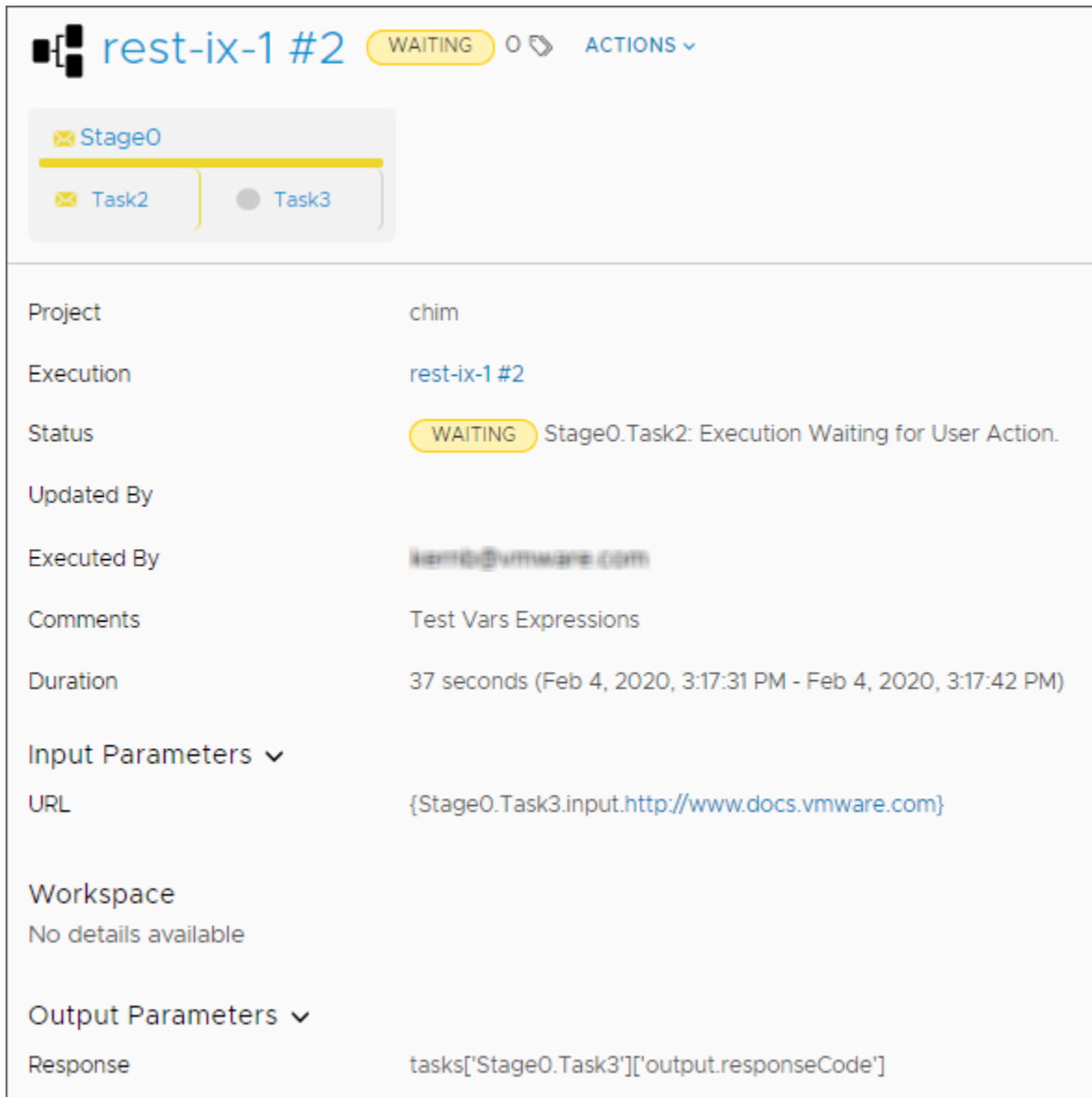
#### Output Parameters ⓘ

ADD

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

- Enregistrez votre pipeline.
- Dans le menu **Actions**, cliquez sur **Exécuter**.
- Cliquez sur **Actions > Afficher les exécutions**.
- Cliquez sur l'exécution et reportez-vous aux paramètres d'entrée et de sortie que vous avez définis.





**rest-ix-1 #2** WAITING 0 ACTIONS ▾

Stage0

Task2 Task3

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

- 9 Pour approuver le pipeline, cliquez sur **Opérations de l'utilisateur** et affichez la liste des approbations dans l'onglet **Éléments actifs**. Sinon, restez dans Exécutions, cliquez sur la tâche, puis sur **Approuver**.
- 10 Pour activer les boutons **Approuver** et **Refuser**, cochez la case en regard de l'exécution.
- 11 Pour afficher les détails, développez la flèche déroulante.
- 12 Pour approuver la tâche, cliquez sur **APPROUVER**, entrez un motif et cliquez sur **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 Cliquez sur **Exécutions** et observez la suite de l'exécution du pipeline.

**Executions** 3,347 items GUIDED SETUP

+ NEW EXECUTION

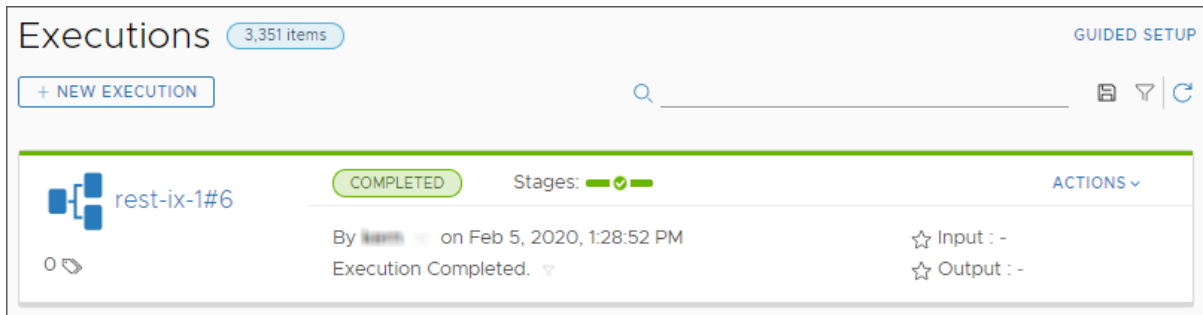
rest-i... #3 RUNNING Stages: 0 ACTIONS

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

**RUNNING** ☆ Output : -

Comments: Testing

- 14 Si le pipeline échoue, corrigez les erreurs, puis enregistrez le pipeline et exécutez-le à nouveau.



## En savoir plus sur les variables et les expressions

Pour obtenir des détails sur l'utilisation de variables et d'expressions lorsque vous liez des tâches de pipeline, reportez-vous à la section [Quelles variables et expressions peuvent être utilisées lors de la liaison de tâches de pipeline dans vRealize Automation Code Stream](#).

Pour savoir comment utiliser la sortie de la tâche de pipeline avec une liaison de variables de condition, reportez-vous à la section [Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline dans vRealize Automation Code Stream](#).

## Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline dans vRealize Automation Code Stream

Vous pouvez faire en sorte que la sortie d'une tâche de votre pipeline détermine si le pipeline s'exécute ou s'arrête en fonction d'une condition que vous indiquez. Pour conditionner la réussite ou l'échec du pipeline en fonction de la sortie de la tâche, vous utilisez le type de tâche Condition.

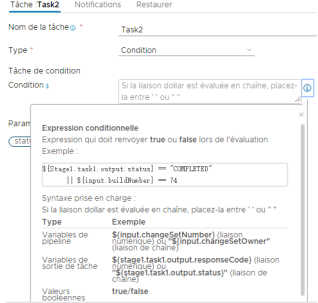
Vous utilisez le type de tâche **Condition** comme point de décision dans votre pipeline. En utilisant la tâche Condition avec une expression de condition que vous indiquez, vous pouvez évaluer toutes les propriétés de votre pipeline, de vos étapes et de vos tâches.

Le résultat de la tâche Condition détermine si la tâche suivante du pipeline s'exécute.

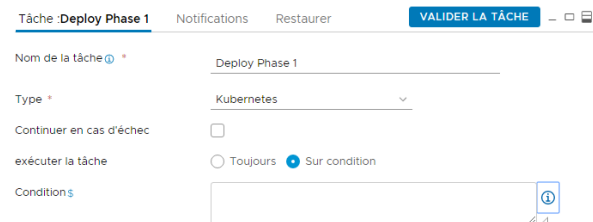
- Une condition true permet au pipeline de continuer à s'exécuter.
- Une condition false arrête le pipeline.

Pour obtenir des exemples d'utilisation de la valeur de sortie d'une tâche comme entrée de la tâche suivante en liant les tâches avec une tâche Condition, reportez-vous à la section [Utilisation des liaisons de variables dans les pipelines vRealize Automation Code Stream](#).

Tableau 3-7. Association de la tâche Condition et de son expression de condition au pipeline

Type de tâche Condition...	Éléments influencés	Effets
Tâche Condition	Pipeline	Le type de tâche <b>Condition</b> détermine si le pipeline s'exécute ou s'arrête à ce stade, selon la sortie de la tâche, true ou false.
Expression de condition	Sortie de la tâche Condition	<p>Lorsque le pipeline s'exécute, l'expression de condition que vous incluez dans la tâche <b>Condition</b> génère un état de sortie true ou false. Par exemple, une expression de condition peut nécessiter que l'état de sortie de la tâche Condition soit <i>Terminé</i> ou que le numéro de build 74 soit utilisé.</p> <p>L'expression de condition s'affiche dans l'onglet Tâche, dans le type de tâche Condition.</p> 

Le type de tâche **Condition** diffère en termes de fonction et de comportement du paramètre **Sur condition** d'autres types de tâche.



Dans d'autres types de tâche, le paramètre **Sur condition** :

- Détermine si la tâche actuelle s'exécute, plutôt que des tâches successives, en fonction de l'évaluation de son expression de condition préalable de true ou false. L'expression de condition du paramètre Sur condition génère un état de sortie true ou false pour la tâche actuelle lorsque le pipeline s'exécute.
- S'affiche dans l'onglet Tâche avec sa propre expression de condition.

Cet exemple utilise la tâche Condition.

### Conditions préalables

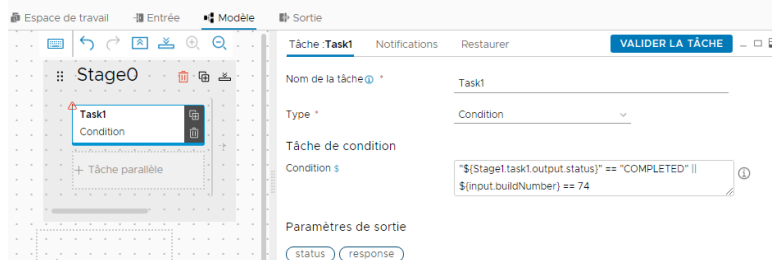
- Vérifiez qu'un pipeline existe, et qu'il inclut des étapes et des tâches.

### Procédure

- 1 Dans votre pipeline, déterminez le point de décision au niveau duquel la tâche Condition doit s'afficher.

- 2 Ajoutez la tâche Condition avant la tâche qui dépend de son état de réussite ou d'échec.
- 3 Ajoutez une expression de condition à la tâche Condition.

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



- 4 Validez la tâche.
- 5 Enregistrez le pipeline, puis activez-le et exécutez-le.

### Résultats

Observez les exécutions de pipeline et notez si le pipeline continue à s'exécuter ou s'arrête au niveau de la tâche Condition.

### Étape suivante

Vous pouvez également utiliser le type de tâche Condition si vous décidez de restaurer le déploiement d'un pipeline. Par exemple, dans un pipeline de restauration, la tâche Condition permet à vRealize Automation Code Stream de marquer l'échec d'un pipeline en fonction de l'expression de condition et peut déclencher un flux de restauration unique pour différents types d'échec.

Pour restaurer un déploiement, reportez-vous à la section [Restauration d'un déploiement dans vRealize Automation Code Stream](#).

## Quelles variables et expressions peuvent être utilisées lors de la liaison de tâches de pipeline dans vRealize Automation Code Stream

Avec les variables et les expressions, vous pouvez utiliser des paramètres d'entrée et de sortie avec vos tâches de pipeline. Les paramètres que vous entrez lient votre tâche de pipeline à une ou plusieurs variables, expressions ou conditions, et déterminent le comportement du pipeline lors de son exécution.

Lorsque vous liez des tâches de pipeline, vous pouvez inclure des expressions par défaut et complexes afin que votre pipeline puisse exécuter des solutions de livraison de logiciel simples ou complexes. Pour créer les paramètres dans votre pipeline, cliquez sur l'onglet **Entrée** ou **Sortie** et ajoutez une variable en entrant le symbole dollar \$ et une expression. Par exemple, ce paramètre est utilisé comme une entrée de tâche qui appelle une URL : `${Stage0.Task3.input.URL}`.

Le format des liaisons de variables utilise des composants de syntaxe appelés portées et clés. `SCOPE` définit le contexte comme entrée ou sortie, et `KEY` définit les détails. Dans l'exemple de paramètre `${Stage0.Task3.input.URL}`, `input` est `SCOPE` et l'`URL` est `KEY`.

Les propriétés de sortie d'une tâche peuvent être résolues en n'importe quel nombre de niveaux imbriqués de liaison de variable.

Pour en savoir plus sur l'utilisation de liaisons de variables dans des pipelines, consultez [Utilisation des liaisons de variables dans les pipelines vRealize Automation Code Stream](#).

## Utilisation d'expressions en dollars avec des étendues et des clés pour lier des tâches de pipeline

Vous pouvez lier des tâches de pipeline en utilisant des expressions dans des variables à symbole de dollar. Vous entrez des expressions en tant que `${SCOPE.KEY.<PATH>}`.

Dans chaque expression, `SCOPE` est le contexte que vRealize Automation Code Stream utilise pour déterminer le comportement d'une tâche de pipeline. La portée recherche une `KEY` qui définit les détails de l'action effectuée par la tâche. Lorsque la valeur de `KEY` est un objet imbriqué, vous pouvez fournir un `PATH` facultatif.

Ces exemples décrivent `SCOPE` et `KEY`, et vous montrent comment les utiliser dans votre pipeline.

Tableau 3-8. Utilisation de SCOPE et de KEY

SCOPE	Objet de l'expression et exemple	KEY	Utilisation de SCOPE et KEY dans votre pipeline
input	Propriétés d'entrée d'un pipeline : <code>\${input.input1}</code>	Nom de la propriété d'entrée	<p>Pour faire référence à la propriété d'entrée d'un pipeline dans une tâche, utilisez ce format :</p> <pre>tasks:   mytask:     type: REST     input:       url: \$       {input.url}     action: get</pre> <pre>input:   url: https://   www.vmware.com</pre>
output	Propriétés de sortie d'un pipeline : <code>\${output.output1}</code>	Nom de la propriété de sortie	<p>Pour faire référence à une propriété de sortie afin que le pipeline puisse envoyer une notification, utilisez ce 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>

Tableau 3-8. Utilisation de SCOPE et de KEY (suite)

SCOPE	Objet de l'expression et exemple	KEY	Utilisation de SCOPE et KEY dans votre pipeline
task input	Entrée d'une tâche : \$ {MY_STAGE.MY_TASK.input. SOMETHING}	Indique l'entrée d'une tâche dans une notification	<p>Lorsqu'un travail Jenkins démarre, il peut faire référence au nom de la tâche déclenchée à partir de l'entrée de la tâche. Dans ce cas, envoyez une notification à l'aide de ce 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>
task output	Sortie d'une tâche : \$ {MY_STAGE.MY_TASK.output .SOMETHING}	Indique la sortie d'une tâche dans une tâche suivante	<p>Pour faire référence à la sortie de la tâche de pipeline 1 dans la tâche 2, utilisez ce format :</p> <pre> taskOrder:   - task1   - task2 tasks:   task1:     type: REST     input:       action: get       url: https://       www.example.org/api       /status   task2:     type: REST     input:       action: post       url: https://       status.internal.exa       mple.org/api/       activity       payload: \$       {MY_STAGE.task1.out       put.responseBody} </pre>



Tableau 3-8. Utilisation de SCOPE et de KEY (suite)

SCOPE	Objet de l'expression et exemple	KEY	Utilisation de SCOPE et KEY dans votre pipeline
var	Variable : <code>\${var.myVariable}</code>	Faire référence à une variable dans un point de terminaison	Pour faire référence à une variable secrète dans un point de terminaison pour mot de passe, utilisez ce format :  <pre>--- project: MyProject kind: ENDPOINT name: MyJenkinsServer type: jenkins properties:   url: https:// jenkins.example.com username: jenkinsUser password: \$ {var.jenkinsPassword}</pre>
var	Variable : <code>\${var.myVariable}</code>	Faire référence à une variable dans un pipeline	Pour faire référence à une variable dans une URL de pipeline, utilisez ce format :  <pre>tasks:   task1:     type: REST     input:       action: get       url: \$ {var.MY_SERVER_URL}</pre>
task status	État d'une tâche :  <pre>\$ {MY_STAGE.MY_TASK.status} \$ {MY_STAGE.MY_TASK.status Message}</pre>		
stage status	État d'une étape :  <pre>\${MY_STAGE.status} \$ {MY_STAGE.statusMessage}</pre>		

## Expressions par défaut

Vous pouvez utiliser des variables avec des expressions dans votre pipeline. Ce résumé inclut les expressions par défaut que vous pouvez utiliser.

Expression	Description
<code>\${comments}</code>	Commentaires fournis lorsque l'exécution a été demandée.
<code>\${duration}</code>	Durée de l'exécution du pipeline.
<code>\${endTime}</code>	Heure de fin de l'exécution du pipeline en UTC, si conclue.
<code>\${executedOn}</code>	Identique à l'heure de début, l'heure de début de l'exécution du pipeline en UTC.
<code>\${executionId}</code>	ID de l'exécution du pipeline.
<code>\${executionUrl}</code>	URL qui permet d'accéder à l'exécution du pipeline dans l'interface utilisateur.
<code>\${name}</code>	Nom du pipeline.
<code>\${requestBy}</code>	Nom de l'utilisateur qui a demandé l'exécution.
<code>\${stageName}</code>	Nom de l'étape actuelle, lorsqu'elle est utilisée dans l'étendue d'une étape.
<code>\${startTime}</code>	Heure de début de l'exécution du pipeline en UTC.
<code>\${status}</code>	État de l'exécution.
<code>\${statusMessage}</code>	Message d'état de l'exécution du pipeline.
<code>\${taskName}</code>	Nom de la tâche actuelle, lorsqu'elle est utilisée au niveau d'une entrée de tâche ou d'une notification.

## Utilisation de SCOPE et KEY dans les types de tâche de pipeline

Vous pouvez utiliser des expressions avec n'importe quel type de tâche de pipeline pris en charge. Utilisez ces exemples comme référence pour voir comment définir `SCOPE` et `KEY`, et vérifier la syntaxe. Ces exemples de code utilisent `MY_STAGE` et `MY_TASK` comme étape de pipeline et noms de tâches.

Pour en savoir plus sur les types de tâches disponibles, reportez-vous à [Types de tâche disponibles dans vRealize Automation Code Stream](#).

Tableau 3-9. Assemblage de tâches

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
<b>Opération de l'utilisateur</b>			
	Input	<p><code>summary</code> : résumé de la demande d'opération d'utilisateur</p> <p><code>description</code> : description de la demande d'opération d'utilisateur</p> <p><code>approvers</code> : liste des adresses e-mail de l'approbateur, dans lesquelles chaque entrée peut être une variable avec une virgule ou utiliser un point-virgule pour les e-mails séparés</p> <p><code>approverGroups</code> : liste des adresses des groupes d'approbateurs pour la plate-forme et l'identité</p> <p><code>sendemail</code> : envoie éventuellement une notification par e-mail lors de la demande ou de la réponse lorsqu'elle est définie sur true</p> <p><code>expirationInDays</code> : nombre de jours qui représente le délai d'expiration de la demande</p>	<pre> \${MY_STAGE.MY_TASK.input.summary} \${MY_STAGE.MY_TASK.input.description} \${MY_STAGE.MY_TASK.input.approvers} \$ {MY_STAGE.MY_TASK.input.approverGroups} \${MY_STAGE.MY_TASK.input.sendemail} \$ {MY_STAGE.MY_TASK.input.expirationInDays} </pre>
	Output	<p><code>index</code> : chaîne hexadécimale à 6 chiffres qui représente la demande</p> <p><code>respondedBy</code> : nom du compte de la personne qui a approuvé/refusé l'opération d'utilisateur</p> <p><code>respondedByEmail</code> : adresse e-mail de la personne qui a répondu</p> <p><code>comments</code> : commentaires fournis lors de la réponse</p>	<pre> \${MY_STAGE.MY_TASK.output.index} \${MY_STAGE.MY_TASK.output.respondedBy} \$ {MY_STAGE.MY_TASK.output.respondedByEmail} \${MY_STAGE.MY_TASK.output.comments} </pre>
<b>Condition</b>			
	Input	<p><code>condition</code> : condition à évaluer. Lorsque la condition est évaluée sur true, elle indique que la tâche est terminée, tandis que d'autres réponses font échouer la tâche</p>	<pre> \${MY_STAGE.MY_TASK.input.condition} </pre>
	Output	<p><code>result</code> : résultat lors de l'évaluation</p>	<pre> \${MY_STAGE.MY_TASK.output.response} </pre>

Tableau 3-10. Tâches de pipeline

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
Pipeline			
	Input	name : nom du pipeline à exécuter inputProperties : propriétés d'entrée à transmettre à l'exécution de pipeline imbriquée	<pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.inputProperties} # Faire référence à toutes les propriétés \$ {MY_STAGE.MY_TASK.input.inputProperties.inpu t1} # Faire référence à la valeur de input1 </pre>
	Output	executionStatus : état de l'exécution du pipeline executionIndex : index de l'exécution du pipeline outputProperties : propriétés de sortie de l'exécution du pipeline	<pre> \${MY_STAGE.MY_TASK.output.executionStatus} \${MY_STAGE.MY_TASK.output.executionIndex} \${MY_STAGE.MY_TASK.output.outputProperties} # Faire référence à toutes les propriétés \$ {MY_STAGE.MY_TASK.output.outputProperties.ou tput1} # Faire référence à la valeur d'output1 </pre>

Tableau 3-11. Automatiser les tâches d'intégration continue

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
CI			
	Input	steps : ensemble de chaînes, qui représentent les commandes à exécuter export : variables d'environnement à conserver après l'exécution des étapes artifacts : chemins d'artefacts à conserver dans le chemin partagé process : ensemble d'éléments de configuration pour le traitement de JUnit, JaCoCo, Checkstyle, FindBugs	<pre> \${MY_STAGE.MY_TASK.input.steps} \${MY_STAGE.MY_TASK.input.export} \${MY_STAGE.MY_TASK.input.artifacts} \${MY_STAGE.MY_TASK.input.process} \$ {MY_STAGE.MY_TASK.input.process[0].path } # Faire référence au chemin d'accès de la première configuration </pre>
	Output	exports : paire clé-valeur, qui représente les variables d'environnement exportées depuis l'entrée export artifacts : chemin d'artefacts correctement conservés processResponse : ensemble de résultats traités pour le process d'entrée	<pre> \${MY_STAGE.MY_TASK.output.exports} # Faire référence à toutes les exportations \$ {MY_STAGE.MY_TASK.output.exports.myvar} # Faire référence à la valeur de myvar \${MY_STAGE.MY_TASK.output.artifacts} \$ {MY_STAGE.MY_TASK.output.processRespons e} \$ {MY_STAGE.MY_TASK.output.processRespons e[0].result} # Résultat de la configuration du premier processus </pre>

Tableau 3-11. Automatiser les tâches d'intégration continue (suite)

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
Personnalisé			
	Input	name : nom de l'intégration personnalisée version : version de l'intégration personnalisée, publiée ou déconseillée properties : propriétés à envoyer à l'intégration personnalisée	<pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.version} \${MY_STAGE.MY_TASK.input.properties} # Faire référence à toutes les propriétés \$ {MY_STAGE.MY_TASK.input.properties.property1} # Faire référence à la valeur de property1 </pre>
	Output	properties : propriétés de sortie de la réponse d'intégration personnalisée	<pre> \${MY_STAGE.MY_TASK.output.properties} # Faire référence à toutes les propriétés \$ {MY_STAGE.MY_TASK.output.properties.property1} # Faire référence à la valeur de property1 </pre>

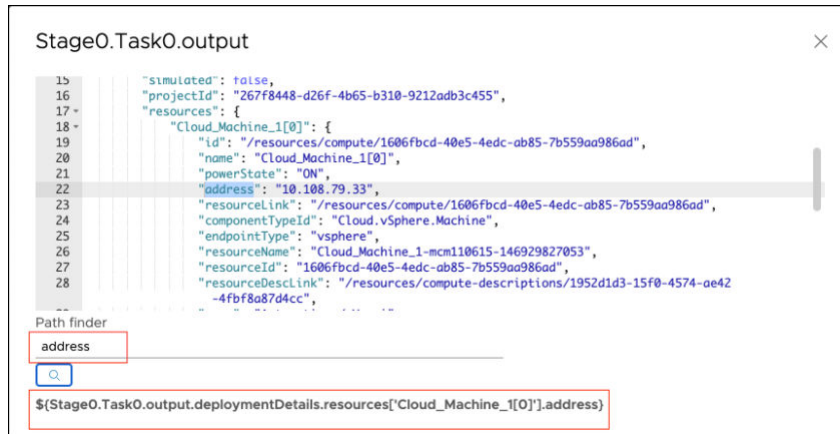
Tableau 3-12. Automatiser les tâches de déploiement continue : modèle de cloud

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
Modèle de cloud			
	Input	<p>action : un des éléments <b>createDeployment</b>, <b>updateDeployment</b>, <b>deleteDeployment</b>, <b>rollbackDeployment</b></p> <p>blueprintInputParams : utilisé pour les actions Créer un déploiement et Mettre à jour le déploiement</p> <p>allowDestroy : des machines peuvent être détruites lors du processus de déploiement de la mise à jour.</p> <p><b>CREATE_DEPLOYMENT</b></p> <ul style="list-style-type: none"> <li>■ blueprintName : nom du modèle de cloud</li> <li>■ blueprintVersion : version du modèle de cloud</li> </ul> <p>OU</p> <ul style="list-style-type: none"> <li>■ fileUrl : URL du modèle de cloud distant YAML, après la sélection d'un serveur GIT.</li> </ul> <p><b>UPDATE_DEPLOYMENT</b></p> <p>Une de ces combinaisons :</p> <ul style="list-style-type: none"> <li>■ blueprintName : nom du modèle de cloud</li> <li>■ blueprintVersion : version du modèle de cloud</li> </ul> <p>OU</p> <ul style="list-style-type: none"> <li>■ fileUrl : URL du modèle de cloud distant YAML, après la sélection d'un serveur GIT.</li> </ul> <p>-----</p> <ul style="list-style-type: none"> <li>■ deploymentId : ID du déploiement</li> </ul> <p>OU</p> <ul style="list-style-type: none"> <li>■ deploymentName : nom du déploiement</li> </ul> <p>-----</p> <p><b>DELETE_DEPLOYMENT</b></p> <ul style="list-style-type: none"> <li>■ deploymentId : ID du déploiement</li> </ul>	

Tableau 3-12. Automatiser les tâches de déploiement continue : modèle de cloud (suite)

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
		<p>OU</p> <ul style="list-style-type: none"> <li>■ deploymentName: nom du déploiement</li> </ul> <p><b>ROLLBACK_DEPLOYMENT</b></p> <p>Une de ces combinaisons :</p> <ul style="list-style-type: none"> <li>■ deploymentId: ID du déploiement</li> </ul> <p>OU</p> <ul style="list-style-type: none"> <li>■ deploymentName: nom du déploiement</li> </ul> <p>-----</p> <ul style="list-style-type: none"> <li>■ blueprintName: nom du modèle de cloud</li> <li>■ rollbackVersion: version à restaurer</li> </ul>	
	Output		<p>Paramètres pouvant être liés à d'autres tâches ou à la sortie d'un pipeline :</p> <ul style="list-style-type: none"> <li>■ Le nom du déploiement est accessible en tant que <code>\$ {Stage0.Task0.output.deploymentName}</code></li> <li>■ L'ID de déploiement est accessible en tant que <code>\$ {Stage0.Task0.output.deploymentId}</code></li> <li>■ Les détails du déploiement constituent un objet complexe et les détails internes sont accessibles à l'aide des résultats JSON.</li> </ul> <p>Pour accéder à n'importe quelle propriété, utilisez l'opérateur point pour suivre la hiérarchie JSON. Par exemple, pour accéder à l'adresse de la ressource Cloud_Machine_1[0], la liaison \$ est :</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}</pre> <p>De même, pour le type, la liaison \$ est :</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}</pre> <p>Dans l'interface utilisateur de vRealize Automation Code Stream, vous pouvez obtenir les liaisons \$ pour n'importe quelle propriété.</p> <ol style="list-style-type: none"> <li>1 Dans la zone de la propriété de sortie de la tâche, cliquez sur <b>EXAMINER LE RÉSULTAT JSON</b>.</li> <li>2 Pour trouver la liaison \$, entrez n'importe quelle propriété.</li> <li>3 Cliquez sur l'icône de recherche, qui affiche la liaison \$ correspondante.</li> </ol>

## Exemple de sortie JSON :



```

15     "simulated": false,
16     "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
17     "resources": {
18       "Cloud_Machine_1[0]": {
19         "id": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
20         "name": "Cloud_Machine_1[0]",
21         "powerState": "ON",
22         "address": "10.108.79.33",
23         "resourceLink": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
24         "componentTypeId": "Cloud.vSphere.Machine",
25         "endpointType": "vsphere",
26         "resourceName": "Cloud_Machine_1-mcm110615-146929827053",
27         "resourceId": "1606fbcd-40e5-4edc-ab85-7b559aa986ad",
28         "resourceDescLink": "/resources/compute-descriptions/1952d1d3-15f0-4574-ae42-4fbf8a87d4cc",

```

Path finder

address

\$(Stage0.Task0.output.deploymentDetails.resources['Cloud\_Machine\_1[0]'].address)

## Exemple d'objet de détails de déploiement :

```

{
  "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"
    }
  }
```

Tableau 3-13. Automatiser les tâches de déploiement continues : Kubernetes

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
Kubernetes			
	Input	<p><b>action</b> : une des opérations <b>GET, CREATE, APPLY, DELETE, ROLLBACK</b></p> <ul style="list-style-type: none"> <li>■ <b>timeout</b> : délai d'expiration global d'une action</li> <li>■ <b>filterByLabel</b> : étiquette supplémentaire pour filtrer l'action <b>GET</b> avec K8S</li> <li>■ <b>labelSelector</b></li> </ul> <p><b>GET, CREATE, DELETE, APPLY</b></p> <ul style="list-style-type: none"> <li>■ <b>yaml</b> : YAML en ligne à traiter et à envoyer à Kubernetes</li> <li>■ <b>parameters</b> : paire KEY, VALUE - Remplacer <b>\$\$KEY</b> par <b>VALUE</b> dans la zone d'entrée YAML en ligne</li> <li>■ <b>filePath</b> : chemin d'accès relatif à partir du point de terminaison Git SCM, s'il est fourni, à partir duquel extraire le YAML</li> <li>■ <b>scmConstants</b> : paire KEY, VALUE - Remplacer <b>\$\$KEY</b> par <b>VALUE</b> dans le YAML extrait sur SCM.</li> <li>■ <b>continueOnConflict</b> : lorsqu'elle est définie sur true, si une ressource est déjà présente, la tâche se poursuit.</li> </ul> <p><b>ROLLBACK</b></p> <ul style="list-style-type: none"> <li>■ <b>resourceType</b> : type de ressource à restaurer</li> <li>■ <b>resourceName</b> : nom de la ressource à restaurer</li> <li>■ <b>namespace</b> : espace de noms dans lequel la restauration doit être effectuée</li> <li>■ <b>revision</b> : révision à restaurer</li> </ul>	<p><code>\${MY_STAGE.MY_TASK.input.action}</code> #Détermine l'action à effectuer.</p> <p><code>\${MY_STAGE.MY_TASK.input.timeout}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filterByLabel}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.yaml}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.parameters}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filePath}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.scmConstants}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.input.continueOnConflict}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceType}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceName}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.namespace}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.revision}</code></p>
	Output	<p><b>response</b> : capture la réponse complète</p> <p><b>response.&lt;RESOURCE&gt;</b> : ressource correspond à configMaps, deployments, endpoints, ingresses, jobs, namespaces, pods, replicaSets, replicationControllers, secrets, services, statefulSets, nodes, loadBalancers.</p> <p><b>response.&lt;RESOURCE&gt;.&lt;KEY&gt;</b> : la clé correspond à apiVersion, kind, metadata, spec</p>	<p><code>\${MY_STAGE.MY_TASK.output.response}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.response. }</code></p>

Tableau 3-14. Intégrer des applications de développement, de test et de déploiement

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
<b>Bamboo</b>			
	Input	plan : nom du plan planKey : clé du plan variables : variables à transmettre au plan parameters : paramètres à transmettre au plan	<pre> \${MY_STAGE.MY_TASK.input.plan} \${MY_STAGE.MY_TASK.input.planKey} \${MY_STAGE.MY_TASK.input.variables} \${MY_STAGE.MY_TASK.input.parameters} # Faire référence à tous les paramètres \${MY_STAGE.MY_TASK.input.parameters.param1} # Faire référence à la valeur de param1 </pre>
	Output	resultUrl : URL de la build résultante buildResultKey : clé de la build résultante buildNumber : numéro de la build buildTestSummary : résumé de l'exécution des tests successfulTestCount : résultat du test transmis failedTestCount : résultat du test ayant échoué skippedTestCount : résultat du test ignoré artifacts : artifacts de la build	<pre> \${MY_STAGE.MY_TASK.output.resultUrl} \${MY_STAGE.MY_TASK.output.buildResultKey} \${MY_STAGE.MY_TASK.output.buildNumber} \${MY_STAGE.MY_TASK.output.buildTestSummary} # Faire référence à tous les résultats \${MY_STAGE.MY_TASK.output.successfulTestCount} # Faire référence à un nombre de tests spécifique \${MY_STAGE.MY_TASK.output.buildNumber} </pre>
<b>Jenkins</b>			
	Input	job : nom du travail Jenkins parameters : paramètres à transmettre à la tâche	<pre> \${MY_STAGE.MY_TASK.input.job} \${MY_STAGE.MY_TASK.input.parameters} # Faire référence à tous les paramètres \${MY_STAGE.MY_TASK.input.parameters.param1} # Faire référence à la valeur d'un paramètre </pre>
	Output	job : nom du travail Jenkins jobId : ID du travail résultant, tel que 1234 jobStatus : état dans Jenkins jobResults : collecte des résultats de la couverture de test/code jobUrl : URL de l'exécution de la tâche résultante	<pre> \${MY_STAGE.MY_TASK.output.job} \${MY_STAGE.MY_TASK.output.jobId} \${MY_STAGE.MY_TASK.output.jobStatus} \${MY_STAGE.MY_TASK.output.jobResults} # Faire référence à tous les résultats \${MY_STAGE.MY_TASK.output.jobResults.junitResponse} # Faire référence aux résultats JUnit \${MY_STAGE.MY_TASK.output.jobResults.jacocoResponse} # Faire référence aux résultats JaCoCo \${MY_STAGE.MY_TASK.output.jobUrl} </pre>
<b>TFS</b>			

Tableau 3-14. Intégrer des applications de développement, de test et de déploiement (suite)

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
	Input	<p>projectCollection : collection de projets depuis TFS</p> <p>teamProject : projet sélectionné dans la collection disponible</p> <p>buildDefinitionId : ID de définition de build à exécuter</p>	<p><code>\${MY_STAGE.MY_TASK.input.projectCollection}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.teamProject}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.buildDefinitionId}</code></p>
	Output	<p>buildId : ID de build résultant</p> <p>buildUrl : URL pour consulter le résumé de la build</p> <p>logUrl : URL pour consulter les journaux</p> <p>dropLocation : emplacement cible des artefacts, le cas échéant</p>	<p><code>\${MY_STAGE.MY_TASK.output.buildId}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.buildUrl}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.logUrl}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.dropLocation}</code></p>
<b>vRO</b>			
	Input	<p>workflowId : ID du workflow à exécuter</p> <p>parameters : paramètres à transmettre au workflow</p>	<p><code>\${MY_STAGE.MY_TASK.input.workflowId}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.parameters}</code></p>
	Output	<p>workflowExecutionId : ID de l'exécution du workflow</p> <p>properties : propriétés de sortie de l'exécution du workflow</p>	<p><code>\${MY_STAGE.MY_TASK.output.workflowExecutionId}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.properties}</code></p>

Tableau 3-15. Intégrer d'autres applications via une API

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
<b>REST</b>			
	Input	<p>url : URL à appeler</p> <p>action : méthode HTTP à utiliser</p> <p>headers : en-têtes HTTP à transmettre</p> <p>payload : charge utile de la demande</p> <p>fingerprint : empreinte digitale à faire correspondre si l'URL est https</p> <p>allowAllCerts : lorsque la valeur est définie sur true, il peut s'agir de n'importe quel certificat disposant de l'URL https</p>	<p><code>\${MY_STAGE.MY_TASK.input.url}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.action}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.headers}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.payload}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.fingerprint}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code></p>

Tableau 3-15. Intégrer d'autres applications via une API (suite)

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
	Output	<p>responseCode : codes de réponse HTTP</p> <p>responseHeaders : en-têtes de réponse HTTP</p> <p>responseBody : format de chaîne de la réponse reçue</p> <p>responseJson : réponse transversale si content-type est <b>application/json</b></p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseHeaders}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.output.responseHeaders.header1}</code> # Faire référence à l'en-tête de réponse « header1 »</p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson}</code> # Faire référence à la réponse JSON</p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson.a.b.c}</code> # Faire référence à l'objet imbriqué suivant le chemin d'accès a.b.c JSON dans la réponse</p>
<b>Interrogation</b>			
	Input	<p>url : URL à appeler</p> <p>headers : en-têtes HTTP à transmettre</p> <p>exitCriteria : critères à respecter pour que la tâche aboutisse ou échoue. Une paire clé-valeur de « success » → expression, « failure » → Expression :</p> <p>pollCount : nombre d'itérations à effectuer</p> <p>pollIntervalSeconds : nombre de secondes à attendre entre chaque itération</p> <p>ignoreFailure : lorsque la valeur est définie sur true, ignore les échecs de réponse intermédiaire</p> <p>fingerprint : empreinte digitale à faire correspondre si l'URL est https</p> <p>allowAllCerts : lorsque la valeur est définie sur true, il peut s'agir de n'importe quel certificat disposant de l'URL https</p>	<p><code>\${MY_STAGE.MY_TASK.input.url}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.headers}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.exitCriteria}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollCount}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollIntervalSeconds}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.ignoreFailure}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.fingerprint}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code></p>
	Output	<p>responseCode : codes de réponse HTTP</p> <p>responseBody : format de chaîne de la réponse reçue</p> <p>responseJson : réponse transversale si content-type est <b>application/json</b></p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson}</code> # Refer to response as JSON</p>

Tableau 3-16. Exécuter des scripts distants et définis par l'utilisateur

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
<b>PowerShell</b>			
	Input	<p>host : adresse IP ou nom de domaine complet de la machine</p> <p>username : nom d'utilisateur à utiliser pour la connexion</p> <p>password : mot de passe à utiliser pour se connecter</p> <p>useTLS : tentative de connexion https</p> <p>trustCert : lorsque la valeur est définie sur true, approuve les certificats auto-signés</p> <p>script : script à exécuter</p> <p>workingDirectory : chemin d'accès au répertoire vers lequel basculer avant d'exécuter le script</p> <p>environmentVariables : paire clé-valeur de variable d'environnement à définir</p> <p>arguments : arguments à passer au script</p>	<p><code>\${MY_STAGE.MY_TASK.input.host}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.username}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.password}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.useTLS}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.trustCert}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.script}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.input.workingDirectory}</code></p> <p><code>}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.input.environmentVariables}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.input.arguments}</code></p>
	Output	<p>response : contenu du fichier <code>\$SCRIPT_RESPONSE_FILE</code></p> <p>responseFilePath : valeur de <code>\$SCRIPT_RESPONSE_FILE</code></p> <p>exitCode : code de sortie du processus</p> <p>logFilePath : chemin d'accès au fichier contenant stdout</p> <p>errorFilePath : chemin d'accès au fichier contenant stderr</p>	<p><code>\${MY_STAGE.MY_TASK.output.response}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.output.responseFilePath}</code></p> <p><code>}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.exitCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.logFilePath}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.errorFilePath}</code></p>
<b>SSH</b>			

Tableau 3-16. Exécuter des scripts distants et définis par l'utilisateur (suite)

Type de tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
	Input	host : adresse IP ou nom de domaine complet de la machine username : nom d'utilisateur à utiliser pour la connexion password : mot de passe à utiliser pour se connecter (si vous le souhaitez, vous pouvez utiliser privateKey) privateKey : PrivateKey à utiliser pour se connecter passphrase : phrase secrète facultative pour déverrouiller privateKey script : script à exécuter workingDirectory : chemin d'accès au répertoire vers lequel basculer avant d'exécuter le script environmentVariables : paire clé-valeur de la variable d'environnement à définir	<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 : contenu du fichier \$SCRIPT_RESPONSE_FILE responseFilePath : valeur de \$SCRIPT_RESPONSE_FILE exitCode : code de sortie du processus logFilePath : chemin d'accès au fichier contenant stdout errorFilePath : chemin d'accès au fichier contenant 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>

## Utilisation d'une liaison de variable entre des tâches

Cet exemple montre comment utiliser des liaisons de variables dans vos tâches de pipeline.

Tableau 3-17. Exemples de formats de syntaxe

Exemple	Syntaxe
Pour utiliser une valeur de sortie de tâche pour les notifications de pipeline et les propriétés de sortie de pipeline	<code>\${&lt;Stage Key&gt;.&lt;Task Key&gt;.output.&lt;Task output key&gt;}</code>
Pour faire référence à la valeur de sortie de la tâche précédente comme entrée de la tâche actuelle	<code>\${&lt;Previous/Current Stage key&gt;.&lt;Previous task key not in current Task group&gt;.output.&lt;task output key&gt;}</code>

## Pour en savoir plus

Pour plus d'informations sur la liaison de variables, reportez-vous à :

- [Utilisation des liaisons de variables dans les pipelines vRealize Automation Code Stream](#)
- [Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline dans vRealize Automation Code Stream](#)
- [Types de tâche disponibles dans vRealize Automation Code Stream](#)

## Envoi de notifications relatives à un pipeline dans vRealize Automation Code Stream

Les notifications sont un moyen de communiquer avec vos équipes et de leur donner l'état de vos pipelines dans vRealize Automation Code Stream.

Vous pouvez configurer vRealize Automation Code Stream pour envoyer des notifications lors de l'exécution d'un pipeline, en fonction de l'état de l'intégralité du pipeline, de l'étape ou de la tâche.

- Une notification par e-mail envoie un e-mail lors des événements suivants :
  - Achèvement, attente, échec, annulation ou démarrage du pipeline.
  - Exécution, échec ou démarrage d'une étape.
  - Exécution, attente, échec ou démarrage d'une tâche.
- Une notification par ticket crée un ticket et l'attribue à un membre de l'équipe lors des événements suivants :
  - Échec ou exécution du pipeline.
  - Échec de l'étape.
  - Échec de la tâche.
- Une notification Webhook envoie une demande à une autre application lors des événements suivants :
  - Échec, achèvement, attente, annulation ou démarrage du pipeline.
  - Échec, achèvement ou démarrage de l'étape.
  - Échec, fin, attente ou démarrage de la tâche.

Par exemple, vous pouvez configurer une notification par e-mail sur une tâche d'opération d'utilisateur pour obtenir une approbation à un point spécifique de votre pipeline. Lorsque le pipeline s'exécute, cette tâche envoie un e-mail à la personne qui doit approuver la tâche. Si le délai d'expiration de la tâche Opération de l'utilisateur est défini en jours, heures ou minutes, l'utilisateur requis doit approuver le pipeline avant l'expiration de la tâche. Sinon, le pipeline échoue comme prévu.



Vous pouvez également configurer une notification pour créer un ticket Jira en cas d'échec d'une tâche de pipeline. Sinon, vous pouvez configurer une notification Webhook pour envoyer une demande à un canal Slack sur l'état d'un pipeline en fonction de l'événement du pipeline.

Vous pouvez utiliser des variables dans tous les types de notifications. Par exemple, vous pouvez utiliser `${var}` dans l'URL d'une notification Webhook.

### Conditions préalables

- Vérifiez qu'un ou plusieurs pipelines sont créés. Consultez les cas d'utilisation dans la section [Chapitre 5 Didacticiels d'utilisation de vRealize Automation Code Stream](#).
- Pour envoyer des notifications par e-mail, confirmez que vous pouvez accéder à un serveur de messagerie actif. Pour obtenir une assistance, consultez votre administrateur.
- Pour créer des tickets, par exemple un ticket Jira, confirmez que le point de terminaison existe. Reportez-vous à la section [Présentation des points de terminaison dans vRealize Automation Code Stream](#).
- Pour envoyer une notification basée sur une intégration, vous pouvez créer une notification Webhook. Ensuite, confirmez que le Webhook est ajouté et qu'il fonctionne. Vous pouvez utiliser des notifications avec des applications telles que Slack, GitHub ou GitLab.

### Procédure

- 1 Ouvrez un pipeline.
- 2 Pour créer une notification sur l'état du pipeline entier, ou sur l'état d'une étape ou d'une tâche :

Pour créer une notification basée sur...	Ce que vous devez faire...
L'état du pipeline	Cliquez sur une zone vide dans le canevas du pipeline.
L'état d'une étape	Cliquez sur une zone vide dans une étape du pipeline.
L'état d'une tâche	Cliquez sur une tâche dans une étape du pipeline.

- 3 Cliquez sur l'onglet **Notifications**.
- 4 Cliquez sur **Ajouter**, sélectionnez le type de notification et configurez les détails de la notification.
- 5 Pour créer une notification Slack lorsqu'un pipeline réussit, créez une notification webhook.
  - a Sélectionnez **Webhook**.
  - b Entrez les informations pour configurer la notification Slack.

- c Cliquez sur **Enregistrer**.
- d Lorsque le pipeline s'exécute, le canal Slack reçoit la notification de l'état du pipeline. Par exemple, les utilisateurs peuvent voir les éléments suivants sur le canal Slack :

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

- 6 Pour créer un ticket Jira, configurez les informations de ticket.
  - a Sélectionnez **Ticket**.
  - b Entrez les informations pour configurer la notification Jira.
  - c Cliquez sur **Enregistrer**.

### Notification

**Send notification type**

☐ Email
 ☒ Ticket
 ☐ Webhook

**When pipeline \***

☒ Fails
 ☐ Completes

**Jira endpoint \***

Jira-Notification ▼

**Create Ticket**

**Jira project \***

YourProject

**Issue type \***

Bug

**Assignee \***

username@yourcompany.com

**Summary \$ \***

Pipeline failed

**Description \$**

Research and correct

CANCEL SAVE

## Résultats

Félicitations ! Vous avez appris que vous pouvez créer différents types de notifications dans plusieurs zones de votre pipeline dans vRealize Automation Code Stream.

## Étape suivante

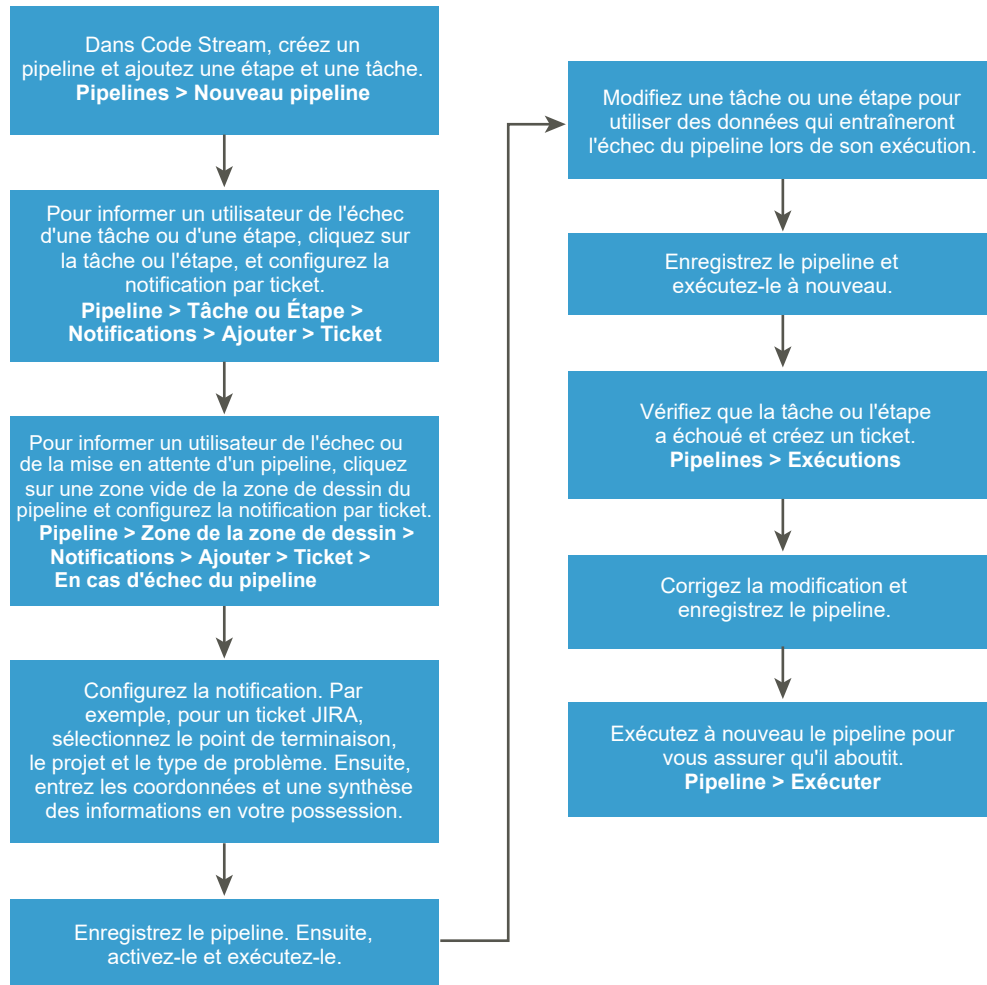
Pour obtenir un exemple détaillé de création d'une notification, reportez-vous à la section [Création d'un ticket JIRA dans vRealize Automation Code Stream en cas d'échec d'une tâche de pipeline](#).

## Création d'un ticket JIRA dans vRealize Automation Code Stream en cas d'échec d'une tâche de pipeline

Si une étape ou une tâche de votre pipeline échoue, vous pouvez créer un ticket JIRA avec vRealize Automation Code Stream. Vous pouvez attribuer le ticket à la personne qui doit résoudre le problème. Vous pouvez également créer un ticket lorsque le pipeline est en attente ou lorsqu'il aboutit.

Vous pouvez ajouter et configurer des notifications pour une tâche, une étape ou un pipeline. vRealize Automation Code Stream crée le ticket en fonction de l'état de la tâche, de l'étape ou du pipeline au niveau duquel vous ajoutez la notification. Par exemple, si un point de terminaison n'est pas disponible, vous pouvez demander à vRealize Automation Code Stream de créer un ticket JIRA pour la tâche qui échoue parce qu'il est impossible d'établir la connexion avec le point de terminaison.

Vous pouvez également créer des notifications lorsque votre pipeline aboutit. Par exemple, vous pouvez informer votre équipe d'assurance qualité de la réussite des pipelines afin qu'elle puisse confirmer la build et exécuter un pipeline de test différent. Vous pouvez également en informer votre équipe de performance afin qu'elle puisse mesurer les performances du pipeline et préparer sa mise à jour à des fins de préparation ou de production.



Dans cet exemple, un ticket JIRA est créé en cas d'échec d'une tâche de pipeline.

#### Conditions préalables

- Vérifiez que vous disposez d'un compte JIRA valide et que vous pouvez vous connecter à votre instance de JIRA.
- Vérifiez qu'un point de terminaison JIRA existe et qu'il fonctionne.

#### Procédure


- 1 Dans votre pipeline, cliquez sur une tâche.
- 2 Dans la zone de configuration des tâches, cliquez sur **Notifications**.
- 3 Cliquez sur **Ajouter** et configurez les informations relatives au ticket.
  - a Cliquez sur **Ticket**.
  - b Sélectionnez le point de terminaison JIRA.
  - c Entrez le projet JIRA et le type de problème.

- d Entrez l'adresse e-mail du destinataire du ticket.
- e Entrez un résumé et une description du ticket, puis cliquez sur **Enregistrer**.

### 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 Enregistrez le pipeline, puis activez-le et exécutez-le.
- 5 Testez le ticket.
  - a Modifiez les informations concernant la tâche pour y inclure les données entraînant l'échec de cette dernière.
  - b Enregistrez le pipeline et exécutez-le à nouveau.
  - c Cliquez sur **Exécutions** et confirmez que le pipeline a échoué.
  - d Lors de l'exécution, confirmez que vRealize Automation Code Stream a créé le ticket et l'a envoyé.
  - e Modifiez de nouveau les informations concernant la tâche pour les corriger, puis exécutez de nouveau le pipeline et assurez-vous qu'il aboutit.

## Résultats

Félicitations ! Vous avez demandé à vRealize Automation Code Stream de créer un ticket JIRA lorsque la tâche de pipeline a échoué et l'avez attribué à la personne qui devait le résoudre.

## Étape suivante

Continuez à ajouter des notifications pour informer votre équipe de vos pipelines.

# Restauration d'un déploiement dans vRealize Automation Code Stream

Vous configurez la restauration sous la forme d'un pipeline avec des tâches qui rétablissent votre déploiement à un état stable antérieur suite à un échec dans un pipeline de déploiement. Vous associez le pipeline de restauration aux tâches ou aux étapes que vous souhaitez restaurer en cas d'échec.

Les raisons d'effectuer une restauration varient en fonction de votre rôle.

- En tant qu'ingénieur en publication, je souhaite que vRealize Automation Code Stream s'assure de la réussite d'un pipeline lors de la publication, afin que je puisse décider, en connaissance de cause, de la poursuite de la publication ou de son annulation. Les échecs possibles incluent l'échec d'une tâche, un rejet dans UserOps et le dépassement du seuil des mesures.
- En tant que propriétaire d'environnements, je souhaite redéployer une version précédente afin de pouvoir rapidement rétablir l'état de fonctionnement optimal d'un environnement.
- En tant que propriétaire d'environnements, je souhaite être en mesure d'annuler un déploiement Bleu-vert afin de pouvoir réduire les interruptions de service causées par les versions ayant échoué.

Lorsque vous utilisez un modèle de pipeline intelligent pour créer un pipeline CD tandis que l'option de restauration est activée, la restauration est automatiquement ajoutée aux tâches du pipeline. Dans ce cas d'utilisation, vous utiliserez le modèle de pipeline intelligent pour définir la restauration d'un déploiement d'application vers un cluster Kubernetes à l'aide du modèle de déploiement de mise à niveau propagée. Le modèle de pipeline intelligent crée un pipeline de déploiement, et un ou plusieurs pipelines de restauration.

- Dans le pipeline de déploiement, la restauration est requise en cas d'échec des tâches Mettre à jour le déploiement ou Vérifier le déploiement.
- Dans le pipeline de restauration, le déploiement est mis à jour avec une ancienne image.

Vous pouvez également créer manuellement un pipeline de restauration à l'aide d'un modèle vide. Avant de créer un pipeline de restauration, vous devez planifier votre flux de restauration. Pour plus d'informations de base sur la restauration, reportez-vous à la section [Planification de la restauration dans vRealize Automation Code Stream](#).

## Conditions préalables

- Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).
- Configurez les clusters Kubernetes sur lesquels votre pipeline déploiera votre application. Configurez un cluster de développement et un cluster de production.
- Créez le développement Kubernetes et les points de terminaison de production qui déploient votre image d'application sur les clusters Kubernetes.
- Vérifiez que vous disposez d'une configuration de registre Docker.
- Vérifiez que vous disposez d'un fichier YAML Kubernetes à appliquer au déploiement.
- Familiarisez-vous avec le modèle de pipeline intelligent CD. Reportez-vous à la section [Planification d'une build native CD dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent](#).

## Procédure

- 1 Cliquez sur **Pipelines > Nouveau pipeline > Modèle intelligent > Livraison continue**.
- 2 Entrez les informations dans le modèle de pipeline intelligent.
  - a Sélectionnez un projet.
  - b Entrez un nom de pipeline comme **MiseÀniveauPropagée-Exemple**.
  - c Sélectionnez les environnements de votre application. Pour ajouter une restauration à votre déploiement, vous devez sélectionner **Prod**.
  - d Cliquez sur **Sélectionner**, choisissez un fichier YAML Kubernetes et cliquez sur **Processus**.  
Le modèle de pipeline intelligent affiche les services et les environnements de déploiement disponibles.
  - e Sélectionnez le service que le pipeline utilisera pour le déploiement.
  - f Sélectionnez les points de terminaison du cluster pour les environnements Dév et Prod.
  - g Pour la source Image, sélectionnez **Entrée d'exécution du pipeline**.
  - h Pour le modèle de déploiement, sélectionnez **Mise à niveau propagée**.

- i Cliquez sur **Restaurer**.
- j Fournissez l'**URL de contrôle de santé**.

Modèle intelligent: Livraison continue

Conditions préalable: ☐ Kubernetes ☐ Registre Docker

Projet: test1

Nom du pipeline: RollbackUpgrade-Exemple

Environnement: ☒ Développement ☐ Production

Fichiers YAML Kub.: [SÉLECTIONNER](#) [TRAITER](#)

Fichiers traités: Kubernetesbgreen1.yaml

Sélectionner un service

Nom du déploiement	Service	Espace de noms	Image
codestream-demo	codestream-demo	bgreen1	symphony-tango-beta2.jfrog.io/codestream-demo

1 services

Déploiement

Environnement	Point de terminaison de cluster	Espace de noms
Développement	K8s-AWS	bgreen1-585283
Production	K8s-AWS	bgreen1

Source d'image: ☐ Déclencheur Docker ☒ Entrée d'exécution du pipeline

Modèle de déploie: ☐ Canary ☒ Mise à niveau propagée ☐ Bleu-vert

Restaurer: ☒

URL de contrôle de: /health-check.json

[CRÉER](#) [ANNULER](#)

- 3 Pour créer le pipeline nommé MiseÀniveauRestauration-Exemple, cliquez sur **Créer**.

Le pipeline MiseÀniveauRestauration-Exemple s'affiche, accompagné de l'icône de restauration sur les tâches des étapes de développement et de production qui peuvent être restaurées.

RollbackUpgrade-Exemple Désactivé

Espace de travail | Entrée | **Modèle** | Sortie

**Development**

Create Namesp...  
Kubernetes

Create Secret  
Kubernetes

Create Sever  
Kubernetes

+ Tâche parallèle

**Production**

Create Se\_trans...  
Kubernetes

Update Deploy...  
Kubernetes

Verify Deploy...  
Kubernetes

+ Tâche parallèle

+ Étape

**Tâche: Create Secret** Notifications Restaurer [VALIDER LA TÂCHE](#)

Nom de la tâche: Create Secret

Type: Kubernetes

Continuer en cas d'... ☐

exécuter la tâche: ☒ Toujours ☐ Sur condition

Propriétés de la tâche Kubernetes

Cluster Kubernetes: Dev-VKE-Cluster

Délai d'attente (en ...): 5

Action: ☐ Obtenir ☒ Créer ☐ Appliquer ☐ Supprimer ☐ Restaurer

Continuer en cas d'... ☐

Type de source: ☐ Contrôle de la source ☒ Définition locale

Définition locale de: [LIRE À PARTIR DU FICHIER](#)

```

1 apiVersion: v1
2 data:
3   .dockercfg: eyJ2Iiw6G9ueS10VW5nb3Y1IiZXRhbi1Ssd1afjka12sdefxrg2hsch2hsh
4   2fdsH5zxdg2dFh5ss13h8dFs5453hFdsfhF3as15ghn1fs315h3f1ds5h5s3df15
5   h315sdf15h53108Fds45h04Fds4h5n4In19
6 Kind: Secret
7 metadata:
8   name: jfrog-beta2
9   namespace: bgreen-549930
10 type: Kubernetes.io/dockercfg

```

Paramètres de sortie: [status](#) [k8sRollbackTaskFields](#) [endpoint](#) [response](#) [yamis](#) [operation](#) [config](#)

[ENREGISTRER](#) [EXÉCUTER](#) [FERMER](#) Dernier enregistrement le : il y a 3 minutes

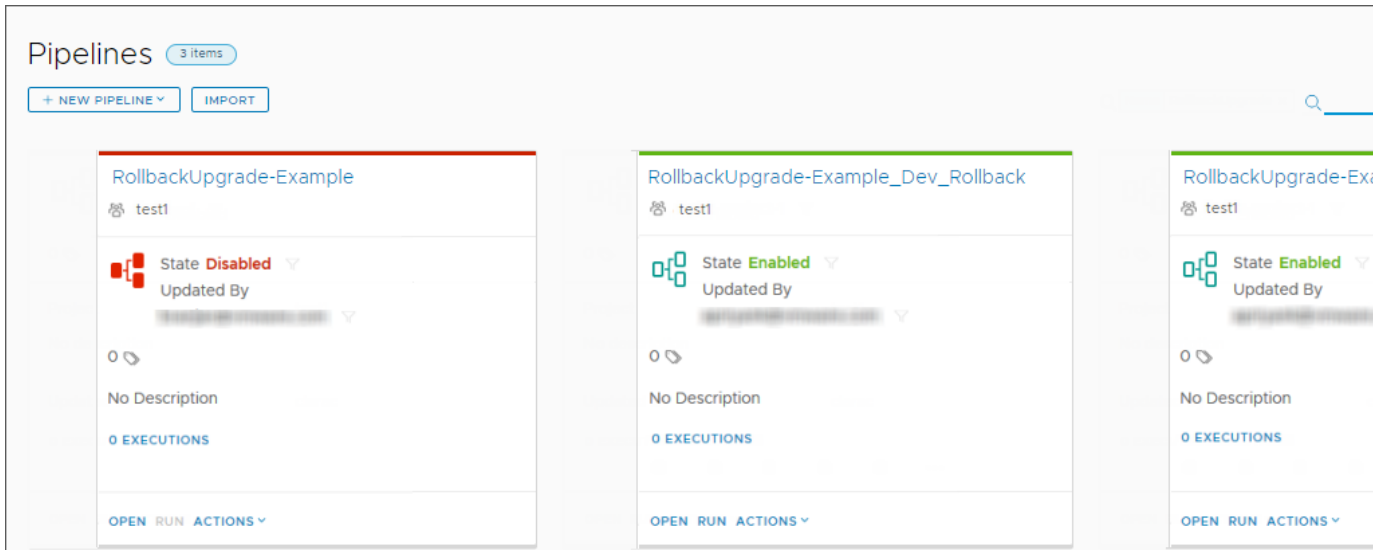
- 4 Fermez le pipeline.

Sur la page Pipelines, vous pouvez observer la présence du pipeline que vous avez créé et d'un nouveau pipeline pour chaque étape de votre pipeline.

- MiseÀniveauPropagée-Exemple. vRealize Automation Code Stream désactive le pipeline que vous avez créé par défaut, ce qui garantit que vous l'examinez avant de l'exécuter.



- **MiseÀniveauPropagée-Exemple\_Dév\_Restauration.** Ce pipeline de développement par restauration est appelé en cas d'échec de tâches lors de l'étape de développement, telles que Créer un service, Créer un secret, Créer un déploiement et Vérifier le déploiement. Le pipeline de développement par restauration est activé par défaut pour garantir la restauration des tâches de développement.
- **MiseÀniveauPropagée-Exemple\_Prod\_Restauration.** Ce pipeline de production par restauration est appelé en cas d'échec de tâches lors de l'étape de production, telles que Déployer la phase 1, Vérifier la phase 1, Déployer la phase de lancement, Terminer la phase de lancement, Vérifier la phase de lancement. Le pipeline de production par restauration est activé par défaut pour garantir la restauration des tâches de production.



- 5 Activez et exécutez le pipeline que vous avez créé.

Lorsque vous démarrez l'exécution, vous êtes invité à saisir des paramètres d'entrée. Vous indiquez l'image et la balise correspondant au point de terminaison du référentiel Docker que vous utilisez.

- 6 Sur la page Exécutions, sélectionnez **Actions > Afficher l'exécution** pour surveiller l'exécution du pipeline.

L'opération **RUNNING** du pipeline commence et les différentes tâches de l'étape de développement sont effectuées une à une. Si le pipeline ne parvient pas à exécuter une tâche pendant l'étape de développement, le pipeline nommé MiseÀniveauPropagée-Exemple\_Dév\_Restauration est déclenché pour restaurer le déploiement et l'état du pipeline est redéfini sur **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

Après restauration, la page Exécutions répertorie deux exécutions de pipeline MiseÀniveauPropagée-Exemple.

- Le pipeline créé qui a été restauré indique **ROLLBACK\_COMPLETED**.
- Le pipeline de développement par restauration qui a été déclenché pour effectuer la restauration indique **COMPLETED**.

## Executions 604 items

+ NEW EXECUTION

RollbackUpgrade-Example\_Dev...#1

COMPLETED

Stages:

1 Rollback for RollbackUpgrade-Example#1

By on 01/11/2019 1:36 PM  
Execution Completed.  
**Comments:**Triggered to rollback Development.Create Deployment of RollbackUpgrade-E

RollbackUpgrade-Example#1

ROLLBACK\_COMPLETED

Stages:

0

By on 01/11/2019 1:24 PM  
**Create Deployment ROLLBACK\_COMPLETED**

### Résultats

Félicitations ! Vous avez défini un pipeline avec restauration et vRealize Automation Code Stream a restauré le pipeline au niveau du point d'échec.

# Planification de la génération, de l'intégration et de la livraison de votre code en mode natif dans vRealize Automation Code Stream

Avant que vRealize Automation Code Stream ne puisse générer, intégrer et livrer votre code en utilisant la capacité native qui crée un pipeline CD, CI et CICD pour vous, planifiez votre build en natif. Ensuite, vous pouvez créer votre pipeline en vous servant de l'un des modèles de pipeline intelligents ou en ajoutant manuellement des étapes et des tâches.

Nous avons inclus plusieurs exemples qui vous montrent comment planifier l'intégration et la livraison de build en continu. Ces plans décrivent les conditions préalables à remplir et fournissent des présentations pour vous accompagner dans la préparation à l'utilisation de la capacité de build native afin de générer efficacement vos pipelines.

Ce chapitre contient les rubriques suivantes :

- Planification d'une build native CICD dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent
- Planification d'une build native CI dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent
- Planification d'une build native CD dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent
- Planification d'une build native CICD dans vRealize Automation Code Stream avant d'ajouter des tâches manuellement
- Planification de la restauration dans vRealize Automation Code Stream

## Planification d'une build native CICD dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent

Pour créer un pipeline d'intégration continue et de livraison continue (CICD) dans vRealize Automation Code Stream, vous pouvez utiliser le modèle de pipeline intelligent CICD. Pour planifier votre build native CICD, vous allez collecter les informations dont vous avez besoin pour remplir le modèle de pipeline intelligent avant de l'utiliser pour créer le pipeline dans cet exemple de plan.

Une fois que vous avez saisi les informations requises dans le modèle de pipeline intelligent et que vous les avez enregistrées, le modèle crée un pipeline comprenant des étapes et des tâches. Il indique également où déployer votre image en fonction des types d'environnement que vous sélectionnez, comme Dev (Développement) et Prod (Production). Le pipeline publie ensuite votre image Docker et effectue les actions requises pour l'exécuter. Après l'exécution de votre pipeline, vous pouvez surveiller les tendances en matière d'exécution de pipeline.

Pour créer un pipeline CI/CD, vous devez planifier les étapes d'intégration continue (Continuous Integration, CI) et de livraison continue (Continuous Delivery, CD) de votre pipeline.

Lorsqu'un pipeline inclut une image depuis Docker Hub, vous devez vous assurer que l'image intègre cURL avant d'exécuter le pipeline. Lorsque le pipeline s'exécute, vRealize Automation Code Stream télécharge un fichier binaire qui utilise cURL pour exécuter des commandes.

## Planification de l'étape d'intégration continue (CI)

Pour planifier l'étape de CI de votre pipeline, vous allez configurer les conditions externes et internes requises, et déterminer les informations à saisir dans la partie relative à la CI du modèle de pipeline intelligent. En voici un résumé.

Points de terminaison et référentiels dont vous aurez besoin :

- D'un référentiel de code source Git dans lequel vos développeurs archivent du code. vRealize Automation Code Stream intègre la toute dernière version du code dans le pipeline lorsque les développeurs valident les modifications.
- D'un point de terminaison Git pour le référentiel dans lequel se trouve le code source du développeur.
- D'un point de terminaison Docker pour l'hôte de la build Docker qui exécutera les commandes de la build dans un conteneur.
- D'un point de terminaison Kubernetes afin que vRealize Automation Code Stream puisse déployer votre image sur un cluster Kubernetes.
- D'une image de générateur qui crée le conteneur sur lequel les tests d'intégration continue s'exécutent.
- D'un point de terminaison de registre d'images afin que l'hôte de la build Docker puisse en extraire l'image du générateur.

Vous aurez besoin d'accéder à un projet. Projet qui regroupe tous vos travaux, y compris votre pipeline, vos points de terminaison et vos tableaux de bord. Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).

Vous aurez besoin d'un Webhook Git, ce qui permet à vRealize Automation Code Stream d'utiliser le déclencheur Git pour déclencher votre pipeline lorsque les développeurs valident les modifications apportées au code. Reportez-vous à la section [Utilisation du déclencheur Git dans vRealize Automation Code Stream pour exécuter un pipeline](#).

Vos boîtes à outils dédiées aux builds :

- Votre type de build, par exemple Maven.
- Tous les outils de génération post-processus que vous utilisez, tels que JUnit, JaCoCo, Checkstyle et FindBugs.

Votre outil de publication :

- Un outil comme docker qui déploiera votre conteneur de builds.
- Une balise d'image, soit l'ID de validation, soit le numéro de build.

Votre espace de travail dédié aux builds :

- Un hôte de build Docker, point de terminaison Docker.
- Un registre d'images. La partie relative à la CI du pipeline extrait l'image du point de terminaison de registre sélectionné. Le conteneur exécute les tâches de CI et déploie votre image. Si le registre requiert des informations d'identification, vous devez d'abord créer un point de terminaison de registre d'images, puis le sélectionner ici pour que l'hôte puisse extraire l'image du registre.
- URL de l'image de générateur qui crée le conteneur sur lequel les tâches d'intégration continue s'exécutent.

## Planification de l'étape de livraison continue (CD)

Pour planifier l'étape de CD de votre pipeline, vous allez configurer les conditions externes et internes requises, et déterminer les informations à saisir dans la partie relative à la CD du modèle de pipeline intelligent.

Points de terminaison dont vous aurez besoin :

- D'un point de terminaison Kubernetes afin que vRealize Automation Code Stream puisse déployer votre image sur un cluster Kubernetes.

Types d'environnement et fichiers :

- Tous les types d'environnement dans lesquels vRealize Automation Code Stream déploiera votre application, comme Dev (Développement) et Prod (Production). Le modèle de pipeline intelligent crée les étapes et les tâches de votre pipeline en fonction des types d'environnement que vous sélectionnez.

Tableau 4-1. Étapes de votre pipeline créées par le modèle de pipeline intelligent CICD

Contenu du pipeline	Fonctionnement
Étape de publication de la build	Génère et teste votre code, crée l'image de générateur et publie l'image sur votre hôte Docker.
Étape de développement	Utilise un cluster de développement Amazon Web Services (AWS) pour créer et déployer votre image. À cette étape, vous pouvez créer un espace de noms sur le cluster et créer une clé secrète.
Étape de production	Utilise une version de production de VMware Tanzu Kubernetes Grid Integrated Edition (anciennement VMware Enterprise PKS) pour déployer votre image sur un cluster Kubernetes de production.

- Fichier YAML Kubernetes que vous sélectionnez dans la partie relative à la CD du modèle de pipeline intelligent CICD.

Pour appliquer le fichier, cliquez sur **Sélectionner**, sélectionnez le fichier YAML Kubernetes, puis cliquez sur **Traiter**. Le modèle de pipeline intelligent affiche les services et les environnements de déploiement disponibles. Vous sélectionnez un service, le point de terminaison du cluster et la stratégie de déploiement. Par exemple, pour utiliser le modèle de déploiement Canary, sélectionnez **Canary** et entrez le pourcentage de la phase de déploiement.

Modèle intelligent: CI/CD

Étape 2 sur 2

Environnement ☒ Développement ☒ Production

Fichiers YAML Kubernetes... **SÉLECTIONNER** **TRAITER**

Fichiers traités: codestream.yaml

Sélectionner un service

Nom du déploiement	Service	Espace de noms	Image
codestream-demo	codestream-demo	bgreen1	3

1 services

Déploiement

Environnement	Point de terminaison de cluster	Espace de noms
Développement	1030Endpoint-Kubernetes 開発環境の Kubernetes	bgreen1-408728
Production	1030Endpoint-Kubernetes 開発環境の Kubernetes	bgreen1

Modèle de déploiement ☒ Canary ☐ Mise à niveau propagée ☐ Bleu-vert

Phase 1 20 %

Restaurer ☐

URL de contrôle de santé /health-check.jsor

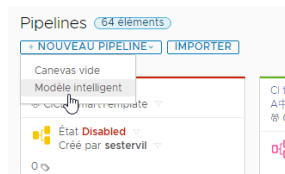
**CRÉER** **RETOUR** **ANNULER**

Pour découvrir un exemple d'utilisation du modèle de pipeline intelligent pour créer un pipeline en vue d'un déploiement Bleu-vert, reportez-vous à la section [Déploiement d'une application dans vRealize Automation Code Stream vers un déploiement Bleu-vert](#).

## Création du pipeline CICD à l'aide du modèle de pipeline intelligent

Une fois que vous aurez collecté toutes les informations et configuré ce dont vous avez besoin, voici comment vous créerez un pipeline à partir du modèle de pipeline intelligent CICD.

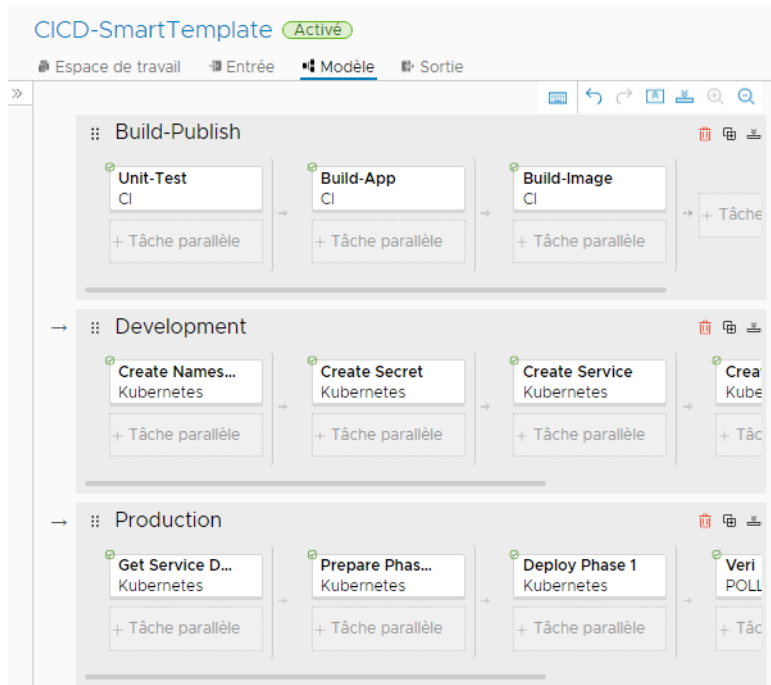
Sur la page Pipelines, sélectionnez **Nouveau pipeline > Modèles intelligents**.



Sélectionnez le modèle de pipeline intelligent CICD.



Vous allez remplir le modèle et enregistrer le pipeline avec les étapes qu'il crée. Si vous devez apporter des modifications finales, vous pouvez modifier le pipeline et l'enregistrer.



Ensuite, vous allez activer le pipeline et l'exécuter. Une fois le pipeline exécuté, voici quelques éléments que vous pouvez rechercher :

- Vérifiez que votre pipeline a abouti. Cliquez sur **Exécutions** et recherchez votre pipeline. S'il a échoué, corrigez les erreurs et exécutez-le à nouveau.
- Vérifiez que le Webhook Git fonctionne correctement. L'onglet **Activité** Git affiche les événements. Cliquez sur **Déclencheurs > Git > Activité**.
- Examinez le tableau de bord de pipeline et observez les tendances. Cliquez sur **Tableaux de bord** et recherchez votre tableau de bord de pipeline. Vous pouvez également créer un tableau de bord personnalisé pour générer des rapports sur des KPI supplémentaires.



Pour obtenir un exemple détaillé, reportez-vous à la section [Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans vRealize Automation Code Stream](#).

## Planification d'une build native CI dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent

Pour créer un pipeline d'intégration continue (CI) dans VMware Code Stream, vous pouvez utiliser le modèle de pipeline intelligent CI. Pour planifier votre build native CI, vous allez collecter les informations dont vous avez besoin pour remplir le modèle de pipeline intelligent avant de l'utiliser pour créer le pipeline dans cet exemple de plan.

Lorsque vous remplissez le modèle de pipeline intelligent, un pipeline CI est créé dans votre référentiel et les actions requises pour l'exécuter sont lancées. Après l'exécution de votre pipeline, vous pouvez surveiller les tendances en matière d'exécution de pipeline.

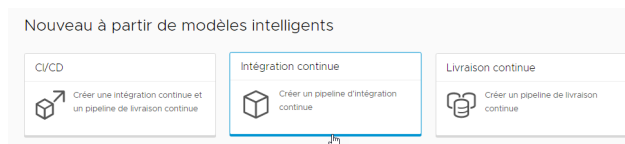
Pour planifier votre build avant d'utiliser le modèle de pipeline intelligent CI, vous allez collecter les informations pour votre build, puis suivre la partie CI de [Planification d'une build native CI/CD dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent](#).

Une fois que vous aurez collecté toutes les informations et configuré ce dont vous avez besoin, voici comment vous créerez un pipeline à partir du modèle de pipeline intelligent CI.

Dans Pipelines, sélectionnez **Modèles intelligents**.



Sélectionnez le modèle de pipeline intelligent CI.



Vous aller remplir le modèle et cliquer sur **Créer** pour enregistrer le pipeline avec les étapes qu'il crée.

Vous pouvez modifier le pipeline pour apporter les modifications finales qui pourraient être nécessaires. Ensuite, vous pouvez activer le pipeline et l'exécuter. Une fois le pipeline exécuté, voici quelques éléments à rechercher :

- Assurez-vous que l'exécution de votre pipeline a réussi. Cliquez sur **Exécutions** et recherchez votre pipeline. S'il a échoué, corrigez les erreurs et exécutez-le à nouveau.

- Assurez-vous que le webhook Git fonctionne correctement. L'onglet **Activité** Git affiche les événements. Cliquez sur **Déclencheurs > Git > Activité**.
- Consultez le tableau de bord du pipeline et examinez les tendances. Cliquez sur **Tableaux de bord**, puis recherchez votre tableau de bord de pipeline. Vous pouvez également créer un tableau de bord personnalisé pour générer un rapport sur des indicateurs de performance clés supplémentaires.

Pour obtenir un exemple détaillé, reportez-vous à la section [Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans vRealize Automation Code Stream](#).

## Planification d'une build native CD dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent

Pour créer un pipeline de prestation continue (CD) dans vRealize Automation Code Stream, vous pouvez utiliser le modèle de pipeline intelligent CD. Pour planifier votre build native CD, vous allez collecter les informations dont vous avez besoin pour remplir le modèle de pipeline intelligent avant de l'utiliser pour créer le pipeline dans cet exemple de plan.

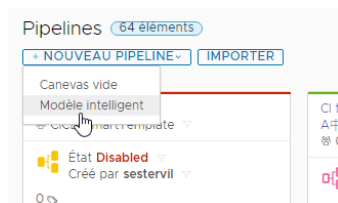
Lorsque vous remplissez le modèle de pipeline intelligent, un pipeline CD est créé dans votre référentiel et les actions requises pour l'exécuter sont lancées. Après l'exécution de votre pipeline, vous pouvez surveiller les tendances en matière d'exécution de pipeline.

Pour planifier votre build avant d'utiliser le modèle de pipeline intelligent CD, vous devez :

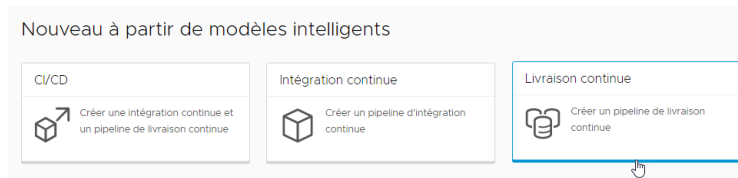
- Collecter les informations concernant votre Build, puis suivre la partie relative à la CD de la section [Planification d'une build native CI/CD dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent](#).
- Ajouter un point de terminaison Kubernetes où vRealize Automation Code Stream déploiera le conteneur.
- Identifier un projet qui regroupera tous vos travaux, y compris votre pipeline, vos points de terminaison et vos tableaux de bord.

Une fois que vous aurez collecté toutes les informations et configuré ce dont vous avez besoin, voici comment vous créez un pipeline à partir du modèle de pipeline intelligent CD.

Dans Pipelines, sélectionnez **Modèles intelligents**.



Sélectionnez le modèle de pipeline intelligent CD.



Remplissez le modèle, indiquez un nom pour le pipeline, puis cliquez sur **Créer** pour enregistrer le pipeline avec les étapes qu'il crée.

Vous pouvez modifier le pipeline pour apporter les modifications finales qui pourraient être nécessaires. Ensuite, vous pouvez activer le pipeline et l'exécuter. Une fois le pipeline exécuté, voici quelques éléments à rechercher :

- Vérifiez que votre pipeline a abouti. Cliquez sur **Exécutions** et recherchez votre pipeline. S'il a échoué, corrigez les erreurs et exécutez-le à nouveau.
- Vérifiez que le Webhook Git fonctionne correctement. L'onglet **Activité** Git affiche les événements. Cliquez sur **Déclencheurs > Git > Activité**.
- Examinez le tableau de bord de pipeline et observez les tendances. Cliquez sur **Tableaux de bord** et recherchez votre tableau de bord de pipeline. Vous pouvez également créer un tableau de bord personnalisé pour générer des rapports sur des KPI supplémentaires.

Pour obtenir un exemple détaillé, reportez-vous à la section [Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans vRealize Automation Code Stream](#).

## Planification d'une build native CI/CD dans vRealize Automation Code Stream avant d'ajouter des tâches manuellement

Pour créer un pipeline d'intégration continue et de prestation continue (CI/CD) dans vRealize Automation Code Stream, vous pouvez manuellement ajouter des étapes et des tâches. Pour planifier votre build native CI/CD, vous allez collecter les informations dont vous avez besoin, puis créer un pipeline et y ajouter manuellement des étapes et des tâches.

Vous devrez planifier les étapes d'intégration continue (Continuous Integration, CI) et de prestation continue (Continuous Delivery, CD) de votre pipeline. Après la création et l'exécution de votre pipeline, vous pourrez surveiller les tendances en matière d'exécution de pipeline.

Pour planifier les étapes CI et CD de votre pipeline, vous vérifierez que toutes les conditions requises sont remplies avant de créer votre pipeline.

## Planification des exigences externes et internes

Pour créer un pipeline à partir de cet exemple de plan, vous allez utiliser un hôte Docker, un référentiel Git, le type de build Maven et plusieurs outils de génération de post-processus.

Points de terminaison et référentiels dont vous aurez besoin :

- D'un référentiel de code source Git dans lequel vos développeurs archivent du code. vRealize Automation Code Stream intègre la toute dernière version du code dans le pipeline lorsque les développeurs valident les modifications.
- D'un point de terminaison Docker pour l'hôte de la build Docker qui exécutera les commandes de la build dans un conteneur.
- D'un point de terminaison Kubernetes afin que vRealize Automation Code Stream puisse déployer votre image sur un cluster Kubernetes.
- D'une image de générateur qui crée le conteneur sur lequel les tests d'intégration continue s'exécutent.
- D'un point de terminaison de registre d'images afin que l'hôte de la build Docker puisse en extraire l'image du générateur.

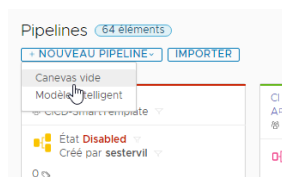
Vous aurez besoin d'accéder à un projet. Projet qui regroupe tous vos travaux, y compris votre pipeline, vos points de terminaison et vos tableaux de bord. Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).

Vous aurez besoin d'un Webhook Git, ce qui permet à vRealize Automation Code Stream d'utiliser le déclencheur Git pour déclencher votre pipeline lorsque les développeurs valident les modifications apportées au code. Reportez-vous à la section [Utilisation du déclencheur Git dans vRealize Automation Code Stream pour exécuter un pipeline](#).

## Création du pipeline CI/CD et configuration de l'espace de travail

Vous devrez créer le pipeline, puis configurer l'espace de travail, les paramètres d'entrée de pipeline et les tâches.

Pour créer le pipeline, cliquez sur **Pipelines > Nouveau pipeline > Canevas vide**.



Dans l'onglet Espace de travail, entrez les informations sur l'intégration continue :

- Incluez votre hôte de build Docker.
- Entrez l'URL de votre image de générateur.
- Sélectionnez le point de terminaison de registre d'images afin que le pipeline puisse en extraire l'image. Le conteneur exécute les tâches de CI et déploie votre image. Si le registre requiert des informations d'identification, vous devez d'abord créer le point de terminaison de registre d'images, puis le sélectionner ici pour que l'hôte puisse extraire l'image du registre.

- Ajoutez les artefacts qui doivent être mis en cache. Pour qu'une build aboutisse, des artefacts tels que les répertoires sont téléchargés sous forme de dépendances. Le cache est l'emplacement dans lequel se trouvent ces artefacts. Par exemple, les artefacts dépendants peuvent inclure le répertoire `.m2` pour Maven et le répertoire `node_modules` pour le Node.js. Ces répertoires sont mis en cache sur toutes les exécutions de pipeline pour gagner du temps lors des générations.



Dans l'onglet **Entrée**, configurez les paramètres d'entrée de pipeline.

- Si votre pipeline utilise des paramètres d'entrée à partir d'un événement déclencheur Git, Gerrit ou Docker, sélectionnez le type de déclencheur pour les paramètres d'insertion automatique. Ces événements sont les suivants : Modifier l'objet de Gerrit ou Git ou Nom du propriétaire de l'événement pour Docker. Si votre pipeline n'utilise aucun paramètre d'entrée transmis par l'événement, laissez les paramètres d'insertion automatique définis sur **Aucun**.
- Pour appliquer une valeur et une description à un paramètre d'entrée de pipeline, cliquez sur les trois points verticaux, puis sur **Modifier**. La valeur que vous saisissez est utilisée comme entrée pour les tâches, les étapes ou les notifications.
- Pour ajouter un paramètre d'entrée de pipeline, cliquez sur **Ajouter**. Par exemple, vous pouvez ajouter le paramètre `approvers` pour afficher une valeur par défaut pour chaque exécution, mais que vous pouvez remplacer par un autre approbateur lors de l'exécution.
- Pour ajouter ou supprimer un paramètre inséré, cliquez sur **Ajouter/supprimer le paramètre inséré**. Par exemple, vous pouvez supprimer un paramètre inutilisé pour réduire l'encombrement sur la page de résultats et afficher uniquement les paramètres d'entrée utilisés.

Espace de travail **Entrée** Modèle Sortie

Paramètres d'entrée

Paramètres d'injection ☐ Gerrit ☒ Git ☐ Docker ☐ Aucun

[AJOUTER](#) [AJOUTER/SUPPRIMER DES PARAMÈTRES INJECTÉS](#)

Marqué d'une étoile	Nom	Valeur	Description
<input checked="" type="checkbox"/>	GIT_BRANCH_NAME		
<input checked="" type="checkbox"/>	GIT_CHANGE_SUBJECT		
<input checked="" type="checkbox"/>	GIT_COMMIT_ID		
<input checked="" type="checkbox"/>	GIT_EVENT_DESCRIPTION		
<input checked="" type="checkbox"/>	GIT_EVENT_OWNER_NAME		
<input checked="" type="checkbox"/>	GIT_EVENT_TIMESTAMP		
<input checked="" type="checkbox"/>	GIT_REPO_NAME		
<input checked="" type="checkbox"/>	GIT_SERVER_URL		

Configurez le pipeline pour tester votre code :

- Ajoutez et configurez une tâche de CI.
- Incluez les étapes pour exécuter `mvn test` sur votre code.
- Exécutez les outils de génération de post-processus, tels que JUnit et JaCoCo, FindBugs et Checkstyle, pour identifier les problèmes après l'exécution de la tâche.

**Tâche Unit-Test** Notifications Restaurer **VALIDER LA TÂCHE**

Nom de la tâche **Unit-Test**

Type **CI**

Continuer e... ☐

exécuter la t... ☒ Toujours ☐ Sur condition

Intégration continue

Étapes

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

Conserver le... ☒

Exporter

Traiter [AJOUTER](#)

JUnit	JUnit	/demo-project	<input checked="" type="checkbox"/>
JaCoCo	Jacoco	/demo-project	<input checked="" type="checkbox"/>
Checkstyle	Checkstyle	/demo-project	<input checked="" type="checkbox"/>
FindBugs	Findbugs	/demo-project	<input checked="" type="checkbox"/>

Paramètres de sortie [status](#) [exports](#)

Configurez le pipeline pour générer votre code :

- Ajoutez et configurez une tâche de CI.
- Incluez les étapes pour exécuter `mvn clean install` sur votre code.
- Incluez l'emplacement et le nom du fichier JAR pour qu'il conserve votre artefact.

Tâche **Build-App** Notifications Restaurer **VALIDER LA TÂCHE**

Nom de la tâche \*

Type \*

Continuer en cas... ☐

exécuter la tâche ☒ Toujours ☐ Sur condition

Intégration continue

Étapes \*

```
1 cd demo_project
2 mvn clean install -DskipTests
```

Conservé le... ☒

Exporter

Traiter

JUnit   ☒

JaCoCo   ☒

Checkstyle   ☒

FindBugs   ☒

Paramètres de sortie

Configurez le pipeline pour publier votre image sur votre hôte Docker :

- Ajoutez et configurez une tâche de CI.
- Ajoutez des étapes qui valideront, exporteront, généreront et transféreront votre image.
- Ajoutez la clé d'exportation de `IMAGE` que la tâche suivante utilisera.

Tâche **Build-Image** Notifications Restaurer **VALIDER LA TÂCHE**

Nom de la tâche \*

Type \*

Continuer en cas... ☐

exécuter la tâche ☒ Toujours ☐ Sur condition

Intégration continue

Étapes \*

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

Conservé les art... ☒

Exporter

Traiter

Paramètres de sortie

Après avoir configuré l'espace de travail, les paramètres d'entrée, les tâches de test et les tâches de build, enregistrez votre pipeline.

## Activation et exécution de votre pipeline

Après avoir configuré votre pipeline avec des étapes et des tâches, vous pouvez enregistrer et activer le pipeline.

Ensuite, patientez le temps que le pipeline s'exécute et se termine, puis vérifiez qu'il a abouti. S'il a échoué, corrigez les erreurs et exécutez-le à nouveau.

Une fois que le pipeline a abouti, voici quelques éléments que vous souhaitez peut-être vérifier :

- Examinez l'exécution du pipeline et affichez les résultats des étapes des tâches.

- Dans l'espace de travail de l'exécution du pipeline, localisez les détails concernant votre conteneur et le référentiel Git cloné.
- Dans l'espace de travail, examinez les résultats de vos outils post-processus, et recherchez les erreurs, la couverture du code, les bogues et les problèmes de style.
- Vérifiez que votre artefact est préservé. Vérifiez également que l'image a été exportée avec le nom et la valeur IMAGE.
- Accédez à votre référentiel Docker et vérifiez que le pipeline a publié votre conteneur.

Pour obtenir un exemple détaillé illustrant l'intégration continue de votre code par vRealize Automation Code Stream, reportez-vous à la section [Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans vRealize Automation Code Stream](#).

## Planification de la restauration dans vRealize Automation Code Stream

En cas d'échec de l'exécution d'un pipeline, vous pouvez utiliser la restauration pour rétablir votre environnement à un état précédemment stable. Pour utiliser la restauration, vous devez planifier un flux de restauration et comprendre comment l'implémenter.

Un flux de restauration prescrit les étapes requises pour remédier à l'échec d'un déploiement. Le flux se présente sous la forme d'un pipeline de restauration incluant une ou plusieurs tâches séquentielles qui varient selon le type du déploiement exécuté et ayant échoué. Par exemple, le déploiement et la restauration d'une application traditionnelle sont différents du déploiement et de la restauration d'une application de conteneur.

Pour revenir à un état de déploiement approprié, un pipeline de restauration inclut généralement des tâches :

- de nettoyage des états ou des environnements ;
- d'exécution d'un script spécifié par l'utilisateur pour restaurer les modifications ;
- de déploiement d'une révision précédente d'un déploiement.

Pour ajouter une restauration à un pipeline de déploiement existant, vous associez le pipeline de restauration aux tâches ou aux étapes du pipeline de déploiement que vous souhaitez restaurer avant d'exécuter votre pipeline de déploiement.

### Comment configurer une restauration

Pour configurer une restauration dans votre déploiement, vous devez :

- Créer un pipeline de déploiement.
- Identifier les points d'échec potentiels dans le pipeline de déploiement, qui déclencheront la restauration, afin que vous puissiez associer votre pipeline de restauration. Par exemple, vous pouvez associer votre pipeline de restauration à un type de tâche de condition ou



d'interrogation dans le pipeline de déploiement, qui vérifie si une tâche précédente s'est terminée avec succès. Pour plus d'informations sur les tâches de condition, reportez-vous à la section [Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline dans vRealize Automation Code Stream](#).

- Déterminez l'étendue de l'échec qui déclenchera le pipeline de restauration, comme l'échec d'une tâche ou d'une étape. Vous pouvez également associer une restauration à une étape.
- Choisissez les tâches de restauration à exécuter en cas d'échec. Vous créez votre pipeline de restauration avec ces tâches.

Vous pouvez créer manuellement un pipeline de restauration ou vRealize Automation Code Stream peut en créer un pour vous.

- À l'aide d'un canevas vide, vous pouvez créer manuellement un pipeline de restauration qui suit un flux parallèlement à un pipeline de déploiement existant. Ensuite, vous associez le pipeline de restauration à une ou plusieurs tâches du pipeline de déploiement qui déclenche la restauration en cas d'échec.
- À l'aide d'un modèle de pipeline intelligent, vous pouvez configurer un pipeline de déploiement avec l'action de restauration. Ensuite, vRealize Automation Code Stream crée automatiquement un ou plusieurs pipelines de restauration par défaut avec des tâches prédéfinies, qui restaurent le déploiement en cas d'échec.

Pour obtenir un exemple détaillé de la configuration, à l'aide d'un modèle de pipeline intelligent, d'un pipeline CD avec une restauration, reportez-vous à la section [Restauration d'un déploiement dans vRealize Automation Code Stream](#).

## Que se passe-t-il si mon pipeline de déploiement comporte plusieurs tâches ou étapes avec restauration ?

Si vous avez plusieurs tâches ou des tâches et des étapes avec ajout de restauration, sachez que la séquence de restauration varie.

**Tableau 4-2. Critères déterminant la séquence de restauration**

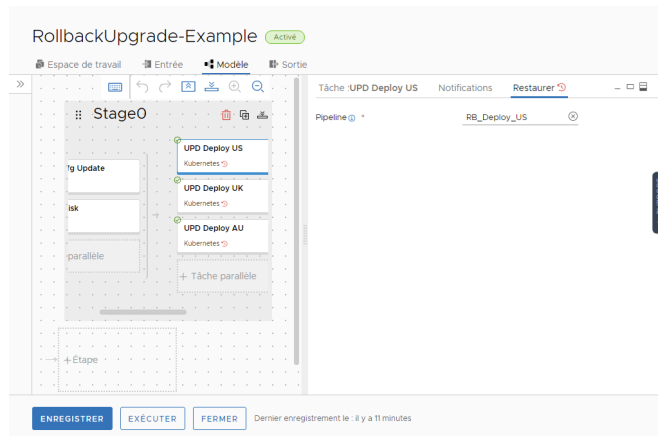
Si vous ajoutez une restauration à...	Au moment de la restauration...
Des tâches parallèles	Si l'une des tâches échoue, sa restauration aura lieu lorsque toutes les autres tâches exécutées en parallèle seront terminées ou auront échoué. La restauration ne se produit pas immédiatement après l'échec de la tâche.
La tâche incluse dans une étape et l'étape	Si une tâche échoue, la restauration de la tâche s'exécute. Si la tâche se trouve dans un groupe de tâches parallèles, la restauration de la tâche s'exécute une fois toutes les tâches parallèles terminées ou ayant échoué. Une fois la restauration de la tâche terminée ou si elle ne parvient pas à s'achever, la restauration de l'étape s'exécute.

Envisagez un pipeline comprenant :

- Une étape de production avec restauration.

- Un groupe de tâches parallèles, chaque tâche comportant sa propre restauration.

La tâche nommée **UPD Deploy US** dispose du pipeline de restauration **RB\_Deploy\_US**. Si **UPD Deploy US** échoue, la restauration suit le flux défini dans le pipeline **RB\_Deploy\_US**.



Si **UPD Deploy US** échoue, le pipeline **RB\_Deploy\_US** s'exécute après que **UPD Deploy UK** et **UPD Deploy AU** se sont également terminés ou ont également échoué. La restauration ne se produit pas immédiatement après l'échec de **UPD Deploy US**. Étant donné que l'étape de production a également été restaurée, une fois le pipeline **RB\_Deploy\_US** exécuté, le pipeline de restauration de l'étape s'exécute.

# Didacticiels d'utilisation de vRealize Automation Code Stream

## 5

Utilisez vRealize Automation Code Stream pour modéliser et prendre en charge le cycle de vie de votre version DevOps et tester et libérer vos applications de manière continue.

Vous avez déjà configuré tout ce dont vous aviez besoin et pouvez maintenant utiliser vRealize Automation Code Stream. Reportez-vous à la section [Chapitre 2 Configuration de vRealize Automation Code Stream pour modéliser le processus de publication](#).

Vous pouvez désormais créer des pipelines qui automatisent la génération et le test du code de développeur avant de le rendre disponible en production. Vous pouvez configurer vRealize Automation Code Stream pour déployer des applications basée sur conteneur ou traditionnelles.

**Tableau 5-1. Utilisation de vRealize Automation Code Stream dans votre cycle de vie DevOps**

Fonctions d'utilisation	Exemples de ce que vous pouvez faire
Utilisez la fonctionnalité de build en natif dans vRealize Automation Code Stream.	Créer des pipelines CD, CI et CICD qui intègrent en permanence, conteneurisent et livrent votre code. <ul style="list-style-type: none"><li>■ Utiliser un modèle de pipeline intelligent pour créer un pipeline pour vous.</li><li>■ Ajouter manuellement des étapes et des tâches à un pipeline.</li></ul>
Publier vos applications et les automatiser.	Intégrer et publier vos applications de différentes manières. <ul style="list-style-type: none"><li>■ Intégrer en continu votre code GitHub ou GitLab dans votre pipeline.</li><li>■ Automatiser le déploiement de votre application à l'aide d'un modèle de cloud YAML</li><li>■ Automatiser le déploiement de votre application sur un cluster Kubernetes.</li><li>■ Publier votre application sur un déploiement Bleu-vert.</li><li>■ Intégrer vRealize Automation Code Stream à vos propres outils de génération, de test et de déploiement.</li><li>■ Utiliser une REST API qui intègre vRealize Automation Code Stream à d'autres applications.</li></ul>
Suivre les tendances, les mesures et les KPI.	Créer des tableaux de bord personnalisés et obtenir un aperçu des performances de vos pipelines.
Résoudre des problèmes	Lorsque l'exécution du pipeline échoue, configurer vRealize Automation Code Stream pour créer un ticket JIRA.

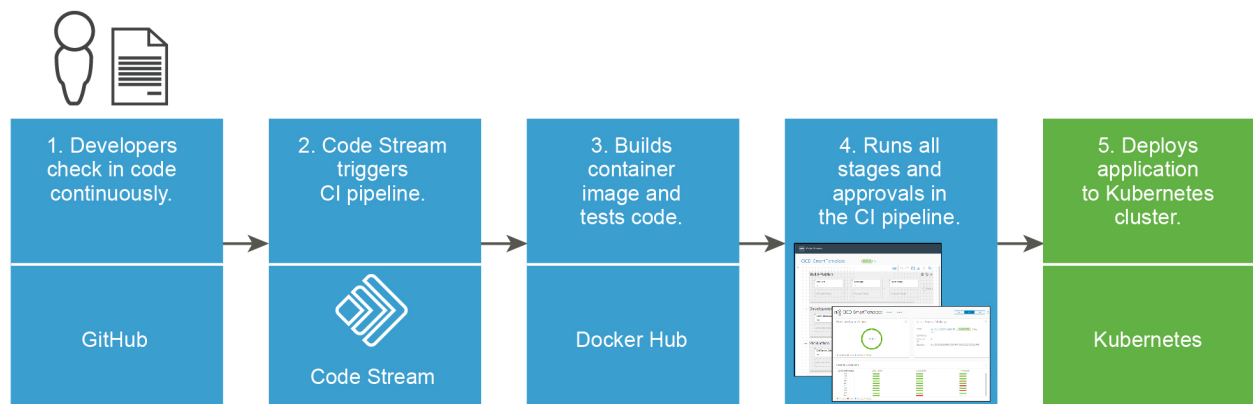
Ce chapitre contient les rubriques suivantes :

- [Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans vRealize Automation Code Stream](#)

- Automatisation de la publication d'une application déployée à partir d'un modèle de cloud YAML dans vRealize Automation Code Stream
- Automatisation de la publication d'une application dans vRealize Automation Code Stream sur un cluster Kubernetes
- Déploiement d'une application dans vRealize Automation Code Stream vers un déploiement Bleu-vert
- Intégration d'outils de génération, de test et de déploiement propres avec vRealize Automation Code Stream
- Utilisation d'une API REST pour intégrer vRealize Automation Code Stream à d'autres applications

## Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans vRealize Automation Code Stream

En tant que développeur, vous souhaitez intégrer continuellement votre code à partir d'un référentiel GitHub ou GitLab Enterprise. Chaque fois que vos développeurs mettent à jour leur code et valident les modifications apportées au référentiel, vRealize Automation Code Stream peut détecter ces modifications et déclencher le pipeline.



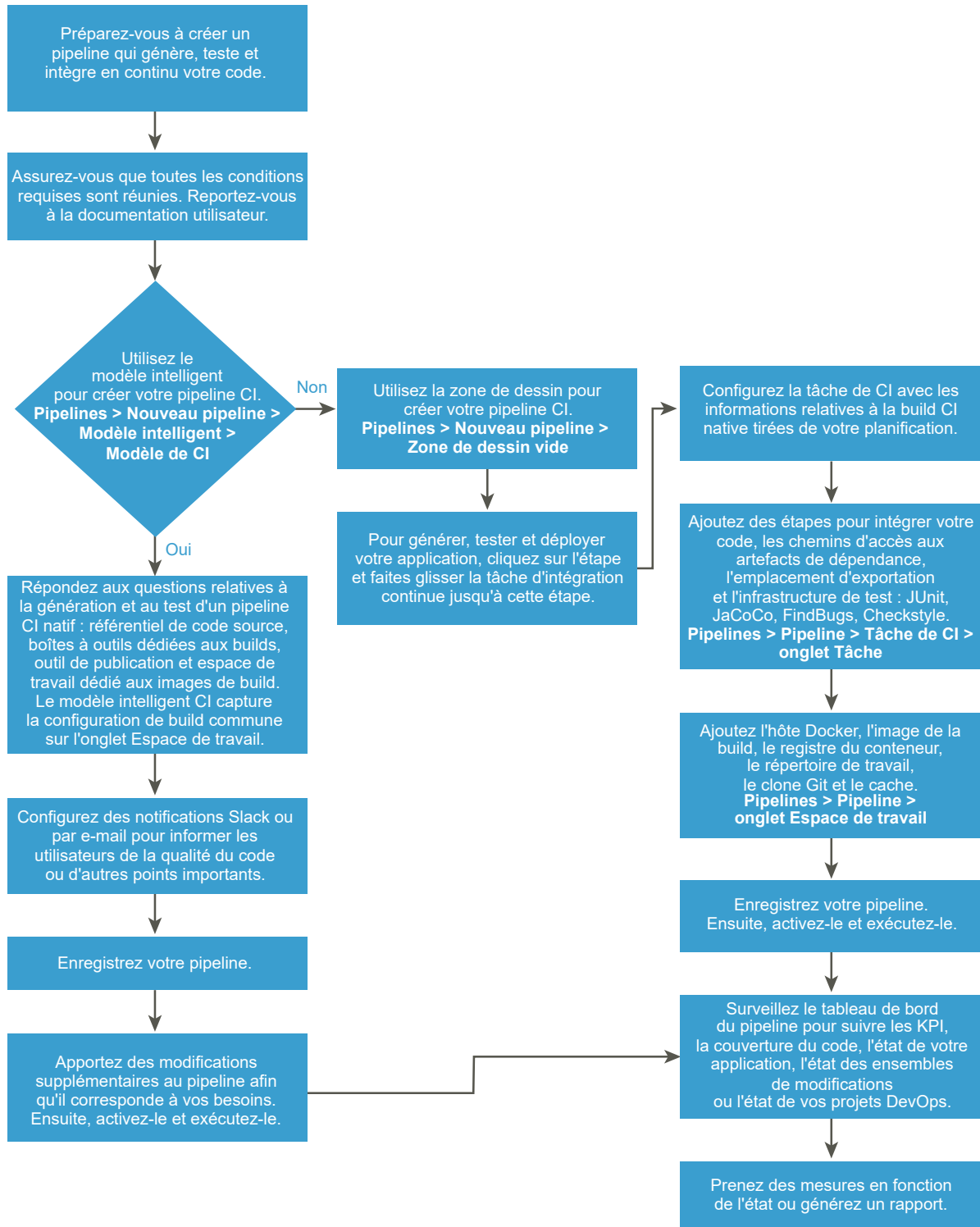
Pour que vRealize Automation Code Stream déclenche votre pipeline dès que le code est modifié, vous devez utiliser le déclencheur Git. vRealize Automation Code Stream déclenche votre pipeline à chaque fois que vous validez les modifications apportées à votre code.

Pour créer votre code, vous utilisez un hôte Docker. Vous utilisez JUnit et JaCoCo comme outils de structure de test, qui exécutent des tests unitaires et la couverture du code, et vous les incluez dans votre pipeline.

Ensuite, vous utilisez le modèle de pipeline intelligent d'intégration continue (CI) pour créer un pipeline CI qui crée, teste et déploie votre code sur le cluster Kubernetes de votre équipe de projet sur AWS. Vous utilisez un cache pour stocker les artefacts de dépendance de code pour votre tâche CI afin de gagner du temps lors de la génération de code.

Dans la tâche de pipeline qui génère et teste votre code, vous devez inclure plusieurs étapes d'intégration continue. Ces étapes se trouvent dans le même répertoire de travail que celui dans lequel le code source est cloné lorsque le pipeline se déclenche.

Pour déployer votre code sur le cluster Kubernetes, vous utilisez une tâche Kubernetes dans votre pipeline. Vous activez et exécutez votre pipeline. Ensuite, apportez une modification à votre code dans le référentiel et observez le déclencheur de pipeline. Pour surveiller et générer des rapports sur vos tendances de pipeline après l'exécution de ce dernier, utilisez les tableaux de bord.



Dans cet exemple, vous utiliserez le modèle de pipeline intelligent d'intégration continue (CI) pour créer un pipeline de CI afin de pouvoir intégrer continuellement votre code à votre pipeline.

Vous pouvez éventuellement créer manuellement le pipeline, et y ajouter des étapes et des tâches. Pour plus d'informations sur la planification d'une build d'intégration continue et la création manuelle du pipeline, reportez-vous à la section [Planification d'une build native CI/CD dans vRealize Automation Code Stream avant d'ajouter des tâches manuellement](#).

### Conditions préalables

- Planifiez votre build d'intégration continue. Reportez-vous à la section [Planification d'une build native CI dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent](#) et à la section sur la planification de l'étape d'intégration continue (CI).
- Vérifiez qu'un référentiel de code source GitLab existe. Pour obtenir de l'aide, consultez votre administrateur vRealize Automation Code Stream.
- Ajoutez un point de terminaison Git. Par exemple, reportez-vous à la section [Utilisation du déclencheur Git dans vRealize Automation Code Stream pour exécuter un pipeline](#).
- Pour que vRealize Automation Code Stream détecte les modifications subies par votre référentiel GitHub ou GitLab, et déclenche un pipeline lorsque des modifications sont apportées, ajoutez un Webhook. Par exemple, reportez-vous à la section [Utilisation du déclencheur Git dans vRealize Automation Code Stream pour exécuter un pipeline](#).
- Ajoutez un point de terminaison hôte Docker, qui crée un conteneur pour la tâche CI, que plusieurs tâches CI peuvent utiliser. Pour plus d'informations sur les points de terminaison, reportez-vous à la section [Présentation des points de terminaison dans vRealize Automation Code Stream](#).
- Obtenez l'URL d'image, l'hôte de build et l'URL d'image de build. Pour obtenir de l'aide, consultez votre administrateur vRealize Automation Code Stream.
- Vérifiez que vous utilisez JUnit et JaCoCo pour vos outils de structure de test.
- Configurez une instance externe pour votre build CI : Jenkins, TFS ou Bamboo. Le plug-in Kubernetes déploiera votre code. Pour obtenir de l'aide, consultez votre administrateur vRealize Automation Code Stream.

### Procédure

- 1 Respectez les conditions préalables.
- 2 Pour créer le pipeline à l'aide du modèle intelligent, ouvrez le modèle de pipeline intelligent CI et remplissez le formulaire.
  - a Cliquez sur **Pipelines > Nouveau pipeline > Modèle intelligent > Intégration continue**.
  - b Répondez aux questions du modèle concernant votre référentiel de code source, les ensembles d'outils de build, l'outil de publication et l'espace de travail de l'image de build.
  - c Ajoutez des notifications Slack ou E-mail pour votre équipe.
  - d Pour que le modèle de pipeline intelligent crée le pipeline, cliquez sur **Créer**.

- e Pour apporter d'autres modifications au pipeline, cliquez sur **Modifier**, apportez vos modifications, puis cliquez sur **Enregistrer**.
  - f Activez le pipeline et exécutez-le.
- 3 Pour créer le pipeline manuellement, ajoutez des étapes et des tâches au canevas et faites en sorte que les informations de votre build CI native soient prêtes pour configurer la tâche d'intégration continue (CI).
- a Cliquez sur **Pipelines > Nouveau pipeline > Canevas vide**.
  - b Cliquez sur l'étape, puis faites glisser les différentes tâches CI du volet de navigation vers l'étape.
  - c Pour configurer la tâche CI, cliquez dessus, puis cliquez sur l'onglet **Tâche**.
  - d Ajoutez les étapes qui intègrent continuellement votre code.
  - e Incluez les chemins d'accès aux artefacts de dépendance.
  - f Ajoutez l'emplacement d'exportation.
  - g Ajoutez les outils de structure de test que vous utiliserez.
  - h Ajoutez l'hôte Docker et l'image de build.
  - i Ajoutez le registre du conteneur, le répertoire de travail et le cache.
  - j Enregistrez le pipeline, puis activez-le.
- 4 Apportez une modification à votre code dans votre référentiel GitHub ou GitLab.  
Le déclencheur Git active votre pipeline, qui commence à s'exécuter.
- 5 Pour vérifier que la modification du code a déclenché le pipeline, cliquez sur **Déclencheurs > Git > Activité**.



- 6 Pour afficher l'exécution de votre pipeline, cliquez sur **Exécutions** et vérifiez que les étapes ont créé et exporté votre image de build.

The screenshot displays the vRealize Automation Code Stream interface. On the left, a sidebar contains navigation options: Dashboards, Executions, User Operations, Pipelines, Manage (with sub-items Endpoints, Variables, and Triggers), and Git. The main area shows the execution details for 'CICD-SmartTemplate #51', which is marked as 'COMPLETED'. A progress bar at the top indicates the status of various steps: Unit-Test, Build-App, Build-Image (all completed), and Create Namespace, Create Secret, Create Service, Create Dep (all in progress). The 'Build-Image' task is selected, showing its details: Task name (Build-Image), Type (CI), Status (COMPLETED), Duration (5s), and Execute Task (Always). The Result section shows the command output, including Docker login and build commands. The Exports section shows the 'IMAGE' value as 'automation/cicd-smart-template:51'.

- 7 Pour surveiller le tableau de bord du pipeline afin de suivre les KPI et les tendances, cliquez sur **Tableaux de bord > Tableaux de bord de pipeline**.

### Résultats

Félicitations ! Vous avez créé un pipeline qui intègre continuellement votre code d'un référentiel GitHub ou GitLab à votre pipeline, et déploie l'image de votre build.

### Étape suivante

Pour plus d'informations, reportez-vous à la section [Plus de ressources pour les administrateurs et les développeurs vRealize Automation Code Stream](#).

## Automatisation de la publication d'une application déployée à partir d'un modèle de cloud YAML dans vRealize Automation Code Stream

En tant que développeur, vous avez besoin d'un pipeline qui extrait un modèle de cloud d'automatisation à partir d'une instance de GitHub sur site à chaque fois que vous validez une modification. Vous avez besoin de ce pipeline pour déployer une application WordPress vers Amazon Web Services (AWS) EC2 ou un centre de données. vRealize Automation Code Stream appelle le modèle de cloud à partir du pipeline et en automatise l'intégration continue et la prestation continue (CICD) pour déployer votre application.

Pour créer et déclencher votre pipeline, vous avez besoin d'un modèle de cloud VMware.

Pour la **Source du modèle de cloud** de votre tâche de modèle de cloud vRealize Automation Code Stream, vous pouvez sélectionner l'une des deux options suivantes :

- **Modèle Cloud Assembly** en tant que contrôle de la source. Dans ce cas, vous n'avez pas besoin d'un référentiel GitLab ou GitHub.
- **Contrôle de la source** si vous utilisez GitLab ou GitHub pour le contrôle de la source. Dans ce cas, vous devez disposer d'un Webhook Git et déclencher le pipeline via le Webhook.

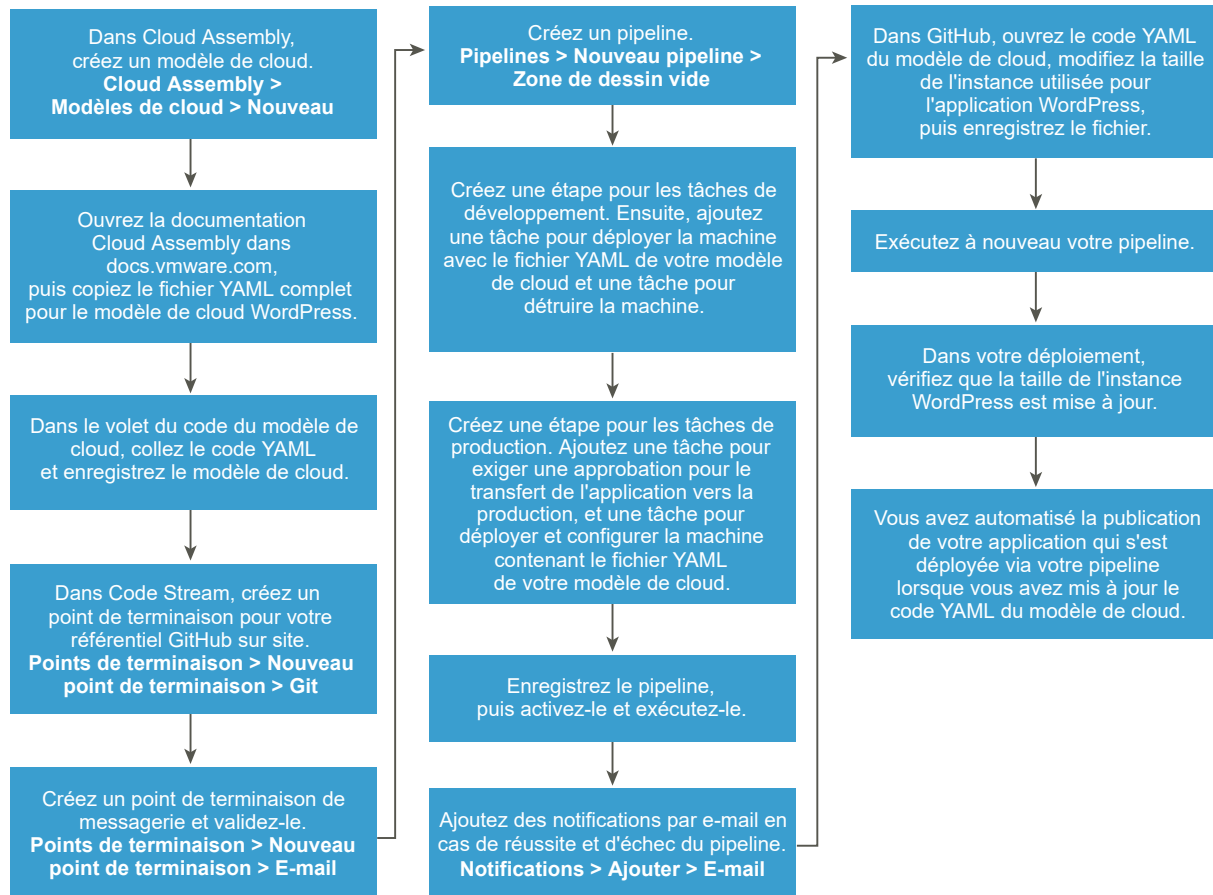
Si vous disposez d'un modèle de cloud YAML dans votre référentiel GitHub et que vous souhaitez utiliser ce modèle de cloud dans votre pipeline, voici ce que vous devez faire.

- 1 Dans vRealize Automation Cloud Assembly, transférez le modèle de cloud vers votre référentiel GitHub.
- 2 Dans vRealize Automation Code Stream, créez un point de terminaison Git. Créez ensuite un Webhook Git qui utilise votre point de terminaison Git et votre pipeline.
- 3 Pour déclencher votre pipeline, mettez à jour tous les fichiers de votre référentiel GitHub et validez votre modification.

Si vous ne disposez pas d'un modèle de cloud YAML dans votre référentiel GitHub et que vous souhaitez utiliser un modèle de cloud à partir du contrôle de la source, suivez cette procédure pour en savoir plus. Elle contient des informations à propos de la création d'un modèle de cloud pour une application WordPress et son déclenchement à partir d'un référentiel GitHub sur site. Chaque fois que vous apportez une modification au modèle de cloud YAML, le pipeline déclenche et automatise la version de votre application.

- Dans vRealize Automation Cloud Assembly, vous devez ajouter un compte de cloud, ajoutez une zone de cloud et créez le modèle de cloud.
- Dans vRealize Automation Code Stream, vous devez ajouter un point de terminaison pour le référentiel GitHub sur site qui héberge le modèle de cloud. Ensuite, vous devez ajouter le modèle de cloud à votre pipeline.

Cet exemple de cas d'utilisation vous montre comment utiliser un modèle de cloud à partir d'un référentiel GitHub sur site.

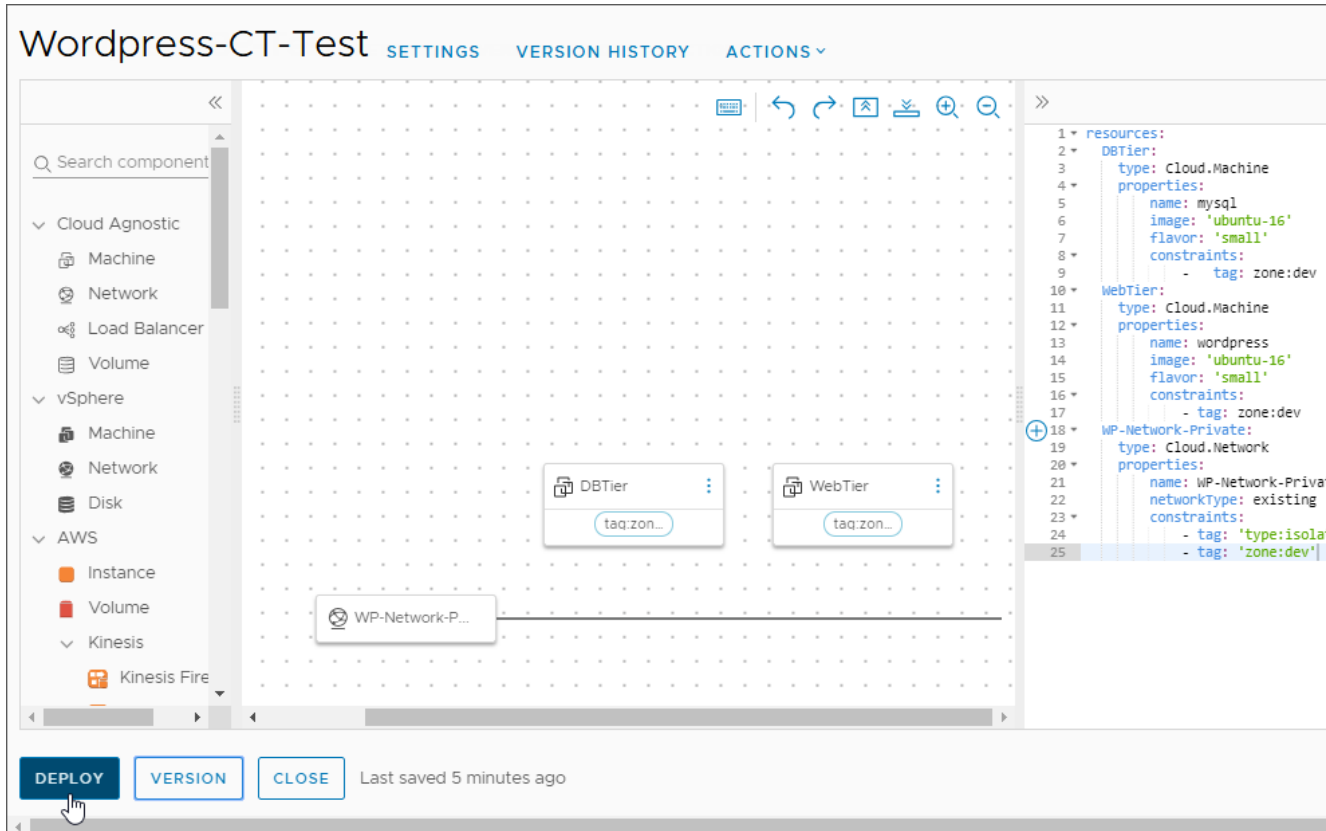


### Conditions préalables

- Ajoutez un compte de cloud et une zone de cloud dans votre infrastructure vRealize Automation Cloud Assembly. Consultez la documentation de vRealize Automation Cloud Assembly.
- Pour créer votre modèle de cloud dans la procédure suivante, copiez le code YAML WordPress dans votre Presse-papiers. Consultez le code YAML du modèle de cloud dans le cas d'utilisation WordPress de la documentation de vRealize Automation Cloud Assembly.
- Ajoutez le code YAML de l'application WordPress à votre instance GitHub.
- Ajoutez un Webhook pour le déclencheur Git afin que votre pipeline puisse extraire votre code YAML chaque fois que validez des modifications. Dans vRealize Automation Code Stream, cliquez sur **Déclencheurs > Git > Webhooks pour Git**.
- Pour utiliser une tâche de modèle de cloud, vous devez disposer de l'un des rôles vRealize Automation Cloud Assembly.

## Procédure

- 1 Dans vRealize Automation Cloud Assembly, procédez comme suit.
  - a Cliquez sur **VMware Cloud Templates**, puis créez un modèle de cloud et un déploiement pour l'application WordPress.
  - b Collez le code YAML WordPress que vous avez copié dans votre Presse-papiers dans votre modèle de cloud et déployez-le.



- 2 Dans vRealize Automation Code Stream, créez des points de terminaison.
  - a Créez un point de terminaison Git pour votre référentiel GitHub sur site, dans lequel se trouve votre fichier YAML.
  - b Ajoutez un point de terminaison E-mail pour informer les utilisateurs de l'état du pipeline lors de son exécution.

Nouveau point de terminaison

Projet \* Codestream

Type \* Email

Nom \* Valeur

Description

Marquer comme rest... ☐ non restreint

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

[CRÉER UNE VARIABLE](#)

[CRÉER](#) [VALIDER](#) [ANNULER](#)

3 Créez un pipeline et ajoutez des notifications de réussite et d'échec de pipeline.

## Notification

**Send notification type**

☒ Email ☐ Ticket ☐ Webhook

**When pipeline**

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

**Email server** ⓘ \*

--Select Email server-- ▾

**Send Email**

**To** ⓘ \$ \*

Email IDs of recipients

**Subject** \$ \*

Email Subject

**Body** ⓘ \$ \*

1	
---	--

CANCEL

SAVE

#### 4 Ajoutez une étape pour le déploiement et ajoutez une tâche de modèle de cloud.

- a Ajoutez une tâche de modèle de cloud qui déploie la machine et configurez-la pour utiliser le fichier YAML du modèle de cloud pour l'application WordPress.

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

- b Ajoutez une tâche de modèle de cloud qui détruit la machine pour libérer des ressources.

- 5 Ajoutez une étape pour la production et incluez des tâches d'approbation et de déploiement.
  - a Ajoutez une tâche d'opération d'utilisateur pour exiger une approbation de transfert de l'application WordPress vers la production.
  - b Ajoutez une tâche de modèle de cloud pour déployer la machine et configurez-la avec le fichier YAML du modèle de cloud pour l'application WordPress.

Lorsque vous sélectionnez **Créer**, le nom du déploiement doit être unique. Si vous ne renseignez pas ce champ, vRealize Automation Code Stream lui attribuera un nom aléatoire unique.

Voici ce que vous devez savoir si vous sélectionnez **Restaurer** dans votre cas d'utilisation : si vous sélectionnez l'action **Restaurer** et que vous entrez une **Version de restauration**, la version doit être au format **n-x**. Par exemple, **n-1**, **n-2**, **n-3**, etc. Si vous créez et mettez à jour le déploiement dans un emplacement autre que vRealize Automation Code Stream, la restauration n'est pas autorisée.

Lorsque vous vous connectez à vRealize Automation Code Stream, un jeton utilisateur est obtenu, ce dernier est valide pendant 30 minutes. Pour les pipelines à durée d'exécution longue, lorsque la tâche antérieure à la tâche du modèle de cloud dure 30 minutes ou plus, le jeton de l'utilisateur expire. Par conséquent, la tâche du modèle de cloud échoue.

Pour vous assurer que votre pipeline peut s'exécuter pendant plus de 30 minutes, vous pouvez entrer un jeton d'API facultatif. Lorsque vRealize Automation Code Stream appelle le modèle de cloud, le jeton d'API persiste et la tâche de modèle de cloud continue d'utiliser le jeton d'API.

Lorsque vous utilisez le jeton d'API en tant que variable, celui-ci est chiffré. Dans les autres cas, il est utilisé comme texte brut.

## 6 Exécutez le pipeline.

Pour vérifier que chaque tâche s'est terminée avec succès, cliquez sur la tâche dans l'exécution et vérifiez l'état dans les détails du déploiement pour afficher des informations détaillées sur les ressources.

## 7 Dans GitHub, redéfinissez le type de l'instance du serveur WordPress, `small`, sur `medium`.

Lorsque vous validez des modifications, le pipeline se déclenche. Il extrait le code mis à jour à partir du référentiel GitHub et crée votre application.

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

## 8 Exécutez à nouveau le pipeline, vérifiez qu'il a abouti et qu'il a redéfini le type de l'instance de WordPress, `petit`, sur `moyen`.



## Résultats

Félicitations ! Vous avez automatisé la publication de votre application déployée à partir d'un modèle de cloud YAML.

## Étape suivante

Pour plus d'informations sur l'utilisation de vRealize Automation Code Stream, reportez-vous à la section [Chapitre 5 Didacticiels d'utilisation de vRealize Automation Code Stream](#).

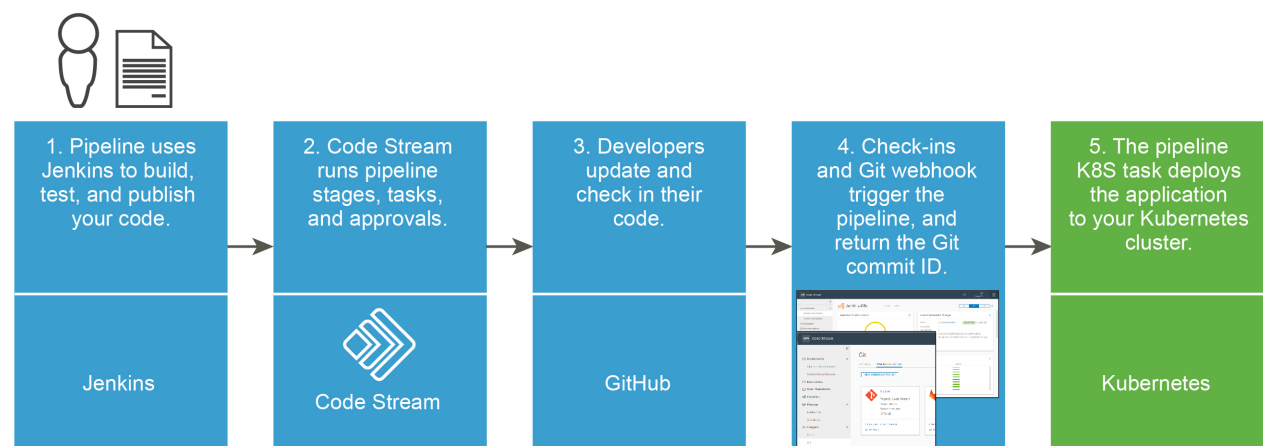
Pour obtenir des références supplémentaires, reportez-vous à la section [Plus de ressources pour les administrateurs et les développeurs vRealize Automation Code Stream](#).

# Automatisation de la publication d'une application dans vRealize Automation Code Stream sur un cluster Kubernetes

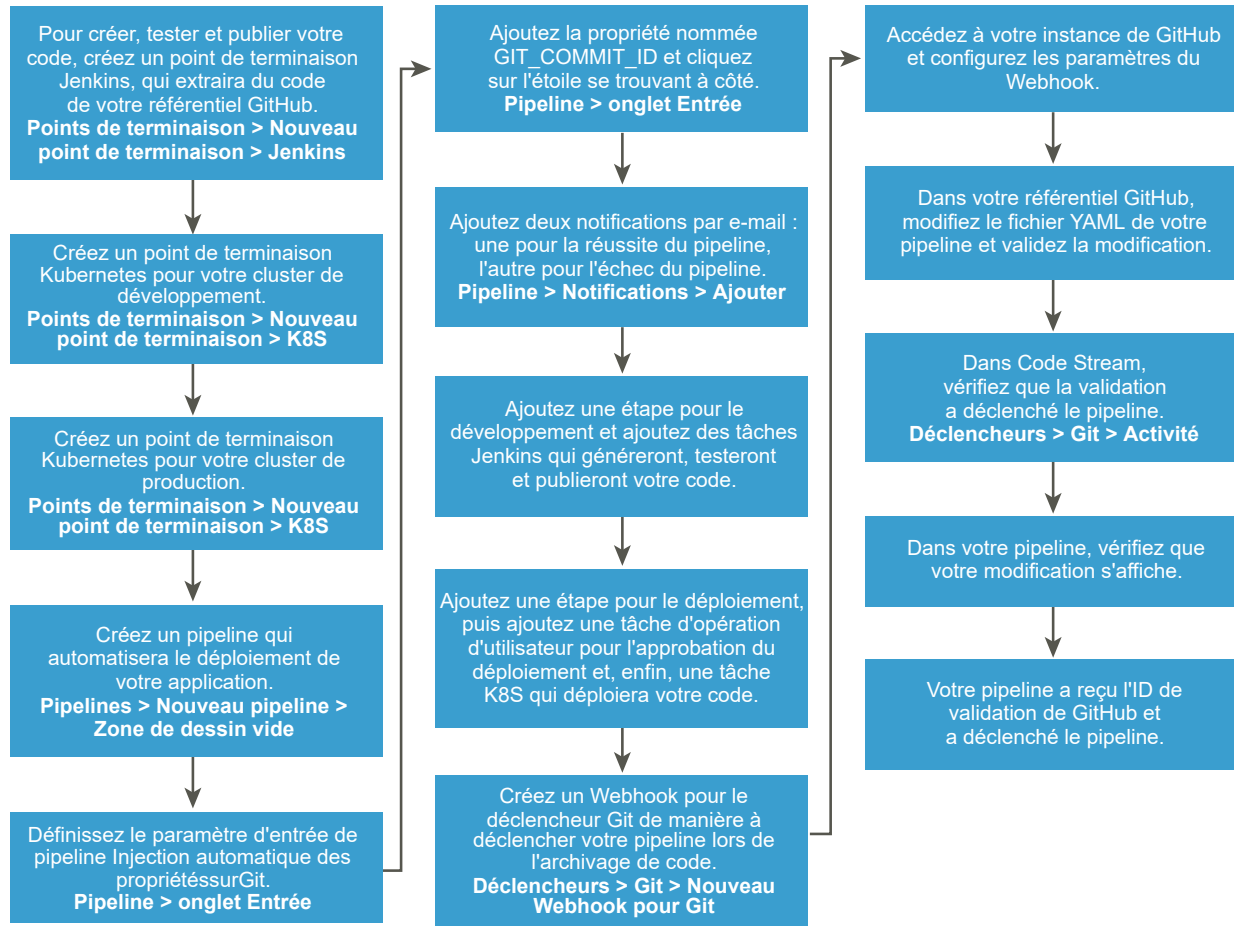
En tant qu'administrateur ou développeur vRealize Automation Code Stream, vous pouvez utiliser vRealize Automation Code Stream et VMware Tanzu Kubernetes Grid Integrated Edition (anciennement VMware Enterprise PKS) pour automatiser le déploiement de vos applications logicielles sur un cluster Kubernetes. Ce cas d'utilisation mentionne d'autres méthodes que vous pouvez utiliser pour automatiser la version de votre application.

Dans ce cas d'utilisation, vous allez créer un pipeline comportant deux étapes, et utiliser Jenkins pour créer et déployer votre application.

- La première étape concerne le développement. Elle utilise Jenkins pour extraire votre code d'une branche de votre référentiel GitHub, puis pour le créer, le tester et le publier.
- La seconde étape concerne le déploiement. Elle exécute une tâche d'opération utilisateur qui nécessite l'approbation des utilisateurs clés pour que le pipeline puisse déployer votre application sur votre cluster Kubernetes.



Les outils de développement, les instances de déploiement et le fichier YAML du pipeline doivent être disponibles afin que votre pipeline puisse créer, tester, publier et déployer votre application. Le pipeline déploiera votre application sur des instances de développement et de production de clusters Kubernetes sur AWS.



Autres méthodes qui automatisent la publication de votre application :

- Au lieu d'utiliser Jenkins pour créer votre application, vous pouvez utiliser la fonctionnalité de build native vRealize Automation Code Stream et un hôte de build Docker.
- Au lieu de déployer votre application sur un cluster Kubernetes, vous pouvez la déployer sur un cluster Amazon Web Services (AWS).

Pour plus d'informations sur l'utilisation de la fonctionnalité de build native vRealize Automation Code Stream et d'un hôte Docker, reportez-vous à :

- [Planification d'une build native CI/CD dans vRealize Automation Code Stream avant d'utiliser le modèle de pipeline intelligent](#)
- [Planification d'une build native CI/CD dans vRealize Automation Code Stream avant d'ajouter des tâches manuellement](#)

#### Conditions préalables

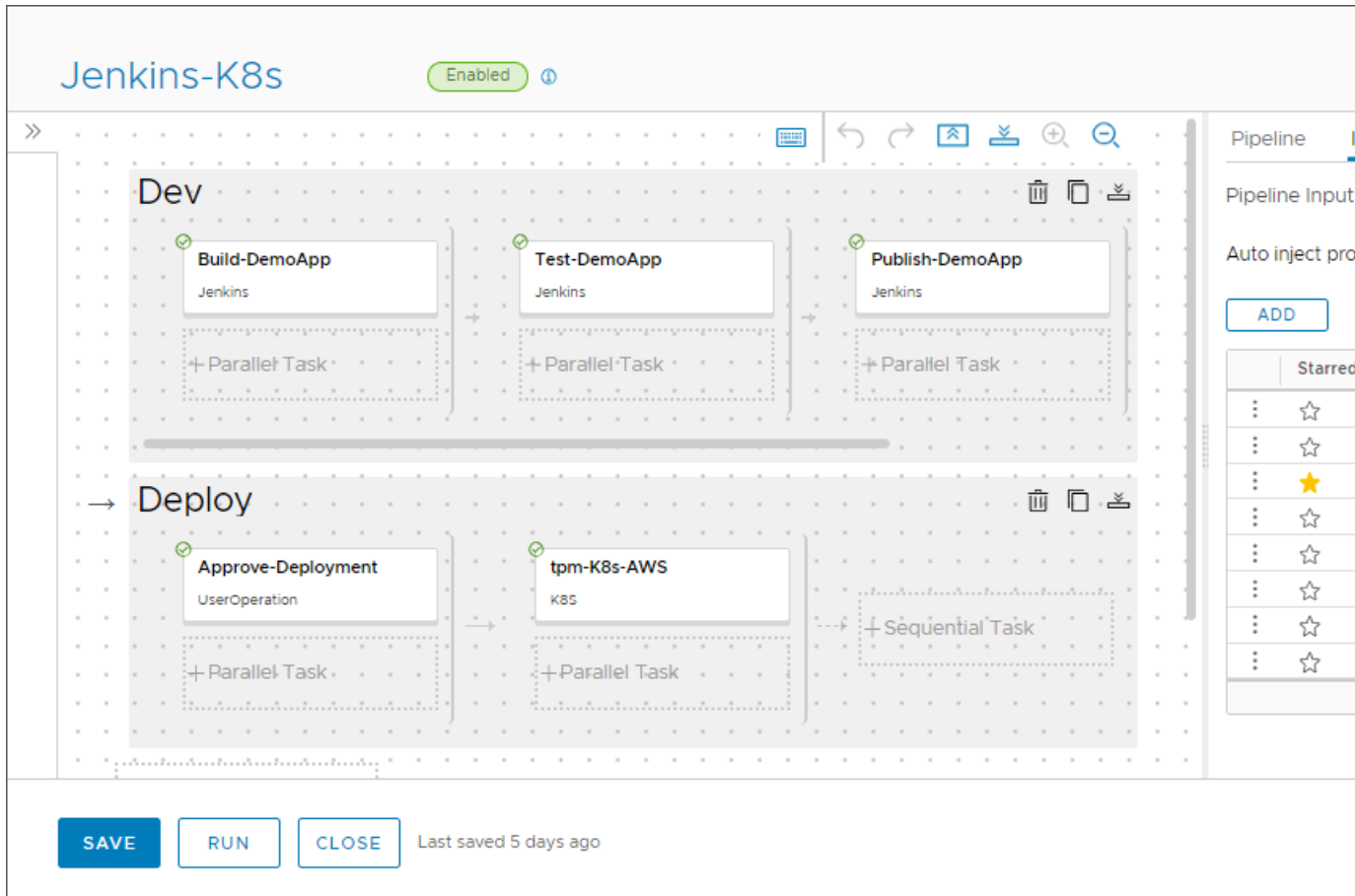
- Vérifiez que le code d'application à déployer se trouve dans un référentiel GitHub opérationnel.
- Vérifiez que vous disposez d'une instance opérationnelle de Jenkins.
- Assurez-vous que vous disposez d'un serveur de messagerie opérationnel.

- Dans vRealize Automation Code Stream, créez un point de terminaison de messagerie qui se connecte à votre serveur de messagerie.
- À des fins de développement et de production, configurez deux clusters Kubernetes sur Amazon Web Services (AWS) où votre pipeline déploiera votre application.
- Vérifiez que le référentiel GitHub contient le code YAML de votre pipeline ou un fichier YAML qui définit les métadonnées et les spécifications de votre environnement.

#### Procédure

- 1 Dans vRealize Automation Code Stream, cliquez sur **Points de terminaison > Nouveau point de terminaison**, puis créez un point de terminaison Jenkins pour l'utiliser dans votre pipeline afin d'extraire le code du référentiel GitHub.
- 2 Pour créer des points de terminaison Kubernetes, cliquez sur **Nouveau point de terminaison**.
  - a Créez un point de terminaison pour votre cluster Kubernetes de développement.
  - b Créez un point de terminaison pour votre cluster Kubernetes de production.

- 3 Créez un pipeline qui déploie un conteneur de votre application, comme WordPress, sur votre cluster Kubernetes de développement, et définissez les propriétés d'entrée pour le pipeline.
  - a Pour permettre à votre pipeline de reconnaître une validation de code dans GitHub qui déclenchera le pipeline, dans le pipeline, cliquez sur l'onglet **Entrée** et sélectionnez **Injection automatique des propriétés**.
  - b Ajoutez la propriété nommée **GIT\_COMMIT\_ID**, puis cliquez sur l'étoile correspondante. Au cours de l'exécution du pipeline, l'ID de validation renvoyé par le déclencheur Git est affiché.



- 4 Ajoutez des notifications pour envoyer un e-mail en cas de réussite ou d'échec du pipeline.
  - a Dans le pipeline, cliquez sur l'onglet **Notifications**, puis sur **Ajouter**.
  - b Pour ajouter une notification par e-mail à la fin de l'exécution du pipeline, sélectionnez **E-mail** et sélectionnez **À l'achèvement**. Ensuite, sélectionnez le serveur de messagerie, entrez les adresses e-mail, puis cliquez sur **Enregistrer**.
  - c Pour ajouter une autre notification par e-mail en cas d'échec du pipeline, sélectionnez **En cas d'échec**, puis cliquez sur **Enregistrer**.

## 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 Ajoutez une étape de développement à votre pipeline et ajoutez des tâches de création, de test et de publication de votre application. Ensuite, validez chaque tâche.
  - a Pour créer votre application, ajoutez une tâche Jenkins qui utilise le point de terminaison Jenkins et exécute une tâche de création à partir du serveur Jenkins. Ensuite, pour que le pipeline récupère votre code, entrez la branche Git comme suit : `{input.GIT_BRANCH_NAME}`
  - b Pour tester votre application, ajoutez une tâche Jenkins qui utilise le point de terminaison Jenkins et exécute une tâche de test à partir du serveur Jenkins. Entrez ensuite la même branche Git.
  - c Pour publier votre application, ajoutez une tâche Jenkins qui utilise le point de terminaison Jenkins et exécute une tâche de publication à partir du serveur Jenkins. Entrez ensuite la même branche Git.

The screenshot displays the Jenkins-K8s pipeline editor. The main workspace shows a pipeline graph with a 'Dev' stage containing three tasks: 'Build-DemoApp', 'Test-DemoApp', and 'Publish-DemoApp', each using the 'Jenkins' provider. Below these is a 'Deploy' task. The right sidebar is configured for the 'Task : Build-DemoApp'.

**Task : Build-DemoApp**

- Task name ⓘ \*
- Type \*
- Continue On Failure
- Execute Task
- Jenkins Job
- Endpoint
- Job
- branchName
- Output Parameters
  - status
  - job
  - jobid

At the bottom, there are buttons for 'SAVE', 'RUN', and 'CLOSE', along with the text 'Last saved 5 days ago'.

- 6 Ajoutez une étape de déploiement à votre pipeline. Ajoutez ensuite une tâche qui nécessite une approbation pour le déploiement de votre application, puis une autre tâche qui déploie l'application sur votre cluster Kubernetes. Ensuite, validez chaque tâche.
  - a Pour exiger une approbation sur le déploiement de votre application, ajoutez une tâche d'opération utilisateur, ajoutez des adresses e-mail pour les utilisateurs qui doivent l'approuver, puis entrez un message. Ensuite, activez **Envoyer un e-mail**.
  - b Pour déployer votre application, ajoutez une tâche Kubernetes. Ensuite, dans les propriétés de tâche Kubernetes, sélectionnez votre cluster Kubernetes de développement, sélectionnez l'action **Créer** et sélectionnez la source de charge utile **Définition locale**. Sélectionnez ensuite votre fichier YAML local.
- 7 Ajoutez un Webhook Git qui permet à vRealize Automation Code Stream d'utiliser le déclencheur Git, qui déclenche votre pipeline lorsque les développeurs valident leur code.

The screenshot displays the 'Git' configuration interface. It includes the following elements:

- Activity** and **Webhooks for Git** tabs.
- Webhook URL**: A long URL starting with 'https://...vmware.com/pipeline/api/git-webhook-listeners/...'.
- Project**: Set to 'Code Stream'.
- Name**: 'muser-Demo-WH'.
- Description**: A text input field.
- Endpoint**: 'tpm-GitHub'.
- Branch**: 'master'.
- Secret token**: A masked field with a 'GENERATE' button.
- File**: A dropdown menu.
- Inclusions**: A dropdown menu with a '+' button.
- Exclusions**: A dropdown menu with a '+' button.
- Prioritize Exclusion**: A toggle switch.
- Trigger**: Radio buttons for 'PUSH' (selected) and 'PULL REQUEST'.
- API token**: A masked field with 'CREATE VARIABLE' and 'GENERATE TOKEN' buttons.
- Pipeline**: 'Jenkins-K8s'.
- Comments**: A text input field.

- 8 Pour tester votre pipeline, accédez à votre référentiel GitHub, mettez à jour le fichier YAML de votre application et validez la modification.
  - a Dans vRealize Automation Code Stream, vérifiez que la validation s'affiche.
  - a Cliquez sur **Déclencheurs > Git > Activité**.
  - b Recherchez le déclencheur de votre pipeline.

- c Cliquez sur **Tableaux de bord > Tableaux de bord du pipeline**.
- d Dans le tableau de bord de votre pipeline, recherchez le GIT\_COMMIT\_ID dans la dernière zone dans laquelle une modification a réussi.

9 Vérifiez votre code de pipeline et vérifiez que la modification s'affiche.

## Résultats

Félicitations ! Vous avez automatisé le déploiement de votre application logicielle sur votre cluster Kubernetes.

## Exemple : Exemple de pipeline YAML déployant une application sur un cluster Kubernetes

Pour le type de pipeline utilisé dans cet exemple, le fichier YAML ressemble au code suivant :

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
eyJzeWlwG9ueS10YW5nbyliZXRhMi5qZnJvZy5pbyI6eyJlc2VybmFtZSI6InRhbmdvLWJldGEyIiwicGFzc3dvcmQIoiJhRGstcmVOLWlUQi1IejciLCJlbWFPbCI6InRhbmdvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoizEdGdVoyOHRZbVYwWVRJNllVUnJMWEPsVGkxdFZFSXRTSG8zIn19
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

```

### Étape suivante

Pour déployer votre application logicielle sur votre cluster Kubernetes de production, effectuez à nouveau la procédure et sélectionnez votre cluster de production.

Pour plus d'informations sur l'intégration de vRealize Automation Code Stream à Jenkins, reportez-vous à la section [Intégration de vRealize Automation Code Stream à Jenkins](#).

## Déploiement d'une application dans vRealize Automation Code Stream vers un déploiement Bleu-vert

Bleu-vert est un modèle de déploiement qui utilise deux hôtes Docker que vous déployez et configurez de la même manière dans un cluster Kubernetes. Avec le modèle de déploiement Bleu-vert, vous réduisez le temps d'arrêt susceptible de se produire dans votre environnement lorsque vos pipelines dans vRealize Automation Code Stream déploient vos applications.

Les instances Bleu et Vert de votre modèle de déploiement servent chacune un objectif différent. Une seule instance à la fois accepte le trafic en direct déployant votre application, et chaque instance accepte ce trafic à des moments spécifiques. L'instance Bleu reçoit la première version de votre application et l'instance Vert en reçoit la seconde.

L'équilibrage de charge dans votre environnement Bleu-vert détermine quelle voie emprunte le trafic en direct lors du déploiement de votre application. En utilisant le modèle Bleu-vert, votre environnement reste opérationnel, les utilisateurs ne remarquent aucun temps d'arrêt, et votre pipeline intègre et déploie continuellement votre application dans votre environnement de production.

Le pipeline que vous créez dans vRealize Automation Code Stream représente votre modèle de déploiement Bleu-vert en deux étapes. Une étape est consacrée au développement, l'autre étape à la production.

**Tableau 5-2. Tâches de l'étape de développement pour le déploiement Bleu-vert**

Type de tâche	Tâche
Kubernetes	Créez un espace de noms pour votre déploiement Bleu-vert.
Kubernetes	Créez une clé secrète pour Docker Hub.
Kubernetes	Créez le service utilisé pour déployer l'application.
Kubernetes	Créez le déploiement Bleu.
Interrogation	Vérifiez le déploiement Bleu.
Kubernetes	Supprimez l'espace de noms.

**Tableau 5-3. Tâches de l'étape de production pour le déploiement Bleu-vert**

Type de tâche	Tâche
Kubernetes	Vert obtient les détails de service de Bleu.
Kubernetes	Obtenez les détails du jeu de répliques Vert.
Kubernetes	Créez le déploiement Vert et utilisez la clé secrète pour extraire l'image de conteneur.
Kubernetes	Mettez le service à jour.
Interrogation	Vérifiez que le déploiement a réussi sur l'URL de production.
Kubernetes	Terminez le déploiement Bleu.
Kubernetes	Supprimez le déploiement Bleu.

Pour déployer l'application de votre propre modèle de déploiement Bleu-vert, vous créez un pipeline dans vRealize Automation Code Stream qui inclut deux étapes. La première étape inclut les tâches Bleu qui déploient votre application sur l'instance Bleu, et la seconde inclut des tâches Vert qui déploient votre application sur l'instance Vert.

Vous pouvez créer votre pipeline à l'aide du modèle de pipeline intelligent CICD. Le modèle crée les étapes et tâches de votre pipeline pour vous, et inclut les sélections de déploiement.

Si vous créez votre pipeline manuellement, vous devez planifier vos étapes de pipeline. Par exemple, reportez-vous à la section [Planification d'une build native CICD dans vRealize Automation Code Stream avant d'ajouter des tâches manuellement](#).

Dans cet exemple, vous utilisez le modèle de pipeline intelligent CICD pour créer votre pipeline Bleu-vert.

## Conditions préalables

- Vérifiez que vous pouvez accéder à un cluster Kubernetes opérationnel sur AWS.
- Vérifiez que vous avez configuré un environnement de déploiement Bleu-vert et configuré vos instances Bleu et Vert pour qu'elles soient identiques.
- Créez un point de terminaison Kubernetes dans vRealize Automation Code Stream qui déploie votre image d'application sur le cluster Kubernetes sur AWS.
- Familiarisez-vous avec l'utilisation du modèle de pipeline intelligent CICD. Reportez-vous à la section [Planification d'une build native CICD dans vRealize Automation Code Stream](#) avant d'utiliser le modèle de pipeline intelligent.


## Procédure

- 1 Cliquez sur **Pipelines > Nouveau pipeline > Modèle intelligent > Modèle CICD**.
- 2 Entrez les informations concernant la partie CI du modèle de pipeline intelligent CICD, puis cliquez sur **Suivant** .  
  
Pour obtenir de l'aide, reportez-vous à la section [Planification d'une build native CICD dans vRealize Automation Code Stream](#) avant d'utiliser le modèle de pipeline intelligent.
- 3 Renseignez la partie CD du modèle de pipeline intelligent.
  - a Sélectionnez les environnements pour le déploiement de votre application. Par exemple, **Dév** et **Prod**.
  - b Sélectionnez le service que le pipeline utilisera pour le déploiement.

- c Dans la zone Déploiement, sélectionnez le point de terminaison du cluster pour les environnements Dév et Prod.
- d Pour le modèle de déploiement Production, sélectionnez **Bleu-vert** , puis cliquez sur **Créer**.

# Modèle intelligent: CI/CD

Étape 2 sur 2

Environnement  \*


Fichiers YAML Kubernetes... \*

☒ Développement ☒ Production

[SÉLECTIONNER](#) [TRAITER](#)



Fichiers traités: codestream.yaml

## Sélectionner un service


	Nom du déploiement	Service	Espace de noms	Image
	codestream-demo	codestream-demo	bgreen1	5

1 services

## Déploiement

Environnement	Point de terminaison de cluster	Espace de noms
Développement	1030Endpoint-Kubernetes 騎家表ホアA中在e國傳B道U8àù'h 	bgreen1-793479
Production	1030Endpoint-Kubernetes 騎家表ホアA中在e國傳B道U8àù'h 	bgreen1

Modèle de déploiement \*

☐ Canary ☐ Mise à niveau propagée ☒ Bleu-vert 

Restaurer ☐

URL de contrôle de santé \*

Entrer l'URL de contrôle de santé

[CRÉER](#) [RETOUR](#) [ANNULER](#)

## Résultats

Félicitations ! Vous avez utilisé le modèle de pipeline intelligent pour créer un pipeline qui déploie votre application sur vos instances Bleu-vert dans votre cluster de production Kubernetes sur AWS.

## Exemple : Exemple de code YAML pour certaines tâches de déploiement Bleu-vert

Le code YAML qui s'affiche dans les tâches de pipeline Kubernetes correspondant à votre déploiement Bleu-vert peut ressembler aux exemples suivants. Une fois que le modèle de pipeline intelligent a créé votre pipeline, vous pouvez, si nécessaire, modifier les tâches de votre propre déploiement.

Code YAML pour créer un exemple d'espace de noms :

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

Code YAML pour créer un exemple de service :

```
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
```

Code YAML pour créer un exemple de déploiement :

```
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
```

### Étape suivante

Pour plus d'informations sur l'utilisation de vRealize Automation Code Stream, reportez-vous à la section [Chapitre 5 Didacticiels d'utilisation de vRealize Automation Code Stream](#).

Pour restaurer un déploiement, reportez-vous à la section [Restauration d'un déploiement dans vRealize Automation Code Stream](#).

Pour obtenir des références supplémentaires, reportez-vous à la section [Plus de ressources pour les administrateurs et les développeurs vRealize Automation Code Stream](#).

## Intégration d'outils de génération, de test et de déploiement propres avec vRealize Automation Code Stream

En tant qu'administrateur ou développeur DevOps, vous pouvez créer des scripts personnalisés qui étendent la capacité de vRealize Automation Code Stream. Avec votre script, vous pouvez intégrer vRealize Automation Code Stream à vos propres outils et API d'intégration continue (CI) et de prestation continue (CD) qui génèrent, testent et déploient vos applications. Les scripts personnalisés sont particulièrement utiles si vous n'exposez pas publiquement vos API d'application.

Votre script personnalisé peut effectuer presque toutes les opérations nécessaires à une intégration à vos outils de génération, de test et de déploiement. Par exemple, il peut utiliser l'espace de travail de votre pipeline pour prendre en charge les tâches CI qui génèrent et testent votre application, ainsi que les tâches CD qui la déploient. Il peut envoyer un message à Slack lorsqu'un pipeline se termine, et bien plus encore.

Vous écrivez votre script personnalisé dans l'un des langages pris en charge. Dans le script, vous incluez votre logique métier, et définissez des entrées et des sorties. Les types de sorties peuvent inclure un nombre, une chaîne, un texte et un mot de passe. Vous pouvez créer plusieurs versions d'un script personnalisé avec une logique d'activité, une entrée et une sortie différentes.

Votre pipeline exécute une version de votre script dans une tâche personnalisée. Les scripts que vous créez résident dans votre instance de vRealize Automation Code Stream.

Lorsqu'un pipeline utilise une intégration personnalisée et que vous tentez de supprimer cette intégration, un message d'erreur s'affiche indiquant que vous ne pouvez pas la supprimer.

La suppression d'une intégration personnalisée supprime toutes les versions de votre script personnalisé. Si vous disposez d'un pipeline existant incluant une tâche personnalisée qui utilise une version du script, ce pipeline va échouer. Pour vous assurer que les pipelines existants n'échouent pas, vous pouvez déconseiller et retirer la version de votre script que vous ne souhaitez plus utiliser. Si aucun pipeline n'utilise cette version, vous pouvez la supprimer.

Tableau 5-4. Actions postérieures à la rédaction de votre script personnalisé

Actions...	Plus d'informations sur cette action...
Ajout d'une tâche personnalisée à votre pipeline.	<p>La tâche personnalisée :</p> <ul style="list-style-type: none"> <li>■ s'exécute sur le même conteneur que les autres tâches CI de votre pipeline.</li> <li>■ inclut les variables d'entrée et de sortie que votre script remplit avant l'exécution de la tâche personnalisée par le pipeline.</li> <li>■ prend en charge plusieurs types de données et différents types de métadonnées que vous définissez comme entrées et sorties dans votre script.</li> </ul>
Sélection de votre script dans la tâche personnalisée.	Vous déclarez les propriétés d'entrée et de sortie dans le script.
Enregistrement de votre pipeline, avant activation et exécution.	Lorsque le pipeline s'exécute, la tâche personnalisée appelle la version du script spécifié et y exécute la logique métier, ce qui intègre votre outil de génération, de test et de déploiement à vRealize Automation Code Stream.
Une fois votre pipeline exécuté, observation des exécutions.	Vérifiez que le pipeline a fourni les résultats attendus.

Cet exemple crée une intégration personnalisée qui connecte vRealize Automation Code Stream à votre instance Slack et publie un message sur un canal Slack.

#### Conditions préalables

- Pour écrire votre script personnalisé, vérifiez que vous disposez de l'un des langages suivantes : Python 2, Python 3, Node.js ; ou de l'un des langages Shell suivants : bash, sh ou zsh.
- Générez une image de conteneur à l'aide du composant d'exécution Node.js ou Python installé.

#### Procédure

##### 1 Créez l'intégration personnalisée.

- Cliquez sur **Intégrations personnalisées > Nouveau** et entrez un nom pertinent.
- Sélectionnez l'environnement d'exécution préféré.
- Cliquez sur **Créer**.

Votre script s'ouvre et affiche le code, qui inclut l'environnement d'exécution requis. Par exemple, `runtime: "nodejs"`. Le script doit inclure le composant d'exécution que l'image du générateur utilise, afin que la tâche personnalisée que vous ajoutez à votre pipeline aboutisse lorsque le pipeline s'exécute. Sinon, la tâche personnalisée échoue.

Les principaux aspects de votre intégration personnalisée YAML incluent l'exécution, le code, les propriétés d'entrée et les propriétés de sortie. Cette procédure explique différents types et la syntaxe.

Clés YAML d'intégration personnalisée	Description
<code>runtime</code>	Environnement d'exécution des tâches où vRealize Automation Code Stream exécute le code, qui peut être l'une de ces chaînes insensibles à la casse : <ul style="list-style-type: none"> <li>■ <code>nodejs</code></li> <li>■ <code>python2</code></li> <li>■ <code>python3</code></li> <li>■ <code>shell</code></li> </ul> Si rien n'est indiqué, <code>shell</code> est la valeur par défaut supposée.
<code>code</code>	Logique métier personnalisée à exécuter dans le cadre de la tâche personnalisée.
<code>inputProperties</code>	Tableau de propriétés d'entrée à capturer dans le cadre de la configuration de la tâche personnalisée. Ces propriétés sont normalement utilisées dans le code.
<code>outputProperties</code>	Tableau de propriétés de sortie que vous pouvez exporter à partir de la tâche personnalisée à propager au pipeline.

- 2 Déclarez les propriétés d'entrée dans votre script en utilisant les types de données et les métadonnées disponibles.

Les propriétés d'entrée sont transmises sous forme de contexte à votre script dans la section `code` : du fichier YAML.

Touches d'entrée YAML de tâche personnalisée	Description	Requis
<code>type</code>	Types d'entrée à rendre : <ul style="list-style-type: none"> <li>■ <code>texte</code></li> <li>■ <code>zone de texte</code></li> <li>■ <code>nombre</code></li> <li>■ <code>case à cocher</code></li> <li>■ <code>mot de passe</code></li> <li>■ <code>sélectionner</code></li> </ul>	Oui
<code>name</code>	Nom ou chaîne de l'entrée de la tâche personnalisée, qui est injectée dans le code YAML d'intégration personnalisée. Doit être unique pour chaque propriété d'entrée définie pour une intégration personnalisée.	Oui
<code>title</code>	Libellé de chaîne de texte de la propriété d'entrée pour la tâche personnalisée sur le canevas du modèle de pipeline. S'il est laissé vide, <code>name</code> est utilisé par défaut.	Non
<code>required</code>	Détermine si un utilisateur doit entrer la propriété d'entrée lorsqu'il configure la tâche personnalisée. Définissez sur <code>true</code> ou <code>false</code> . Lorsque la valeur est <code>true</code> , si un utilisateur ne fournit pas de valeur lorsqu'il configure la tâche personnalisée sur le canevas du pipeline, l'état de la tâche reste non configuré.	Non
<code>placeholder</code>	Texte par défaut pour la zone d'entrée de la propriété d'entrée lorsqu'aucune valeur n'est présente. Correspond à l'attribut <code>html placeholder</code> . Uniquement pris en charge pour certains types de propriétés d'entrée.	Non



Touches d'entrée YAML de tâche personnalisée	Description	Requis
<b>defaultValue</b>	Valeur par défaut qui renseigne la zone d'entrée de la propriété d'entrée lorsque la tâche personnalisée s'affiche sur la page du modèle de pipeline.	Non
<b>bindable</b>	Détermine si la propriété d'entrée accepte les variables de symbole dollar lors de la modélisation de la tâche personnalisée sur le canevas du pipeline. Ajoute l'indicateur <b>\$</b> en regard du titre. Uniquement pris en charge pour certains types de propriétés d'entrée.	Non
<b>labelMessage</b>	Chaîne qui agit comme une info-bulle d'aide pour les utilisateurs. Ajoute une icône d'info-bulle <b>i</b> en regard du titre d'entrée.	Non
<b>enum</b>	<p>Prend un tableau de valeurs qui affiche les options de sélection de propriétés d'entrée. Uniquement pris en charge pour certains types de propriétés d'entrée.</p> <p>Lorsqu'un utilisateur sélectionne une option et l'enregistre pour la tâche personnalisée, la valeur de <b>inputProperty</b> correspond à cette valeur et figure dans la modélisation de tâche personnalisée.</p> <p>Par exemple, la valeur 2015.</p> <ul style="list-style-type: none"> <li>■ 2015</li> <li>■ 2016</li> <li>■ 2017</li> <li>■ 2018</li> <li>■ 2019</li> <li>■ 2020</li> </ul>	Non
<b>options</b>	<p>Prend un tableau d'objets en utilisant <b>optionKey</b> et <b>optionValue</b>.</p> <ul style="list-style-type: none"> <li>■ <b>optionKey</b>. Valeur propagée à la section code de la tâche.</li> <li>■ <b>optionValue</b>. Chaîne qui affiche l'option dans l'interface utilisateur.</li> </ul> <p>Uniquement pris en charge pour certains types de propriétés d'entrée.</p> <p>Options :</p> <p><b>optionKey</b> : key1. Lorsque cette option est sélectionnée et enregistrée pour la tâche personnalisée, la valeur de <b>inputProperty</b> correspond à <b>key1</b> dans la section code.</p> <p><b>optionValue</b> : « Label for 1 ». Valeur d'affichage de <b>key1</b> dans l'interface utilisateur et ne figure nulle part ailleurs pour la tâche personnalisée.</p> <p><b>optionKey</b> : key2</p> <p><b>optionValue</b> : « Label for 2 »</p> <p><b>optionKey</b> : key3</p> <p><b>optionValue</b> : « Label for 3 »</p>	Non

Touches d'entrée YAML de tâche personnalisée	Description	Requis
<b>minimum</b>	Prend un nombre qui agit comme la valeur minimale qui est valide pour cette propriété d'entrée. Uniquement pris en charge pour la propriété d'entrée de type nombre.	Non
<b>maximum</b>	Prend un nombre qui agit comme la valeur maximale valide pour cette propriété d'entrée. Uniquement pris en charge pour la propriété d'entrée de type nombre.	Non

Tableau 5-5. Types de données et métadonnées pris en charge pour les scripts personnalisés

Types de données pris en charge	Métadonnées prises en charge pour l'entrée
<ul style="list-style-type: none"> <li>■ Chaîne</li> <li>■ Texte</li> <li>■ Liste : sous forme de liste de n'importe quel type</li> <li>■ Carte : sous la forme map[chaîne]any</li> <li>■ Sécurisé : rendu sous forme de zone de texte pour mot de passe, chiffrée lorsque vous enregistrez la tâche personnalisée</li> <li>■ Nombre</li> <li>■ Booléen : s'affiche sous forme de zones de texte</li> <li>■ URL : identique au type Chaîne, avec validation supplémentaire</li> <li>■ Sélection, case d'option</li> </ul>	<ul style="list-style-type: none"> <li>■ type : Une chaîne   Texte...</li> <li>■ valeur par défaut : valeur par défaut</li> <li>■ options : liste ou carte d'options à utiliser avec la sélection ou la case d'option</li> <li>■ min. : valeur ou taille minimale</li> <li>■ max. : valeur ou taille maximale</li> <li>■ titre : nom détaillé de la zone de texte</li> <li>■ espace réservé : espace réservé dans l'interface utilisateur</li> <li>■ description : devient une info-bulle</li> </ul>

Par exemple :

```
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 Déclarez les propriétés de sortie dans votre script.

Le script capture les propriétés de sortie à partir de la section de la logique métier `code` : de votre script, dans laquelle vous déclarez le contexte de la sortie.

Lorsque le pipeline s'exécute, vous pouvez entrer le code de réponse correspondant au résultat de la tâche. Par exemple, **200**.

Clés que vRealize Automation Code Stream prend en charge pour chaque **outputProperty**.

key	Description
type	Inclut actuellement une seule valeur de <b>label</b> .
name	Clé que le bloc de code de l'intégration personnalisée YAML émet.
title	Étiquette dans l'interface utilisateur qui affiche <b>outputProperty</b> .

Par exemple :

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

- 4 Pour interagir avec l'entrée et la sortie de votre script personnalisé, obtenez une propriété d'entrée ou définissez une propriété de sortie en utilisant **context**.

Pour une propriété d'entrée : `(context.getInput("key"))`

Pour une propriété de sortie : `(context.setOutput("key", "value"))`

Pour Node.js :

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

Pour Python :

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

Pour Shell :

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

- 5 Dans la section `code:`, déclarez l'ensemble de la logique métier pour votre intégration personnalisée.

Par exemple, avec l'environnement d'exécution Node.js :

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

  //Get the entered message from task config page and assign it to message var
  var message = context.getInput("message");
```

```

var slackPayload = JSON.stringify(
  {
    text: message
  });

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

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

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

```

- 6 Avant de procéder au contrôle de la version et à la publication de votre script d'intégration personnalisé, téléchargez le fichier de contexte pour Python ou Node.js, et testez la logique d'activité que vous avez incluse dans votre script.
  - a Placez le curseur dans le script, puis cliquez sur le bouton Fichier de contexte en haut du canevas. Par exemple, si votre script est écrit dans le langage de programmation Python, cliquez sur **CONTEXT.PY**.
  - b Modifiez le fichier et enregistrez-le.
  - c Sur votre système de développement, exécutez et testez votre script personnalisé à l'aide du fichier de contexte.
- 7 Appliquez une version à votre script d'intégration personnalisé.
  - a Cliquez sur **Version**.
  - b Entrez les informations relatives à la version.

- c Cliquez sur **Versión** pour pouvoir sélectionner le script dans votre tâche personnalisée.
- d Pour créer la version, cliquez sur **Créer**.

Création de la version

Version \* 1.0

Description New

Journal des modifications New for 1.0

Publier la version ☒

ANNULER CRÉER

- 8 Pour enregistrer le script, cliquez sur **Enregistrer**.
- 9 Dans votre pipeline, configurez l'espace de travail.
  - a Cliquez sur l'onglet **Espace de travail**.
  - b Sélectionnez l'hôte Docker et l'URL de l'image du générateur.

Demo-customTask-nodejs Activé

Espace de travail Entrée Modèle Sortie

Fournissez des détails sur le conteneur et l'hôte pour exécuter des tâches d'intégration continue.

Hôte \* Docker-saas

URL de l'image du ... \* node:latest

Registre d'image --Sélectionner un point de terminaison de registre d'image--

Répertoire de travail

Cache

Limite de CPU \*

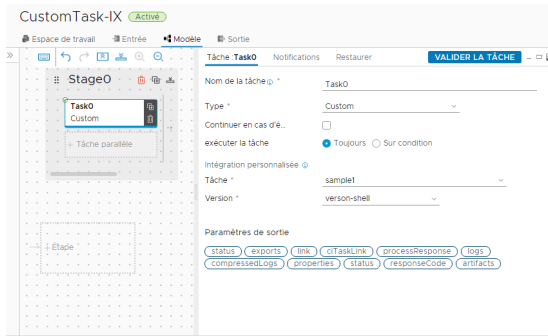
Limite de mémoire \*

Clone Git ☐

Si ce pipeline est lié au Git via un Webhook, le pipeline se déclenche sur les événements Git. Pour les tâches de CI, le référentiel Git lié (détails des paramètres du Webhook Git) est automatiquement cloné dans l'espace de travail.

- 10 Ajoutez une tâche personnalisée à votre pipeline et configurez-la.
  - a Cliquez sur l'onglet **Modèle**.
  - b Ajoutez une tâche, sélectionnez le type **personnalisé** et entrez un nom pertinent.

- c Sélectionnez votre script d'intégration personnalisé et sa version.
  - d Pour afficher un message personnalisé dans Slack, entrez le texte du message.
- Tout texte que vous entrez remplace la `defaultValue` dans votre script d'intégration personnalisé. Par exemple :

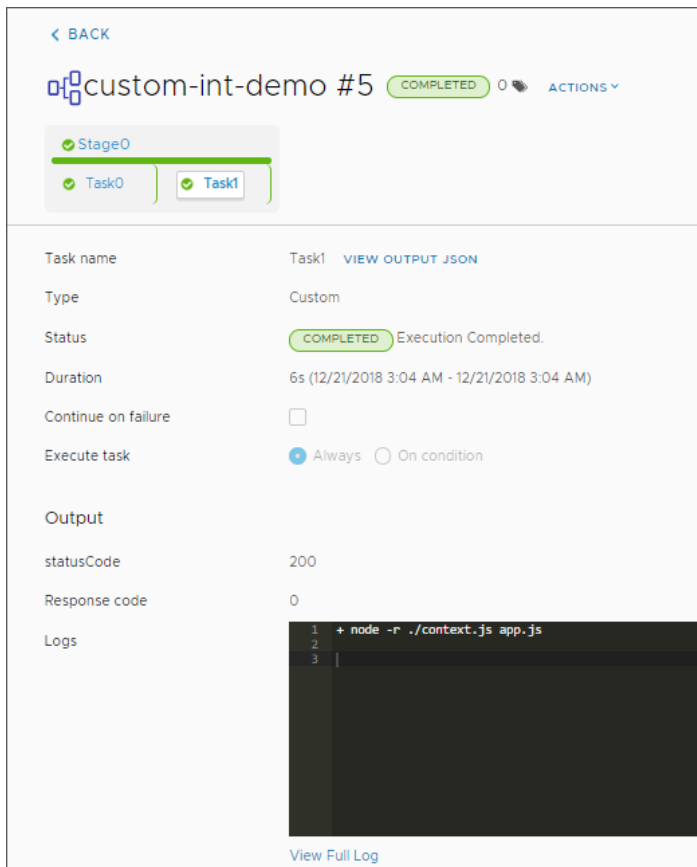


- 11 Enregistrez et activez votre pipeline.
  - a Cliquez sur **Enregistrer**.
  - b Dans l'onglet Pipeline, cliquez sur **Activer le pipeline** afin que le cercle se déplace vers la droite.
- 12 Exécutez votre pipeline.
  - a Cliquez sur **Exécuter**.
  - b Observez l'exécution du pipeline.

- c Vérifiez que la sortie inclut le code d'état, le code de réponse, l'état et le résultat déclaré attendus.

Vous avez défini **statusCode** comme propriété de sortie. Par exemple, un **statusCode** de 200 peut indiquer une publication Slack réussie et un **responseCode** de 0 peut indiquer la réussite sans erreur du script.

- d Pour vérifier le résultat dans les journaux d'exécution, cliquez sur **Exécutions**, sur le lien vers votre pipeline, puis sur la tâche, et examinez les données consignées. Par exemple :



- 13 Si une erreur se produit, résolvez le problème et réexécutez le pipeline.

Par exemple, si un fichier ou un module de l'image de base est manquant, vous devez créer une autre image de base qui inclut le fichier manquant. Ensuite, fournissez le fichier Docker et transmettez l'image via le pipeline.

## Résultats

Félicitations ! Vous avez créé un script d'intégration personnalisé qui connecte vRealize Automation Code Stream à votre instance de Slack et publie un message sur un canal Slack.

## Étape suivante

Continuez à créer des intégrations personnalisées pour prendre en charge l'utilisation de tâches personnalisées dans vos pipelines, afin de pouvoir étendre la capacité de vRealize Automation Code Stream dans l'automatisation du cycle de vie de votre version logicielle.

## Utilisation d'une API REST pour intégrer vRealize Automation Code Stream à d'autres applications

vRealize Automation Code Stream fournit un plug-in REST qui vous permet d'intégrer vRealize Automation Code Stream à d'autres applications qui utilisent une API REST afin de pouvoir développer et fournir en continu des applications logicielles qui doivent interagir entre elles. Le plug-in REST appelle une API qui échange des informations entre vRealize Automation Code Stream et une autre application.

Avec le plug-in REST, vous pouvez effectuer les opérations suivantes :

- Intégrez des systèmes basés sur des API REST externes dans un pipeline vRealize Automation Code Stream.
- Intégrez un pipeline vRealize Automation Code Stream dans le flux de systèmes externes.

Le plug-in REST fonctionne avec n'importe quelle API REST et prend en charge les méthodes GET, POST, PUT, PATCH et DELETE pour échanger des informations entre vRealize Automation Code Stream et d'autres applications.

**Tableau 5-6. Préparation d'un pipeline pour communiquer sur l'API REST**

Actions...	Résultat...
Ajoutez une tâche REST à votre pipeline.	La tâche REST transmet les informations entre les applications et peut fournir des informations relatives à l'état pour une tâche consécutive de l'étape du pipeline.
Dans la tâche REST, sélectionnez l'action REST et indiquez l'URL.	La tâche de pipeline appelle l'URL lorsque le pipeline s'exécute. Pour les actions POST, PUT et PATCH, vous devez inclure une charge utile. Dans la charge utile, vous pouvez lier vos propriétés de pipeline et de tâche lorsque le pipeline s'exécute.
Observez l'exemple suivant.	Exemple d'utilisation du plug-in REST : Vous pouvez ajouter une tâche REST pour créer une balise pour une build sur une validation Git et demander à la tâche de publier une demande pour obtenir l'ID d'archivage auprès du référentiel. La tâche peut envoyer une charge utile à votre référentiel et créer une balise pour la build, et le référentiel peut renvoyer la réponse avec la balise.

À l'instar de l'utilisation du plug-in REST pour appeler une API, vous pouvez inclure une tâche d'interrogation dans votre pipeline pour appeler une API REST et l'interroger jusqu'à ce qu'elle se termine et que la tâche de pipeline réponde aux critères de sortie.

Vous pouvez également utiliser les API REST pour importer et exporter un pipeline, et utiliser les exemples de script pour exécuter un pipeline.

Cette procédure permet d'obtenir une URL simple.

### Procédure

- 1 Pour créer un pipeline, cliquez sur **Pipelines > Nouveau pipeline > Canevas vide**.
- 2 Dans votre étape de pipeline, cliquez sur **+ Tâche séquentielle**.



### 3 Dans le volet des tâches, ajoutez la tâche REST :

- a Entrez un nom pour la tâche.
- b Dans le menu déroulant Type, sélectionnez **REST**.
- c Dans la zone Demande REST, sélectionnez **GET**.

Pour que la tâche REST demande des données auprès d'une autre application, vous devez sélectionner la méthode GET. Pour envoyer des données à une autre application, vous sélectionnez la méthode POST.

- d Entrez l'URL qui identifie le point de terminaison de l'API REST. Par exemple, `https://www.google.com`.

Pour qu'une tâche REST importe des données d'une autre application, vous pouvez inclure la variable de charge utile. Par exemple, pour une action d'importation, vous pouvez entrer `${Stage0.export.responseBody}`. Si la taille des données de réponse dépasse 5 Mo, la tâche REST peut échouer.

- e Pour accorder une autorisation pour la tâche, cliquez sur **Ajouter des en-têtes** et entrez une clé et une valeur d'en-tête.

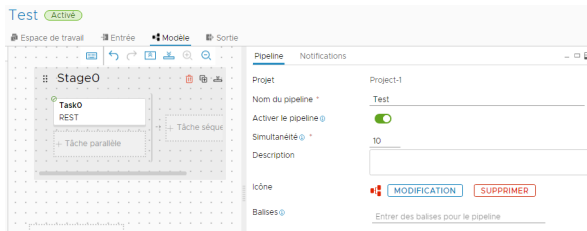
The screenshot displays the vRealize Automation Code Stream interface. The main workspace shows a 'Stage0' containing a 'Task0' of type 'REST'. The right-hand panel is configured for 'Task : Task0' with the following settings:

- 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
    - Content-Type: application
- Output Parameters:** status

At the bottom of the interface, there are buttons for 'SAVE', 'RUN', and 'CLOSE', along with a status message: 'Last saved an hour ago'.

### 4 Pour enregistrer votre pipeline, cliquez sur **Enregistrer**.

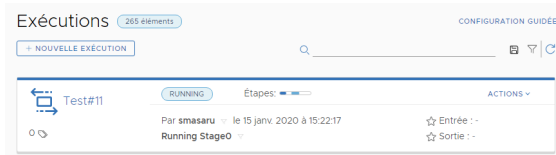
## 5 Dans l'onglet Pipeline, cliquez sur **Activer le pipeline**.



## 6 Cliquez sur **Enregistrer**, puis sur **Fermer**.

## 7 Cliquez sur **Exécuter**.

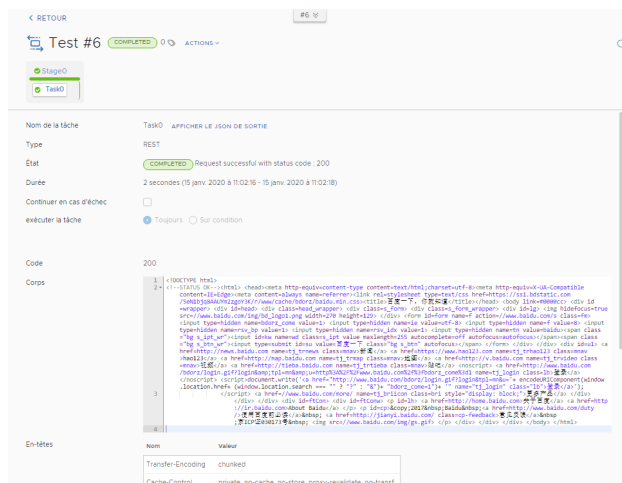
## 8 Pour surveiller l'exécution du pipeline, cliquez sur **Exécutions**.



## 9 Pour vérifier que le plug-in REST renvoie les informations que vous attendez, examinez l'exécution du pipeline et les résultats de la tâche.

- Une fois le pipeline terminé, pour vérifier que l'autre application a renvoyé les données que vous avez demandées, cliquez sur le lien vers l'exécution du pipeline.
- Cliquez sur la tâche REST dans le pipeline.
- Dans l'exécution du pipeline, cliquez sur la tâche, examinez-en les détails et vérifiez que le plug-in REST a renvoyé les résultats attendus.

Les détails de la tâche affichent le code, le corps, les clés d'en-tête et les valeurs de la réponse.



## 10 Pour examiner le résultat JSON, cliquez sur **EXAMINER LE RÉSULTAT JSON**.

```

Stage0.Task0.output
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
    =RTUKVjVhyg9KvAZR1S8yCCSEw8WosYfn9MwIDFQ1N5fNd5DavrXUm5B3J8PyKXK1Z_zRnp3usxttMpd7YiqRUOSfMkTC7cTERbd
    UmOnj3cTpphe3PHIXJPGHnTSZeweb3cXtjvIHVolS85ezVxatSRyFcg0B_XIHZBkq88uwl1aE; 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=\"aWwW/ydugKgr9CHU6QQGz==\"><function() {window.google={kEI: 'cnf8W6Kp3IEvKwXk-aLODA', kEXPI: '0
    ,1353747,57,50,1150,454,303,1017,1120,286,690,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457
    ,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,6056,636,2239
    ,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284
    ,2,579,727,612,1820,58,2,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
    ,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8
    ,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483
    ,698,59,318,273,108,167,323,744,101,1119,30,363,557,430,135,145,155,497,2,718,383,970,487,47,1080,901
    ,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

### Résultats

Félicitations ! Vous avez configuré une tâche REST qui a appelé une API REST et échangé des informations entre vRealize Automation Code Stream et une autre application en utilisant le plug-in REST.

### Étape suivante

Continuez à utiliser des tâches REST dans vos pipelines pour exécuter des commandes et intégrer vRealize Automation Code Stream à d'autres applications afin de pouvoir développer et proposer vos applications logicielles. Envisagez l'utilisation de tâches d'interrogation pour interroger l'API jusqu'à ce qu'elle se termine et que la tâche de pipeline réponde aux critères de sortie.

# Connexion de vRealize Automation Code Stream à des points de terminaison

## 6

vRealize Automation Code Stream s'intègre aux outils de développement par le biais de plug-ins. Les plug-ins pris en charge sont notamment Jenkins, Bamboo, vRealize Operations, Bugzilla, Team Foundation Server, Git, etc.

Vous pouvez également développer vos propres plug-ins pour intégrer vRealize Automation Code Stream à d'autres applications de développement.

Pour intégrer vRealize Automation Code Stream à JIRA, un plug-in externe n'est pas nécessaire, car la capacité de création de tickets JIRA est intégrée dans vRealize Automation Code Stream comme type de notification. Pour créer des tickets JIRA sur l'état du pipeline, vous devez ajouter un point de terminaison JIRA.

Ce chapitre contient les rubriques suivantes :

- [Présentation des points de terminaison dans vRealize Automation Code Stream](#)
- [Intégration de vRealize Automation Code Stream à Jenkins](#)
- [Comment intégrer vRealize Automation Code Stream à Git](#)
- [Intégration de vRealize Automation Code Stream à Gerrit](#)
- [Intégration de vRealize Automation Code Stream à vRealize Orchestrator](#)

## Présentation des points de terminaison dans vRealize Automation Code Stream

Un point de terminaison est une instance d'une application DevOps qui se connecte à vRealize Automation Code Stream pour mettre à la disposition de vos pipelines les données nécessaires à leur exécution, comme une source de données, un référentiel ou un système de notification.

Votre rôle dans vRealize Automation Code Stream détermine votre utilisation des points de terminaison.

- Les administrateurs et les développeurs peuvent créer, mettre à jour, supprimer et afficher des points de terminaison.
- Les administrateurs peuvent marquer un point de terminaison comme étant limité et exécuter des pipelines qui utilisent des points de terminaison limités.

- Les utilisateurs qui disposent du rôle de visualisation peuvent voir les points de terminaison, mais ne peuvent pas les créer, les mettre à jour ou les supprimer.

Pour plus d'informations, reportez-vous à la section [Gestion de l'accès et des approbations utilisateur dans vRealize Automation Code Stream](#).

Pour connecter vRealize Automation Code Stream à un point de terminaison, vous ajoutez une tâche dans votre pipeline et la configurez pour qu'elle communique avec le point de terminaison. Pour vérifier que vRealize Automation Code Stream peut se connecter au point de terminaison, cliquez sur **Valider**. Ensuite, lorsque vous exécutez le pipeline, votre tâche de pipeline se connecte au point de terminaison pour exécuter la tâche.

**Tableau 6-1. Points de terminaison pris en charge par vRealize Automation Code Stream**

Point de terminaison	Opérations possibles	Versions prises en charge	Configuration requise
Bamboo	Crée des plans de build.	6.9.*	
Docker	Les builds natives peuvent utiliser des hôtes Docker pour le déploiement.		Lorsqu'un pipeline inclut une image depuis Docker Hub, vous devez vous assurer que l'image intègre cURL avant d'exécuter le pipeline. Lorsque le pipeline s'exécute, vRealize Automation Code Stream télécharge un fichier binaire qui utilise cURL pour exécuter des commandes.
Registre Docker	Enregistre les images de conteneur afin qu'un hôte de build Docker puisse extraire des images.	2.7.1	
Gerrit	Se connecte à un serveur Gerrit pour les révisions et le déclencheur	2.14.*	
Git	Déclenche des pipelines lorsque les développeurs mettent à jour le code et l'archivent dans le référentiel.	Git Hub Enterprise 2.1.8 Git Lab Enterprise 1 1.9.12-ee	
Jenkins	Génère des artefacts de code.	1.6.* et 2.*	
Jira	Crée un ticket JIRA en cas d'échec d'une tâche de pipeline.	8.3.*	
Kubernetes	Automatise les étapes de déploiement, de mise à l'échelle et de gestion des applications en conteneur.	1.9.*	
PowerShell	Créez des tâches qui exécutent des scripts PowerShell sur des machines Windows ou Linux.	4 et 5	
SSH	Créez des tâches qui exécutent des scripts SSH sur des machines Windows ou Linux.	7.0	

Tableau 6-1. Points de terminaison pris en charge par vRealize Automation Code Stream (suite)

Point de terminaison	Opérations possibles	Versions prises en charge	Configuration requise
TFS, Team Foundation Server	Gère le code source, les builds automatisées, les tests et les activités associées.	2015 et 2017	
vRealize Orchestrator	Organise et automatise les workflows dans votre processus de génération.	7.* et 8.*	

## Exemple de code YAML pour un point de terminaison GitHub

Cet exemple de code YAML définit un point de terminaison GitHub auquel vous pouvez vous référer dans une tâche Git.

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

## Intégration de vRealize Automation Code Stream à Jenkins

vRealize Automation Code Stream fournit un plug-in Jenkins qui déclenche les travaux Jenkins chargés de générer et de tester votre code source. Le plug-in Jenkins exécute des cas de test et peut utiliser des scripts personnalisés.

Pour exécuter un travail Jenkins dans votre pipeline, utilisez un serveur Jenkins et ajoutez le point de terminaison Jenkins dans vRealize Automation Code Stream. Ensuite, vous créez un pipeline et y ajoutez une tâche Jenkins.

### Conditions préalables

- Configurez un serveur Jenkins qui exécute la version 1.561 ou une version ultérieure.

- Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).
- Vérifiez qu'un travail existe sur le serveur Jenkins afin que votre tâche de pipeline puisse l'exécuter.

#### Procédure

- 1 Ajoutez et validez un point de terminaison Jenkins.
  - a Cliquez sur **Points de terminaison > Nouveau point de terminaison**.
  - b Sélectionnez un projet et, pour le type de point de terminaison, sélectionnez **Jenkins**. Ensuite, entrez un nom et une description.
  - c Si ce point de terminaison est un composant stratégique de votre infrastructure, activez **Marquer comme restreint**.
  - d Entrez l'URL du serveur Jenkins.

- e Saisissez le nom d'utilisateur et le mot de passe de connexion au serveur Jenkins. Entrez ensuite les informations restantes.

**Tableau 6-2. Informations restantes pour le point de terminaison Jenkins**

Entrée du point de terminaison	Description
Chemin d'accès au dossier	<p>Chemin d'accès au dossier qui regroupe vos travaux. Jenkins peut exécuter tous les travaux du dossier. Vous pouvez créer des sous-dossiers. Par exemple :</p> <ul style="list-style-type: none"> <li>■ <code>folder_1</code> peut inclure <code>job_1</code>.</li> <li>■ <code>folder_1</code> peut inclure <code>folder_2</code>, qui peut inclure <code>job_2</code>.</li> </ul> <p>Lorsque vous créez un point de terminaison pour <code>folder_1</code>, le chemin d'accès au dossier est <code>job/folder_1</code> et le point de terminaison répertorie uniquement <code>job_1</code>. Pour obtenir la liste des travaux dans le dossier enfant nommé <code>folder_2</code>, vous devez créer un autre point de terminaison qui utilise le chemin d'accès au dossier, comme <code>/job/folder_1/job/folder_2/</code>.</p>
URL	URL hôte du serveur Jenkins. Entrez l'URL sous la forme <code>protocol://host:port</code> . Par exemple : <code>http://192.10.121.13:8080</code>
Intervalle d'interrogation	Durée de l'intervalle d'interrogation du serveur Jenkins par vRealize Automation Code Stream, à la recherche de mises à jour.
Nombre de relances de la demande	Nombre de fois que la demande de génération planifiée concernant le serveur Jenkins est relancée.
Temps d'attente avant relance	Nombre de secondes d'attente avant la relance de la demande de génération concernant le serveur Jenkins.

- f Cliquez sur **Valider** et vérifiez que le point de terminaison se connecte à vRealize Automation Code Stream. S'il ne se connecte pas, corrigez les erreurs éventuelles, puis cliquez sur **Enregistrer**.

Modifier le point de terminaison

Projet: test1

Type: Jenkins

Nom \*: aa

Description:

Marquer comme recommandé: ☐ non restreint

URL \*: www.baidu.com

Username: username

Password: password [CRÉER UNE VARIABLE](#)

Folder Path: /job/DevFolder/

Poll Interval (sec) \*: 15

Request Retries \*: 5

Retry Wait Time (s) \*: 60

[ENREGISTRER](#) [VALIDER](#) [ANNULER](#)

- 2 Pour créer votre code, créez un pipeline et ajoutez une tâche qui utilise votre point de terminaison Jenkins.
  - a Cliquez sur **Pipelines > Nouveau pipeline > Canevas vide**.
  - b Cliquez sur l'étape par défaut.
  - c Dans la zone Tâche, entrez un nom pour la tâche.



- d Sélectionnez le type de tâche **Jenkins**.
- e Sélectionnez le point de terminaison Jenkins que vous avez créé.
- f Dans le menu déroulant, sélectionnez, sur le serveur Jenkins, un travail que votre pipeline exécutera.
- g Saisissez les paramètres correspondant au travail.
- h Entrez le jeton d'authentification du travail Jenkins.

**Build and Deploy** Enabled ⓘ

Task : *Build* Notifications VALIDATE TASK

Task name ⓘ \* Build

Type \* Jenkins ▾

Continue On Failure ☐

Execute Task ☒ Always ☐ On Condition

Jenkins

Endpoint aa ▾

Job \* add\_numbers ▾

Num1 \$ 22

Num2 \$ 22

Token


Output Parameters

status job jobId jobResults jobUrl

SAVE RUN CLOSE Last saved a month ago

### 3 Activez et exécutez votre pipeline, et affichez-en l'exécution.

[< BACK](#)

 **Build and Deploy #28** COMPLETED 0 [ACTIONS](#) ▼

Stage0

Build

Test

Approval for Deployment

Deployment

Wait for application to start

Task name

Build

[VIEW OUTPUT JSON](#)

Type

Jenkins

Status

COMPLETED Execution Completed.

Duration

11s (08/06/2018 12:27 AM - 08/06/2018 12:27 AM)

Continue On Failure

☐

Execute Task

☒ Always ☐ On Condition

Jenkins Job

Endpoint

aa

Job Name

add\_numbers

Job ID

1428

Job URL

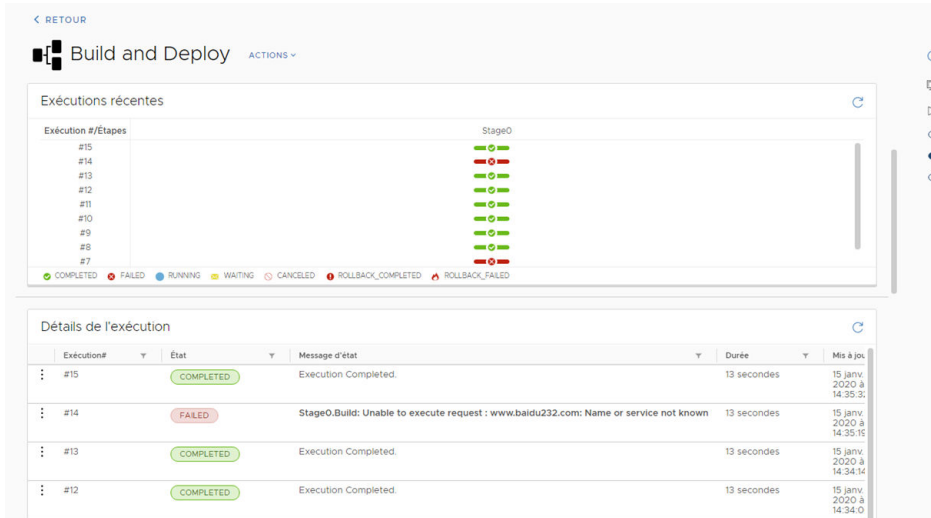
[http://.../job/add\\_numbers/1428/](http://.../job/add_numbers/1428/)

Job Result

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

#### 4 Examinez les détails et l'état de l'exécution sur le tableau de bord du pipeline.

Vous pouvez identifier les éventuels échecs et leur raison. Vous pouvez également afficher les tendances qui se dégagent en matière de durée, d'aboutissement et d'échec d'exécution de pipeline.



### Résultats

Félicitations ! Vous avez intégré vRealize Automation Code Stream à Jenkins en ajoutant un point de terminaison, en créant un pipeline et en configurant une tâche Jenkins qui génère votre code.

## Exemple : Exemple de fichier YAML pour une tâche de génération Jenkins

Pour le type de tâche de génération Jenkins utilisé dans cet exemple, le fichier YAML ressemble au code suivant, accompagné de notifications activées :

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

### Étape suivante

Consultez les autres sections pour plus d'informations. Reportez-vous à la section [Chapitre 6 Connexion de vRealize Automation Code Stream à des points de terminaison](#).

## Comment intégrer vRealize Automation Code Stream à Git

vRealize Automation Code Stream offre un moyen de déclencher un pipeline si une modification de code se produit dans votre référentiel GitHub, GitLab ou Bitbucket. Le déclencheur Git utilise un point de terminaison Git sur la branche du référentiel que vous souhaitez surveiller. vRealize Automation Code Stream se connecte au point de terminaison Git via un webhook.

Pour définir un point de terminaison Git dans vRealize Automation Code Stream, vous sélectionnez un projet et entrez la branche du référentiel Git dans lequel se trouve le point de terminaison. Le projet regroupe le pipeline avec le point de terminaison et autres objets associés. Lorsque vous choisissez le projet dans la définition de votre webhook, vous sélectionnez le point de terminaison et pipeline à déclencher.

---

**Note** Si vous définissez un Webhook avec votre point de terminaison et modifiez ultérieurement le point de terminaison, vous ne pouvez pas modifier les détails du point de terminaison dans le Webhook. Pour modifier les détails du point de terminaison, vous devez supprimer et redéfinir le Webhook avec le point de terminaison. Reportez-vous à la section [Utilisation du déclencheur Git dans vRealize Automation Code Stream pour exécuter un pipeline](#).

---

Vous pouvez créer plusieurs Webhooks pour différentes branches en utilisant le même point de terminaison Git et en fournissant des valeurs différentes pour le nom de la branche sur la page de configuration du Webhook. Pour créer un autre Webhook pour une autre branche dans le même référentiel Git, vous n'avez pas besoin de cloner le point de terminaison Git plusieurs fois pour plusieurs branches. Vous fournissez plutôt le nom de la branche dans le Webhook, ce qui vous permet de réutiliser le point de terminaison Git. Si la branche dans le Webhook Git est la même que celle du point de terminaison, vous n'avez pas besoin de fournir un nom de branche dans la page Git du Webhook.

### Conditions préalables

- Vérifiez que vous pouvez accéder au référentiel GitHub, GitLab ou Bitbucket auquel vous prévoyez de vous connecter.
- Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).

### Procédure

- 1 Définissez un point de terminaison Git.
  - a Cliquez sur **Points de terminaison > Nouveau point de terminaison**.
  - b Sélectionnez un projet et pour le type de point de terminaison, sélectionnez **Git**. Ensuite, entrez un nom et une description.

- c Si ce point de terminaison est un composant stratégique de votre infrastructure, activez **Marquer comme restreint**.

Lorsque vous utilisez un point de terminaison limité dans un pipeline, un administrateur peut exécuter le pipeline et doit approuver son exécution. Si un point de terminaison ou une variable est marqué comme étant limité et qu'un utilisateur non administratif déclenche le pipeline, celui-ci s'interrompt à cette tâche et attend qu'un administrateur le relance.

Un administrateur de projet peut démarrer un pipeline qui inclut des points de terminaison ou des variables limités si ces ressources se trouvent dans le projet dans lequel l'utilisateur est un administrateur de projet.

Lorsqu'un utilisateur qui n'est pas un administrateur tente d'exécuter un pipeline qui inclut une ressource restreinte, le pipeline s'arrête sur la tâche qui utilise la ressource restreinte. Ensuite, un administrateur doit reprendre l'exécution du pipeline.

Pour plus d'informations sur les ressources limitées et les rôles personnalisés qui incluent l'autorisation **Gérer les pipelines limités**, consultez :

- [Gestion de l'accès et des approbations utilisateur dans vRealize Automation Code Stream](#)
- [Chapitre 2 Configuration de vRealize Automation Code Stream pour modéliser le processus de publication](#)

- d Sélectionnez l'un des types de serveur Git pris en charge.
- e Entrez l'URL du référentiel avec la passerelle API pour le serveur dans le chemin d'accès. Par exemple, entrez **`https://api.github.com/vmware-example/repo-example`**.

- f Saisissez la branche dans le référentiel où se trouve le point de terminaison.
- g Sélectionnez le type d'authentification et entrez le nom d'utilisateur pour GitHub, GitLab ou BitBucket. Entrez ensuite le mot de passe, le jeton privé ou la clé privée associés au nom d'utilisateur.

- Mot de passe. Votre mot de passe assure la fin de l'accès au référentiel. Vous pouvez également créer une variable pour le mot de passe.

Utilisez des variables secrètes pour masquer et chiffrer des informations sensibles. Utilisez une variable limitée pour les chaînes, les mots de passe et les URL qui doivent être masqués et chiffrés, ainsi que pour en restreindre l'utilisation dans les exécutions. Par exemple, utilisez une variable secrète pour un mot de passe ou une URL. Vous pouvez utiliser des variables secrètes et restreintes pour n'importe quel type de tâche dans votre pipeline.

- Jeton privé. Ce jeton est spécifique de Git et donne accès à une action spécifique. Reportez-vous à la section [https://docs.gitlab.com/ee/user/profile/personal\\_access\\_tokens.html](https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html). Vous pouvez également créer une variable pour le jeton privé.
- Clé privée. Cette clé SSH est une clé privée qui permet d'accéder à un référentiel spécifique. Lorsqu'un événement Git se produit, vRealize Automation Code Stream utilise cette clé pour cloner un référentiel. Reportez-vous à la section <https://help.github.com/articles/reviewing-your-ssh-keys/>.

- 2 Cliquez sur **Valider** et vérifiez que le point de terminaison se connecte à vRealize Automation Code Stream.

S'il ne se connecte pas, corrigez les erreurs, puis cliquez sur **Créer**.

**New endpoint**

Project \* test

Type \* GIT

Name \* DemoApp-Git

Description Git example branch

Mark restricted ☐ non-restricted

Git Server Type \* GitHub

Repo URL ⓘ \* https://api.github.com/vmware-example/repo-example

ACCEPT CERTIFICATE

Branch \* master

Authentication Type \* Password

Username \* ExampleUser

Password \* .....

CREATE VARIABLE

CREATE VALIDATE CANCEL

### Étape suivante

Pour plus d'informations, consultez les autres sections. Reportez-vous à la section [Utilisation du déclencheur Git dans vRealize Automation Code Stream pour exécuter un pipeline](#).

## Intégration de vRealize Automation Code Stream à Gerrit

vRealize Automation Code Stream fournit un mode de déclenchement d'un pipeline si le code fait l'objet d'une révision dans votre projet Gerrit. La définition du déclencheur Gerrit inclut le projet Gerrit et les pipelines à exécuter pour différents types d'événement.

Le déclencheur Gerrit utilise un écouteur Gerrit sur le serveur que vous souhaitez surveiller. Pour définir un point de terminaison Gerrit dans vRealize Automation Code Stream, vous sélectionnez un projet et entrez l'URL du serveur Gerrit. Spécifiez ensuite le point de terminaison lorsque vous créez un écouteur Gerrit sur ce serveur.

## Conditions préalables

- Vérifiez que vous pouvez accéder au serveur Gerrit auquel vous prévoyez de vous connecter.
- Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).

## Procédure

- 1 Définissez un point de terminaison Gerrit.
  - a Cliquez sur **Configurer > Points de terminaison** et cliquez sur **Nouveau point de terminaison**.
  - b Sélectionnez un projet et sélectionnez Gerrit pour le type de point de terminaison. Ensuite, entrez un nom et une description.
  - c Si ce point de terminaison est un composant stratégique de votre infrastructure, activez **Marquer comme restreint**.
  - d Entrez l'URL du serveur Gerrit.

Vous pouvez fournir un numéro de port avec l'URL ou laisser le champ vide pour utiliser le port par défaut.
  - e Indiquez un nom d'utilisateur et un mot de passe pour le serveur Gerrit.

Si vous souhaitez chiffrer le mot de passe, cliquez sur **Créer une variable** et sélectionnez le type :

    - Secret. Le mot de passe est résolu lors de l'exécution par un utilisateur avec n'importe quel rôle.
    - Restreint. Le mot de passe est résolu lors de l'exécution par un utilisateur disposant du rôle d'administrateur.

Pour la valeur, entrez le mot de passe que vous souhaitez sécuriser, tel que le mot de passe d'un serveur Jenkins.



- f Pour la clé privée, entrez la clé SSH utilisée pour accéder en toute sécurité au serveur Gerrit.

Cette clé est la clé privée RSA située dans le répertoire `.ssh`.

- g (Facultatif) Si une phrase secrète est associée à la clé privée, entrez la phrase secrète.

Si vous souhaitez chiffrer la phrase secrète, cliquez sur **Créer une variable** et sélectionnez le type :

- Secret. La phrase secrète est résolue lors de l'exécution par un utilisateur avec n'importe quel rôle.
- Restreint. La phrase secrète est résolue lors de l'exécution par un utilisateur disposant du rôle d'administrateur.

Pour la valeur, entrez la phrase secrète que vous souhaitez sécuriser, telle que la phrase secrète d'un serveur SSH.

- 2 Cliquez sur **Valider** et vérifiez que le point de terminaison Gerrit dans vRealize Automation Code Stream se connecte au serveur Gerrit.

S'il ne se connecte pas, corrigez les erreurs, puis tentez de le valider à nouveau.

- 3 Cliquez sur **Créer**.

### Étape suivante

Pour plus d'informations, consultez les autres sections. Reportez-vous à la section [Utilisation du déclencheur Gerrit dans vRealize Automation Code Stream pour exécuter un pipeline](#).

## Intégration de vRealize Automation Code Stream à vRealize Orchestrator

vRealize Automation Code Stream peut s'intégrer à vRealize Orchestrator (vRO) pour étendre sa capacité en exécutant des workflows vRO. vRealize Orchestrator inclut de nombreux workflows

prédéfinis qui peuvent s'intégrer à des outils tiers. Ces workflows permettent d'automatiser et de gérer vos processus DevOps, d'automatiser des opérations en bloc, etc.

Par exemple, vous pouvez utiliser un workflow dans une tâche `vRO` de votre pipeline pour activer un utilisateur, supprimer un utilisateur, déplacer des machines virtuelles, procéder à une intégration à des structures de test pour tester votre code au fur et à mesure de l'exécution du pipeline, et bien plus encore. Vous pouvez parcourir des exemples de code pour les workflows vRealize Orchestrator dans [code.vmware.com](https://code.vmware.com).

Avec un workflow vRealize Orchestrator, votre pipeline peut exécuter une action au fur et à mesure qu'il génère, teste et déploie votre application. Vous pouvez inclure des workflows prédéfinis dans votre pipeline, ou créer et utiliser des workflows personnalisés. Chaque workflow inclut des entrées, des tâches et des sorties.

Pour exécuter un workflow `vRO` dans votre pipeline, le workflow doit figurer dans la liste des workflows disponibles dans la tâche `vRO` que vous incluez dans votre pipeline.

Pour que le workflow ne puisse figurer dans la tâche `vRO` de votre pipeline, un administrateur doit effectuer les étapes suivantes dans vRealize Orchestrator :

- 1 Appliquez la balise `CODESTREAM` au workflow `vRO`.
- 2 Marquez le workflow `vRO` comme étant global.

#### Conditions préalables

- Vérifiez que vous pouvez accéder en tant qu'administrateur à une instance sur site de vRealize Orchestrator. Pour obtenir de l'aide, rapprochez-vous de votre propre administrateur et consultez la [Documentation de vRealize Orchestrator](#).
- Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).
- Dans vRealize Automation Code Stream, créez un pipeline et ajoutez une étape.

#### Procédure

- 1 En tant qu'administrateur, préparez un workflow vRealize Orchestrator pour l'exécution de votre pipeline.
  - a Dans vRealize Orchestrator, recherchez le workflow que vous devez utiliser dans votre pipeline, tel qu'un workflow pour activer un utilisateur.  
Si vous avez besoin d'un workflow qui n'existe pas, vous pouvez le créer.
  - b Dans la barre de recherche, entrez **Workflow de balise** pour trouver le workflow nommé `Workflow de balise`.
  - c Sur la fiche nommée `Workflow de balise`, cliquez sur **Exécuter** afin d'afficher la zone de configuration.

- d Dans la zone de texte `Workflow` balisé, entrez le nom du workflow à utiliser dans votre pipeline vRealize Automation Code Stream, puis sélectionnez-le dans la liste.
- e Dans les zones de texte `Balise` et `Valeur`, entrez `CODESTREAM` en lettres majuscules.
- f Cochez la case nommée **Balise globale**.
- g Cliquez sur **Exécuter** afin d'associer la balise nommée `CODESTREAM` au workflow que vous devez sélectionner dans votre pipeline de vRealize Automation Code Stream.
- h Dans le volet de navigation, cliquez sur **Workflows** et vérifiez que la balise nommée `CODESTREAM` apparaît sur la fiche du workflow que votre pipeline va exécuter.

Après la connexion à vRealize Automation Code Stream et l'ajout d'une tâche `vRO` à votre pipeline, le workflow balisé s'affiche dans la liste des workflows.

- 2 Dans vRealize Automation Code Stream, créez un point de terminaison pour votre instance de vRealize Orchestrator.

- a Cliquez sur **Points de terminaison > Nouveau point de terminaison**.
- b Sélectionnez un projet.
- c Entrez un nom pertinent.
- d Entrez l'URL du point de terminaison vRealize Orchestrator.

Utilisez ce format : `https://cava-n-01-234.eng.vmware.com:8281`

N'utilisez pas ce format : `https://cava-n-01-234.eng.vmware.com:8281/vco/api`

- e Cliquez sur **Accepter le certificat** si l'URL que vous avez entrée a besoin d'un certificat.
- f Entrez le nom d'utilisateur et le mot de passe correspondant au serveur vRealize Orchestrator.

- 3 Préparez votre pipeline pour exécuter la tâche `vRO`.

- a Ajoutez une tâche `vRO` à votre étape de pipeline.
- b Entrez un nom pertinent.
- c Dans la zone Propriétés du workflow, sélectionnez le point de terminaison vRealize Orchestrator.
- d Sélectionnez le workflow que vous avez marqué `CODESTREAM` dans vRealize Orchestrator.

Si vous sélectionnez un workflow personnalisé que vous avez créé, vous devrez peut-être entrer les valeurs des paramètres d'entrée.

- e Pour **exécuter la tâche**, cliquez sur **Sur condition**.

- f Entrez les conditions qui s'appliquent lors de l'exécution du pipeline.

Quand exécuter le pipeline...	Conditions à sélectionner...
<b>Sur condition</b>	<p>Exécute la tâche de pipeline uniquement si la condition définie est évaluée comme étant true. Si la condition est false, la tâche est ignorée.</p> <p>La tâche vRO vous permet d'inclure une expression booléenne qui utilise les opérandes et les opérateurs suivants.</p> <ul style="list-style-type: none"> <li>■ Variables de pipeline comme <code>\${pipeline.variableName}</code>. Utilisez uniquement des accolades lors de la saisie de variables.</li> <li>■ Variables de sortie de tâche comme <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code>.</li> <li>■ Variables de liaison de pipeline par défaut comme <code>{releasePipelineName}</code>.</li> <li>■ Valeurs booléennes insensibles à la casse comme <code>true</code>, <code>false</code>, <code>'true'</code>, <code>'false'</code>.</li> <li>■ Valeurs entières ou décimales sans guillemets.</li> <li>■ Valeurs de chaîne utilisées avec des guillemets simples ou doubles, comme <code>"test"</code> ou <code>'test'</code>.</li> <li>■ Types de chaînes et de valeurs numériques, comme <code>==</code> Equals et <code>!=</code> Not Equals.</li> <li>■ Opérateurs relationnels comme <code>&gt;</code>, <code>&gt;=</code>, <code>&lt;</code> et <code>&lt;=</code>.</li> <li>■ Logique booléenne comme <code>&amp;&amp;</code> et <code>  </code>.</li> <li>■ Opérateurs arithmétiques comme <code>+</code>, <code>-</code>, <code>*</code> et <code>/</code>.</li> <li>■ Expressions imbriquées à l'aide de parenthèses.</li> <li>■ Les chaînes qui incluent la valeur littérale <code>ABCD</code> sont évaluées comme étant false et la tâche est ignorée.</li> <li>■ Les opérateurs unaires ne sont pas pris en charge.</li> </ul> <p>Un exemple de condition peut être <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>
<b>Toujours</b>	<p>Si vous sélectionnez <b>Toujours</b>, le pipeline exécute la tâche sans conditions.</p>

- g Entrez un message d'accueil.
- h Cliquez sur **Valider la tâche** et corrigez les erreurs qui se sont éventuellement produites.

- 4 Enregistrez, activez et exécutez votre pipeline.
- 5 Une fois le pipeline exécuté, examinez les résultats.
  - a Cliquez sur **Exécutions**.
  - b Cliquez sur le pipeline.
  - c Cliquez sur la tâche.
  - d Examinez les résultats, la valeur d'entrée et les propriétés.

Vous pouvez identifier l'ID d'exécution du workflow, la personne qui a répondu à la tâche, le moment auquel elle a répondu et les commentaires qu'elle a inclus.

### Résultats

Félicitations ! Vous avez marqué un workflow vRealize Orchestrator à utiliser dans vRealize Automation Code Stream et ajouté une tâche vRO dans votre pipeline vRealize Automation Code Stream afin qu'il exécute un workflow qui automatise une action dans votre environnement DevOps.

## Exemple : Format de sortie de tâche vRO

Le format de sortie d'une tâche vRO ressemble à cet exemple.

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

### Étape suivante

Continuez à inclure des tâches de workflow vRO dans vos pipelines afin de pouvoir automatiser des tâches dans vos environnements de développement, de test et de production.

# Déclenchement des pipelines dans vRealize Automation Code Stream

## 7

Vous pouvez demander à vRealize Automation Code Stream de déclencher un pipeline lorsque certains événements se produisent.

Par exemple, vous pouvez :

- Utiliser le déclencheur Docker pour exécuter un pipeline lorsqu'un nouvel artefact est créé ou mis à jour.
- Utiliser le déclencheur Git pour déclencher un pipeline lorsque les développeurs mettent à jour le code.
- Utiliser le déclencheur Gerrit pour déclencher un pipeline lorsque les développeurs révisent le code.
- Utiliser la commande `curl` pour que Jenkins déclenche le pipeline lorsqu'une build est terminée.

Ce chapitre contient les rubriques suivantes :

- [Utilisation du déclencheur Docker dans vRealize Automation Code Stream pour exécuter un pipeline de livraison continue](#)
- [Utilisation du déclencheur Git dans vRealize Automation Code Stream pour exécuter un pipeline](#)
- [Utilisation du déclencheur Gerrit dans vRealize Automation Code Stream pour exécuter un pipeline](#)

## Utilisation du déclencheur Docker dans vRealize Automation Code Stream pour exécuter un pipeline de livraison continue

En tant qu'administrateur ou développeur de vRealize Automation Code Stream, vous pouvez utiliser le déclencheur Docker dans vRealize Automation Code Stream. Le déclencheur Docker exécute un pipeline de livraison continue (CD) autonome lors de la création ou de la mise à jour d'un artefact de build. Le déclencheur Docker exécute votre pipeline de livraison continue, qui transfère le nouvel artefact ou l'artefact mis à jour en tant qu'image de conteneur vers un référentiel Docker Hub. Le pipeline de livraison continue peut s'exécuter dans le cadre de vos builds automatisées.

Par exemple, pour déployer continuellement votre image de conteneur mise à jour dans votre pipeline de livraison continue, utilisez le déclencheur Docker. Lorsque votre image de conteneur est archivée dans le registre Docker, le Webhook du Docker Hub informe vRealize Automation Code Stream que l'image a changé. Cette notification déclenche l'exécution du pipeline de livraison continue avec l'image de conteneur mise à jour et télécharge l'image vers le référentiel Docker Hub.

Pour utiliser le déclencheur Docker, vous effectuez plusieurs étapes dans vRealize Automation Code Stream.

**Tableau 7-1. Utilisation du déclencheur Docker**

Actions...	Plus d'informations sur cette action...
Créez un point de terminaison de registre Docker.	<p>Pour que vRealize Automation Code Stream déclenche votre pipeline, vous devez disposer d'un point de terminaison de registre Docker. Si le point de terminaison n'existe pas, vous pouvez sélectionner une option qui le crée lorsque vous ajoutez le Webhook pour le déclencheur Docker.</p> <p>Le point de terminaison du registre Docker inclut l'URL vers le référentiel Docker Hub.</p>
Ajoutez des paramètres d'entrée au pipeline qui injecte automatiquement des paramètres Docker lors de son exécution.	<p>Vous pouvez injecter automatiquement des paramètres Docker dans le pipeline. Ces paramètres peuvent inclure le nom du propriétaire de l'événement Docker, l'image, le nom du référentiel, l'espace de noms du référentiel et la balise.</p> <p>Dans votre pipeline de livraison continue, vous incluez les paramètres d'entrée que le Webhook Docker transmet au pipeline avant que le pipeline ne se déclenche.</p>
Créez un Webhook Docker.	<p>Lorsque vous créez le Webhook Docker dans vRealize Automation Code Stream, cette action déclenche également la création d'un Webhook correspondant dans Docker Hub. Le Webhook Docker dans vRealize Automation Code Stream se connecte à Docker Hub via l'URL que vous incluez dans le Webhook.</p> <p>Les Webhooks communiquent entre eux et déclenchent le pipeline lorsqu'un artefact est créé ou mis à jour dans Docker Hub.</p> <p>Si vous mettez à jour ou supprimez le Webhook Docker dans vRealize Automation Code Stream, le Webhook de Docker Hub est également mis à jour ou supprimé.</p>
Ajoutez et configurez une tâche Kubernetes dans votre pipeline.	<p>Lorsqu'un artefact est créé ou mis à jour dans le référentiel Docker Hub, le pipeline se déclenche. Ensuite, il déploie l'artefact via le pipeline sur l'hôte Docker de votre cluster Kubernetes.</p>
Incluez une définition YAML locale dans la tâche.	<p>La définition YAML que vous appliquez à la tâche de déploiement inclut l'image de conteneur Docker et toute clé secrète requise pour extraire l'image du référentiel à des fins de déploiement.</p>

Lorsqu'un artefact est créé ou mis à jour dans le référentiel Docker Hub, le Webhook de Docker Hub en informe le Webhook dans vRealize Automation Code Stream, qui déclenche le pipeline. Les actions suivantes se produisent :

- 1 Docker Hub envoie une demande POST à l'URL du Webhook.
- 2 vRealize Automation Code Stream exécute le déclencheur Docker.
- 3 Le déclencheur Docker démarre votre pipeline de livraison continue.
- 4 Le pipeline de livraison continue transfère l'artefact vers le référentiel Docker Hub.
- 5 vRealize Automation Code Stream déclenche son Webhook Docker, qui exécute un pipeline de livraison continue qui déploie l'artefact sur votre hôte Docker.

Dans cet exemple, vous créez un point de terminaison Docker et un Webhook Docker dans vRealize Automation Code Stream qui déploie votre application sur votre cluster Kubernetes de développement. Les étapes incluent l'exemple de code de la charge utile que Docker publie sur l'URL du Webhook, le code d'API qu'il utilise et le code d'authentification avec le jeton sécurisé.

#### Conditions préalables

- Vérifiez qu'un pipeline de livraison continue (CD) existe dans votre instance de vRealize Automation Code Stream. Vérifiez également qu'il inclut une ou plusieurs tâches Kubernetes qui déploient votre application. Reportez-vous à la section [Chapitre 4 Planification de la génération, de l'intégration et de la livraison de votre code en mode natif dans vRealize Automation Code Stream](#) .
- Vérifiez que vous pouvez accéder à un cluster Kubernetes existant sur lequel votre pipeline de livraison continue peut déployer votre application à des fins de développement.
- Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).

#### Procédure

- 1 Créez un point de terminaison de registre Docker.
  - a Cliquez sur **Points de terminaison**.
  - b Cliquez sur **Nouveau point de terminaison**.
  - c Entrez un nom pertinent.
  - d Sélectionnez le type de serveur **Docker Hub**.



- e Entrez l'URL du référentiel Docker Hub.
- f Entrez le nom et le mot de passe utilisés pour accéder au référentiel.

- 2 Dans votre pipeline de livraison continue, définissez les propriétés d'entrée pour injecter automatiquement les paramètres Docker lors de l'exécution du pipeline.

- 3 Créez un Webhook Docker.
  - a Cliquez sur **Déclencheurs > Docker**.
  - b Cliquez sur **Nouveau Webhook pour Docker**.
  - c Sélectionnez un projet.
  - d Entrez un nom pertinent.
  - e Sélectionnez votre point de terminaison de registre Docker.  
Si le point de terminaison n'existe pas encore, cliquez sur **Créer un point de terminaison** et créez-le.
  - f Sélectionnez le pipeline dans lequel des paramètres Docker sont injectés et que le Webhook doit déclencher. Reportez-vous à la section [Étape 2](#).

Si le pipeline a été configuré avec des paramètres d'entrée personnalisés supplémentaires, la liste des paramètres d'entrée affiche les paramètres et les valeurs. Vous pouvez saisir des valeurs pour les paramètres d'entrée qui seront transmis au pipeline avec l'événement déclencheur. Sinon, vous pouvez laisser les valeurs vides ou utiliser les valeurs par défaut, si elles sont définies.

Pour plus d'informations sur les paramètres de l'onglet Entrée, reportez-vous à la section [Planification d'une build native CI/CD dans vRealize Automation Code Stream](#) avant d'ajouter des tâches manuellement.

- g Entrez le jeton de l'API.

Le jeton d'API CSP vous authentifie pour les connexions d'API externes avec vRealize Automation Code Stream. Pour obtenir le jeton d'API :

- 1 Cliquez sur **Générer le jeton**.
- 2 Entrez l'adresse e-mail associée à votre nom d'utilisateur et votre mot de passe, puis cliquez sur **Générer**.

Le jeton généré est valide pendant six mois. Il est également appelé jeton d'actualisation.

- Pour conserver le jeton sous forme de variable pour une utilisation ultérieure, cliquez sur **Créer une variable**, entrez un nom pour la variable et cliquez sur **Enregistrer**.
- Pour conserver le jeton sous forme de valeur de texte pour une utilisation ultérieure, cliquez sur **Copier** et collez le jeton dans un fichier texte pour l'enregistrer localement.

Vous pouvez choisir à la fois de créer une variable et de stocker le jeton dans un fichier texte pour une utilisation ultérieure.

- 3 Cliquez sur **Fermer**.
- h Entrez l'image de la build.
- i Entrez une balise.

The screenshot shows the 'Webhooks pour Docker' configuration page. The 'URL du Webhook' field contains a long alphanumeric string. The 'Projet' field is set to 'test'. The 'Nom' field is 'sm-1-Docker-WH'. The 'Description' field contains 'Docker webhook trigger for sm-1'. The 'Registre Docker' field is 'Docker-Register-Endpoint'. The 'Pipeline' field is 'sm-1'. The 'Jeton d'API' field is empty, with a red 'x' icon and a 'GÉNÉRER LE JETON' button next to it. The 'Image' field is 'Image' and the 'Balise' field is 'Balises'. At the bottom, there are 'ENREGISTRER' and 'ANNULER' buttons.

- j Cliquez sur **Enregistrer**.

La fiche du Webhook s'affiche avec le Webhook Docker activé. Si vous souhaitez effectuer un transfert factice vers le référentiel Docker Hub sans déclencher le Webhook Docker ni exécuter de pipeline, cliquez sur **Désactiver**.

- 4 Dans votre pipeline de livraison continue, configurez votre tâche de déploiement Kubernetes.
  - a Dans les propriétés de la tâche Kubernetes, sélectionnez votre cluster Kubernetes de développement.
  - b Sélectionnez l'action **Créer**.

- c Sélectionnez la **définition locale** correspondant à la source de la charge utile.
- d Sélectionnez ensuite votre fichier YAML local.

Par exemple, Docker Hub peut publier cette définition YAML locale en tant que charge utile sur l'URL du Webhook :

```
{
  "callback_url": "https://registry.hub.docker.com/u/svendowideit/testhook/hook/2141b5bi5i5b02bec211i4eeih0242eg11000a/",
  "push_data": {
    "images": [
      "27d47432a69bca5f2700e4dff7de0388ed65f9d3fb1ec645e2bc24c223dc1cc3",
      "51a9c7c1f8bb2fa19bcd09789a34e63f35abb80044bc10196e304f6634cc582c",
      "...",
    ],
    "pushed_at": 1.417566161e+09,
    "pusher": "trustedbuilder",
    "tag": "latest"
  },
  "repository": {
    "comment_count": 0,
    "date_created": 1.417494799e+09,
    "description": "",
    "dockerfile": "#\n# BUILD\u0009\u0009docker build -t svendowideit/apt-cacher .\n# RUN\u0009\u0009docker run -d -p 3142:3142 -name apt-cacher-run apt-cacher\n#\n# and then you can run containers with:\n#\n\u0009\u0009docker run -t -i -rm -e http_proxy http://192.168.1.2:3142/debian bash\n#\nFROM\u0009\u0009ubuntu\n#\nVOLUME\u0009\u0009[/var/cache/apt-cacher-ng]\nRUN\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n#\nEXPOSE\n\u0009\u000993142\nCMD\u0009\u0009chmod 777 /var/cache/apt-cacher-ng ; /etc/init.d/apt-cacher-ng start ; tail -f /var/log/apt-cacher-ng/*\n",
    "full_description": "Docker Hub based automated build from a GitHub repo",
    "is_official": false,
    "is_private": true,
    "is_trusted": true,
    "name": "testhook",
    "namespace": "svendowideit",
    "owner": "svendowideit",
    "repo_name": "svendowideit/testhook",
    "repo_url": "https://registry.hub.docker.com/u/svendowideit/testhook/",
    "star_count": 0,
    "status": "Active"
  }
}
```

L'API qui crée le Webhook dans Docker Hub utilise

ce formulaire : [https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook\\_pipeline/](https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/)

Le corps du code JSON ressemble à ce qui suit :

```
{
  "name": "demo_webhook",
```

```
"webhooks": [
{
"name": "demo_webhook",
"hook_url": "http://www.google.com"
}
]
}
```

Pour recevoir des événements du serveur Docker Hub, le schéma d'authentification correspondant au Webhook Docker que vous créez dans vRealize Automation Code Stream utilise un mécanisme d'authentification sur liste autorisée avec un jeton de chaîne aléatoire pour le Webhook. Il filtre les événements en fonction du jeton sécurisé, que vous pouvez ajouter à `hook_url`.

vRealize Automation Code Stream peut vérifier toutes les demandes du serveur Docker Hub en utilisant le jeton sécurisé configuré. Par exemple : `hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""`

- 5 Créez un artefact Docker dans votre référentiel Docker Hub. Sinon, mettez à jour un artefact existant.
- 6 Pour vous assurer que le déclencheur est intervenu et afficher l'activité sur le Webhook Docker, cliquez sur **Déclencheurs > Docker > Activité**.

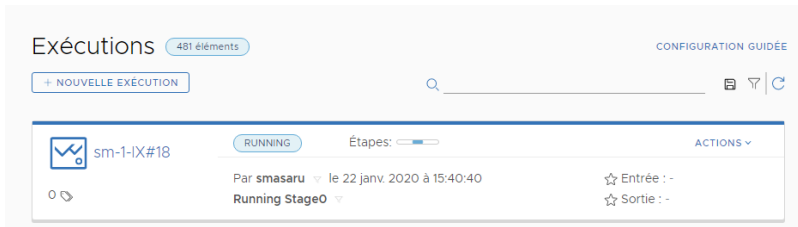
## Docker

GUIDED SETUP

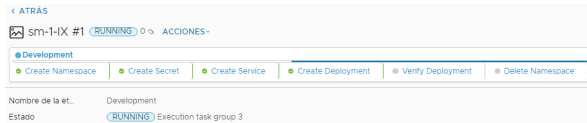
Activity
Webhooks for Docker

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 Cliquez sur **Exécutions** et suivez la progression de votre pipeline au fil de son exécution.



- 8 Cliquez sur l'étape d'exécution et affichez les tâches au cours de l'exécution du pipeline.



## Résultats

Félicitations ! Vous avez configuré le déclencheur Docker afin qu'il exécute continuellement votre pipeline de livraison continue. Votre pipeline peut désormais télécharger des artefacts Docker nouveaux et mis à jour vers le référentiel Docker Hub.

## Étape suivante

Vérifiez que votre nouvel artefact ou votre artefact mis à jour est déployé sur l'hôte Docker dans votre cluster Kubernetes de développement.

# Utilisation du déclencheur Git dans vRealize Automation Code Stream pour exécuter un pipeline

En tant qu'administrateur ou développeur de vRealize Automation Code Stream, vous pouvez utiliser le déclencheur Git pour intégrer vRealize Automation Code Stream au cycle de vie Git. Lorsque vous modifiez le code dans GitHub, GitLab ou Bitbucket Enterprise, l'événement communique avec vRealize Automation Code Stream via un Webhook et déclenche l'exécution d'un pipeline. Le Webhook fonctionne avec les versions d'entreprise sur site de GitLab, GitHub et Bitbucket lorsque vRealize Automation Cloud Assembly et la version d'entreprise sont accessibles sur le même réseau.

Lorsque vous ajoutez le Webhook pour Git dans vRealize Automation Code Stream, cette action crée également un Webhook dans le GitHub, GitLab ou le référentiel Bitbucket. Si vous mettez à jour ou supprimez le Webhook ultérieurement, le Webhook dans GitHub, GitLab ou Bitbucket est également mis à jour ou supprimé.

Votre définition du Webhook doit inclure un point de terminaison Git sur la branche du référentiel que vous souhaitez surveiller. vRealize Automation Code Stream utilise le point de terminaison Git pour créer le Webhook. Si le point de terminaison n'existe pas, vous pouvez le créer lorsque vous ajoutez le Webhook. Dans cet exemple, vous devez disposer d'un point de terminaison Git prédéfini dans GitHub.

Vous pouvez créer plusieurs Webhooks pour différentes branches en utilisant le même point de terminaison Git et en fournissant des valeurs différentes pour le nom de la branche sur la page de configuration du Webhook. Pour créer un autre Webhook pour une autre branche dans le même référentiel Git, vous n'avez pas besoin de cloner le point de terminaison Git plusieurs fois pour plusieurs branches. Vous fournissez plutôt le nom de la branche dans le Webhook, ce qui vous permet de réutiliser le point de terminaison Git. Si la branche dans le Webhook Git est la même que celle du point de terminaison, vous n'avez pas besoin de fournir un nom de branche dans la page Git du Webhook.

Cet exemple vous présente comment utiliser le déclencheur Git avec un référentiel GitHub, mais les conditions préalables incluent des préparations requises si un autre type de serveur Git est utilisé.

#### Conditions préalables

- Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).
- Vérifiez que vous disposez d'un point de terminaison Git sur la branche GitHub que vous souhaitez surveiller. Reportez-vous à la section [Comment intégrer vRealize Automation Code Stream à Git](#).
- Vérifiez que vous disposez des droits de création d'un Webhook dans le référentiel Git.
- Si vous configurez un Webhook dans GitLab, modifiez les paramètres réseau par défaut dans GitLab Enterprise pour activer les demandes sortantes et autoriser la création de Webhooks locaux.

---

**Note** Cette modification n'est requise que pour GitLab Enterprise. Ces paramètres ne s'appliquent pas à GitHub ou Bitbucket.

---

- a Connectez-vous à votre instance de GitLab en tant qu'administrateur.
- b Accédez aux paramètres réseau à l'aide d'une URL telle que, `http://{gitlab-server}/admin/application_settings/network`.
- c Développez **Demandes sortantes**, puis cliquez sur :
  - Autorisez les demandes au réseau local à partir de Webhooks et des services Web.
  - Autorisez les demandes au réseau local à partir du hook système.
- Pour les pipelines que vous souhaitez déclencher, vérifiez que vous avez défini les propriétés d'entrée pour injecter des paramètres Git lorsque le pipeline s'exécute.

**Build and Deploy** Activé

Espace de travail **Entrée** Modèle Sortie

Paramètres d'entrée ⓘ

Paramètres d'injection ... ☐ Gerrit ☒ Git ☐ Docker ☐ Aucun

**AJOUTER** **AJOUTER/SUPPRIMER DES PARAMÈTRES INJECTÉS**

Marqué d'une étoile	Nom
☆	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

Pour plus d'informations sur les paramètres d'entrée, reportez-vous à la section [Planification d'une build native CI/CD dans vRealize Automation Code Stream](#) avant d'ajouter des tâches manuellement.

#### Procédure

- 1 Dans vRealize Automation Code Stream, cliquez sur **Déclencheurs > Git**.
- 2 Cliquez sur l'onglet **Webhooks pour Git**, puis cliquez sur **Nouveau Webhook pour Git**.
  - a Sélectionnez un projet.
  - b Entrez un nom et une description significatifs pour le Webhook.
  - c Sélectionnez un point de terminaison Git configuré pour la branche que vous souhaitez surveiller.

Lorsque vous créez votre Webhook, sa définition inclut les détails actuels du point de terminaison.

- Si vous modifiez ultérieurement le type de Git, le type de serveur Git ou l'URL du référentiel Git dans le point de terminaison, le Webhook ne pourra plus déclencher de pipeline, car il essaiera d'accéder au référentiel Git à l'aide des détails du point de terminaison d'origine. Vous devez supprimer le Webhook et le recréer avec le point de terminaison.
- Si vous modifiez ultérieurement le type d'authentification, le nom d'utilisateur ou le mot de passe dans le point de terminaison, le Webhook continuera à fonctionner.

Reportez-vous à la section [Comment intégrer vRealize Automation Code Stream à Git](#).

- d (Facultatif) Entrez la branche que vous souhaitez faire surveiller par le Webhook.

Si elle n'est pas spécifiée, le Webhook surveille la branche configurée pour le point de terminaison Git.



- e (Facultatif) Générez un jeton secret pour le Webhook.

S'il est utilisé, vRealize Automation Code Stream génère un jeton de chaîne aléatoire pour le Webhook. Ensuite, lorsque le Webhook reçoit les données d'événement Git, il les envoie avec le jeton secret. vRealize Automation Code Stream utilise les informations pour déterminer si les appels proviennent de la source attendue, telle que l'instance, le référentiel et la branche de GitHub configurés. Le jeton secret fournit une couche supplémentaire de sécurité qui est utilisée pour vérifier que les données d'événement Git proviennent de la source appropriée.

- f (Facultatif) Précisez des inclusions ou des exclusions de fichier comme conditions pour le déclencheur.
- L'indication d'inclusions de fichiers conditionne le déclenchement des pipelines lorsqu'un des fichiers d'une validation correspond aux fichiers spécifiés dans les chemins d'inclusion ou regex. Avec un regex spécifié, vRealize Automation Code Stream déclenche uniquement les pipelines dont les noms de fichier dans l'ensemble de modifications correspondent à l'expression fournie. Le filtre regex est utile lors de la configuration d'un déclencheur pour plusieurs pipelines sur un référentiel unique.
  - L'indication d'exclusions de fichiers conditionne l'absence de déclenchement des pipelines lorsque tous les fichiers d'une validation correspondent aux fichiers spécifiés dans les chemins d'exclusion ou regex.
  - Lorsque cette option est activée, l'exclusion par priorité garantit que les pipelines ne sont pas déclenchés, même si l'un des fichiers d'une validation correspond aux fichiers spécifiés dans les chemins d'exclusion ou regex. Le paramètre par défaut est désactivé.

Si les conditions d'inclusion et d'exclusion sont satisfaites, les pipelines ne sont pas déclenchés.

Dans l'exemple suivant, les inclusions et les exclusions de fichiers sont des conditions de déclenchement.

The screenshot shows a configuration window titled 'Fichier' with a sub-header 'Inclusions'. Below this, there are two sections: 'Inclusions' and 'Exclusions'. Each section has a dropdown menu for the rule type (PLAIN or REGEX) and a text input for the file path or regex pattern. To the right of each input is a button with a plus or minus sign. At the bottom, there is a toggle switch for 'Exclusion par priorité'.

Type	Rule	Path/Pattern	Action
Inclusions	PLAIN	runtime/src/main/a.java	+
Exclusions	REGEX	([a-z A-Z]+[/][a-z A-Z])+	-
Exclusions	PLAIN	runtime/pom.xml	-
Exclusions	PLAIN	runtime/demo.yaml	- +

Exclusion par priorité: ☐

- Pour les inclusions de fichiers, une validation comportant une modification apportée à `runtime/src/main/a.java` ou un fichier Java déclenche les pipelines configurés dans la configuration des événements.
- En ce qui concerne les exclusions de fichiers, une validation comportant des modifications dans les deux fichiers uniquement ne déclenche pas les pipelines configurés dans les configurations d'événement.

- g Pour l'événement Git, sélectionnez une demande d'**opération push** ou d'**extraction**.
- h Entrez le jeton de l'API.

Le jeton d'API CSP vous authentifie pour les connexions d'API externes avec vRealize Automation Code Stream. Pour obtenir le jeton d'API :

- 1 Cliquez sur **Générer le jeton**.
- 2 Entrez l'adresse e-mail associée à votre nom d'utilisateur et votre mot de passe, puis cliquez sur **Générer**.

Le jeton généré est valide pendant six mois. Il est également appelé jeton d'actualisation.

- Pour conserver le jeton sous forme de variable pour une utilisation ultérieure, cliquez sur **Créer une variable**, entrez un nom pour la variable et cliquez sur **Enregistrer**.
- Pour conserver le jeton sous forme de valeur de texte pour une utilisation ultérieure, cliquez sur **Copier** et collez le jeton dans un fichier texte pour l'enregistrer localement.

Vous pouvez choisir à la fois de créer une variable et de stocker le jeton dans un fichier texte pour une utilisation ultérieure.

- 3 Cliquez sur **Fermer**.
- i Sélectionnez le pipeline que le Webhook doit déclencher.

Si le pipeline a été configuré avec des paramètres d'entrée personnalisés supplémentaires, la liste des paramètres d'entrée affiche les paramètres et les valeurs. Vous pouvez saisir des valeurs pour les paramètres d'entrée qui seront transmis au pipeline avec l'événement déclencheur. Sinon, vous pouvez laisser les valeurs vides ou utiliser les valeurs par défaut, si elles sont définies.

Pour plus d'informations sur les paramètres d'entrée injectés automatiquement pour les déclencheurs Git, reportez-vous aux [conditions préalables](#).

- j Cliquez sur **Créer**.

Le Webhook s'affiche sous la forme d'une nouvelle fiche.

- 3 Cliquez sur la fiche du Webhook.

Lorsque le formulaire de données du Webhook s'affiche à nouveau, vous constatez qu'une URL Webhook a été ajoutée en haut du formulaire. Le Webhook Git se connecte au référentiel GitHub via l'URL Webhook.

## Git

Activity **Webhooks for Git**

Webhook URL ⓘ `https://ca[REDACTED]om/codestream/api/git-webhook-listeners/963b2287-527f-4e9b`

Project `test`

Name \* `test-webhook`

Description

Endpoint `DemoApp-Git`

Branch ⓘ `master`

Secret token ⓘ \* `GYH0cBWZx4dUn47Y/KA8H/BOKts=` GENERATE

File ⓘ

	--Select--	Value	
Inclusions	--Select--	Value	+
Exclusions	--Select--	Value	+


Prioritize Exclusion ☐

Trigger

For Git ☒ PUSH ☐ PULL REQUEST

API token \* 

.....

 CREATE VARIABLE GENERATE TOKEN

Pipeline \* `CICD-2` ⓘ

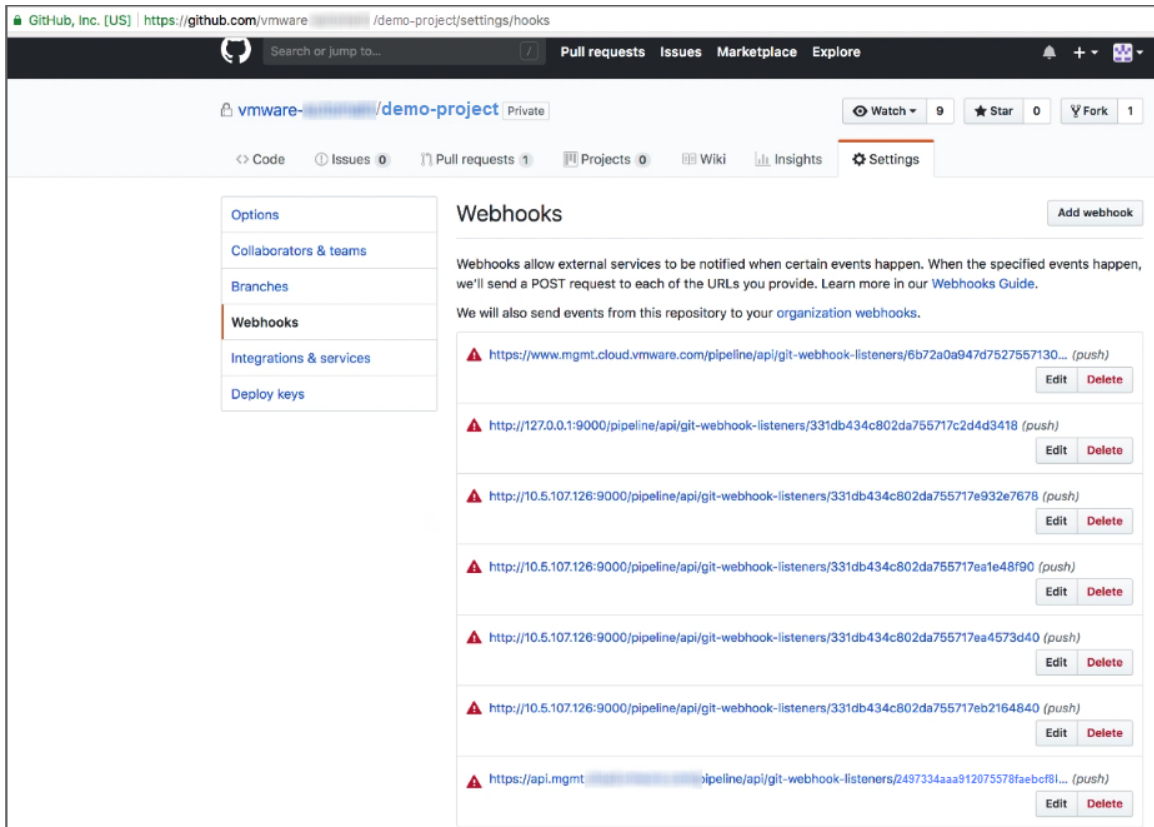
Comments

Execution trigger delay ⓘ `1` ⓘ

SAVE CANCEL

- 4 Dans une nouvelle fenêtre de navigateur, ouvrez le référentiel GitHub connecté via le Webhook.
  - a Pour voir le Webhook que vous avez ajouté dans vRealize Automation Code Stream, cliquez sur l'onglet **Paramètres** et sélectionnez **Webhooks**.

Au bas de la liste des Webhooks, la même URL Webhook est indiquée.



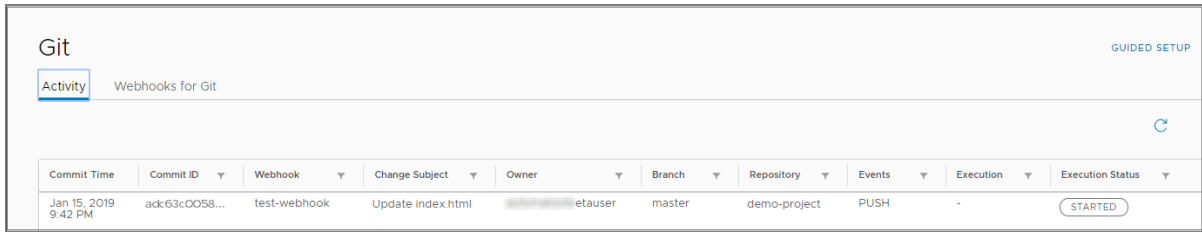
- b Pour apporter une modification au code, cliquez sur l'onglet **Code** et sélectionnez un fichier dans lequel effectuer le changement sur la branche à modifier. Validez la modification.
  - c Pour vérifier que l'URL Webhook fonctionne, cliquez sur l'onglet **Paramètres** et sélectionnez à nouveau **Webhooks**.

Au bas de la liste des Webhooks, une coche verte s'affiche en regard de l'URL Webhook.



- 5 Revenez à vRealize Automation Code Stream pour afficher l'activité sur le Webhook Git. Cliquez sur **Déclencheurs > Git > Activité**.

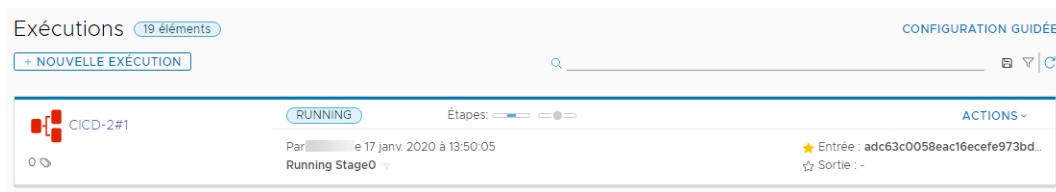
Sous État de l'exécution, vérifiez que l'exécution du pipeline a démarré.



Commit Time	Commit ID	Webhook	Change Subject	Owner	Branch	Repository	Events	Execution	Execution Status
Jan 15, 2019 9:42 PM	ack63c0058...	test-webhook	Update index.html	etauser	master	demo-project	PUSH	-	STARTED

- 6 Cliquez sur **Exécutions** pour suivre l'exécution de votre pipeline.

Vous pouvez appuyer sur le bouton Actualiser pour surveiller l'exécution du pipeline.



Exécution	Statut	Étapes	Par	Date	Entrée	Sortie
CICD-2#1	RUNNING	0/3	Par	17 janv. 2020 à 13:50:05	Entrée : adc63c0058eac16ecfe973bd...	Sortie : -

## Résultats

Félicitations ! Vous avez correctement utilisé un déclencheur Git pour exécuter un pipeline.

# Utilisation du déclencheur Gerrit dans vRealize Automation Code Stream pour exécuter un pipeline

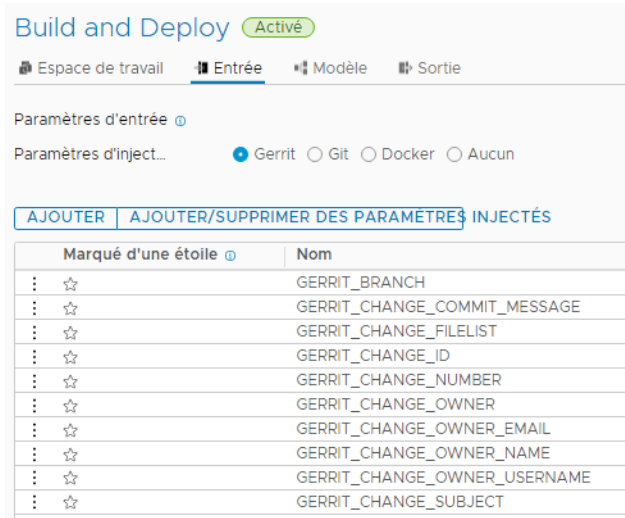
En tant qu'administrateur ou développeur de vRealize Automation Code Stream, vous pouvez utiliser le déclencheur Gerrit pour intégrer vRealize Automation Code Stream au cycle de vie de révision du code Gerrit. L'événement déclenche un pipeline à exécuter lorsque vous créez un jeu de correctifs, publiez des brouillons, fusionnez des modifications de code sur le projet Gerrit ou transmettez directement les modifications sur la branche Git.

Lorsque vous ajoutez le déclencheur pour Gerrit, vous sélectionnez un écouteur Gerrit et un projet Gerrit sur le serveur, et configurez des événements Gerrit. Dans cet exemple, vous configurez tout d'abord un écouteur Gerrit, puis vous utilisez cet écouteur dans un déclencheur Gerrit avec deux événements sur trois pipelines différents.

## Conditions préalables

- Assurez-vous d'être membre d'un projet dans vRealize Automation Code Stream. Si ce n'est pas le cas, demandez à un administrateur de vRealize Automation Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section [Ajout d'un projet dans vRealize Automation Code Stream](#).

- Vérifier que vous disposez d'un point de terminaison Gerrit configuré dans vRealize Automation Code Stream. Reportez-vous à la section [Intégration de vRealize Automation Code Stream à Gerrit](#).
- Pour les pipelines à déclencher, vérifiez que vous avez défini les propriétés d'entrée pour injecter des paramètres Gerrit lorsque le pipeline s'exécute.



Pour plus d'informations sur les paramètres d'entrée, reportez-vous à la section [Planification d'une build native CI/CD dans vRealize Automation Code Stream](#) avant d'ajouter des tâches manuellement.

#### Procédure

- 1 Dans vRealize Automation Code Stream, cliquez sur **Déclencheurs > Gerrit**.
- 2 (Facultatif) Cliquez sur l'onglet **Écouteurs**, puis sur **Nouvel écouteur**.

**Note** Ignorez cette étape si l'écouteur Gerrit que vous prévoyez d'utiliser pour le déclencheur Gerrit est déjà défini.

- a Sélectionnez un projet.
- b Entrez un nom pour l'écouteur Gerrit.
- c Sélectionnez un point de terminaison Gerrit.

## d Entrez le jeton de l'API.

Le jeton d'API CSP vous authentifie pour les connexions d'API externes avec vRealize Automation Code Stream. Pour obtenir le jeton d'API :

- 1 Cliquez sur **Générer le jeton**.
- 2 Entrez l'adresse e-mail associée à votre nom d'utilisateur et votre mot de passe, puis cliquez sur **Générer**.

Le jeton généré est valide pendant six mois. Il est également appelé jeton d'actualisation.

- Pour conserver le jeton sous forme de variable pour une utilisation ultérieure, cliquez sur **Créer une variable**, entrez un nom pour la variable et cliquez sur **Enregistrer**.
- Pour conserver le jeton sous forme de valeur de texte pour une utilisation ultérieure, cliquez sur **Copier** et collez le jeton dans un fichier texte pour l'enregistrer localement.

Vous pouvez choisir à la fois de créer une variable et de stocker le jeton dans un fichier texte pour une utilisation ultérieure.

3 Cliquez sur **Fermer**.

Si vous avez créé une variable, le jeton d'API affiche le nom de variable que vous avez entré. Si vous avez copié le jeton, le jeton d'API affiche le jeton masqué.

e Pour valider le jeton et les détails du point de terminaison, cliquez sur **Valider**.

Votre jeton expire après 90 jours.

- f Cliquez sur **Créer**.
- g Sur la fiche de l'écouteur, cliquez sur **Connecter**.

L'écouteur commence à surveiller toute l'activité sur le serveur Gerrit et écoute tous les déclencheurs activés sur ce serveur. Pour arrêter d'écouter un déclencheur sur ce serveur, désactivez le déclencheur.

---

**Note** Pour mettre à jour un point de terminaison Gerrit connecté à un écouteur, vous devez déconnecter l'écouteur avant de mettre à jour le point de terminaison.

- Cliquez sur **Configurer > Déclencheurs > Gerrit**.
  - Cliquez sur l'onglet **Écouteurs**.
  - Cliquez sur **Déconnecter** sur l'élécteur connecté au point de terminaison que vous souhaitez mettre à jour.
- 

- 3 Cliquez sur l'onglet **Déclencheurs**, puis sur **Nouveau déclencheur Gerrit**.

- 4 Sélectionnez un projet sur le serveur Gerrit.

- 5 Entrez un nom.

Le nom du déclencheur Gerrit doit être unique.

- 6 Sélectionnez un écouteur Gerrit configuré.

vRealize Automation Code Stream utilise la sélection de l'écouteur Gerrit pour fournir une liste de projets Gerrit disponibles sur le serveur.

- 7 Sélectionnez un projet sur le serveur Gerrit.

- 8 Entrez la branche dans le référentiel à surveiller.

- 9 (Facultatif) Précisez des inclusions ou des exclusions de fichier comme conditions pour le déclencheur.

- Vous fournissez des inclusions de fichiers afin que les pipelines se déclenchent. Lorsqu'un des fichiers d'une validation correspond aux fichiers spécifiés dans les chemins d'inclusion ou regex, les pipelines se déclenchent. Avec un regex spécifié, vRealize Automation Code Stream déclenche uniquement les pipelines dont les noms de fichier dans l'ensemble de modifications correspondent à l'expression fournie. Le filtre regex est utile lors de la configuration d'un déclencheur pour plusieurs pipelines sur un référentiel unique.
- Vous fournissez des exclusions de fichiers pour empêcher le déclenchement des pipelines. Lorsque tous les fichiers d'une validation correspondent aux fichiers spécifiés dans les chemins d'exclusion ou regex, les pipelines ne se déclenchent pas.
- Lorsque cette option est activée, l'exclusion par priorité garantit que les pipelines ne sont pas déclenchés, même si l'un des fichiers d'une validation correspond aux fichiers spécifiés dans les chemins d'exclusion ou regex. Le paramètre par défaut est désactivé.

Si les conditions d'inclusion et d'exclusion sont satisfaites, les pipelines ne sont pas déclenchés.



Dans l'exemple suivant, les inclusions et les exclusions de fichiers sont des conditions de déclenchement.

The screenshot shows a configuration window titled 'Fichier' with a sub-header 'Inclusions'. Below it, there are two sections: 'Inclusions' and 'Exclusions'. Each section has a dropdown menu for the file type (PLAIN or REGEX) and a text input for the file path. The 'Inclusions' section shows a single entry: 'runtime/src/main/a.java' with a '+' button. The 'Exclusions' section shows three entries: 'runtime/pom.xml' with a '-' button, 'runtime/demo.yaml' with a '-' button, and 'runtime/demo.yaml' with a '+' button. At the bottom, there is a toggle switch for 'Exclusion par priorité' which is currently turned off.

Type	File Path	Action
PLAIN	runtime/src/main/a.java	+
REGEX	([a-z A-Z])+[/][a-z A-Z])+	-
PLAIN	runtime/pom.xml	-
PLAIN	runtime/demo.yaml	-
PLAIN	runtime/demo.yaml	+

Exclusion par priorité: ☐

- Pour les inclusions de fichiers, une validation comportant une modification apportée à `runtime/src/main/a.java` ou un fichier Java déclenche les pipelines configurés dans la configuration des événements.
- En ce qui concerne les exclusions de fichiers, une validation comportant des modifications dans les deux fichiers uniquement ne déclenche pas les pipelines configurés dans les configurations d'événement.

## 10 Cliquez sur **Nouvelle configuration**.

- Pour un événement Gerrit, sélectionnez **Jeu de correctifs créé**, **Version préliminaire publiée** ou **Modification fusionnée**. Pour un transfert direct vers Git en contournant Gerrit, sélectionnez **Transfert Git direct**.
- Sélectionnez le pipeline à déclencher.

Si le pipeline a été configuré avec des paramètres d'entrée personnalisés supplémentaires, la liste des paramètres d'entrée affiche les paramètres et les valeurs. Vous pouvez saisir des valeurs pour les paramètres d'entrée qui seront transmis au pipeline avec l'événement déclencheur. Sinon, vous pouvez laisser les valeurs vides ou utiliser les valeurs par défaut.

**Note** Si les valeurs par défaut sont définies :

- Toutes les valeurs que vous entrez pour les paramètres d'entrée remplaceront les valeurs par défaut définies dans le modèle de pipeline.
- Les valeurs par défaut utilisées pour configurer le déclencheur ne seront pas mises à jour si les valeurs des paramètres dans le modèle de pipeline sont modifiées.

Pour plus d'informations sur les paramètres d'entrée injectés automatiquement pour les déclencheurs Gerrit, reportez-vous aux [conditions préalables](#).

- Pour **Jeu de correctifs créé**, **Version préliminaire publiée** et **Modification fusionnée**, certaines actions s'affichent par défaut avec des étiquettes. Vous pouvez modifier l'étiquette ou ajouter des commentaires. Ensuite, lorsque le pipeline s'exécute, l'étiquette ou le commentaire s'affiche dans l'onglet **Activité** en tant qu'**Action effectuée** pour le pipeline.
- Cliquez sur **Enregistrer**.

Pour ajouter plusieurs événements déclencheurs sur plusieurs pipelines, cliquez de nouveau sur **Nouvelle configuration**.

Dans l'exemple suivant, vous pouvez voir des événements pour trois pipelines :

- Si un événement **Modification fusionnée** se produit dans le projet Gerrit, **Pipeline-Gerrit** est déclenché.
- Si un événement **Jeu de correctifs créé** se produit dans le projet Gerrit, **Pipeline-Déclencheur-Gerrit** et **Pipeline-Démo-Gerrit** se déclenchent.

Type d'événement	Pipeline	Etiquette
Change Merged	Gerrit-Pipeline	
Patchset Created	Gerrit-Trigger-Pipeline	Verified
Patchset Created	Gerrit-Demo-Pipeline	Verified

3 éléments

#### 11 Cliquez sur **Créer**.

Le déclencheur Gerrit apparaît sous la forme d'une nouvelle fiche dans l'onglet **Déclencheurs** et est défini sur **Désactivé** par défaut.

#### 12 Sur la fiche de déclencheur, cliquez sur **Activer**.

Lorsque le déclencheur est activé, il utilise l'écouteur Gerrit pour commencer à surveiller les événements qui se produisent sur la branche du projet Gerrit.

Lorsque vous créez le déclencheur, vous incluez le référentiel dans lequel le code est validé. Si vous souhaitez créer un déclencheur avec les mêmes conditions d'inclusion ou d'exclusion de fichier, mais avec un référentiel différent, par exemple, vous pouvez cliquer sur **Actions** > **Cloner** sur la fiche de déclencheur. Cliquez ensuite sur **Ouvrir** sur le nouveau déclencheur et modifiez les paramètres.

### Résultats

Félicitations ! Vous avez correctement configuré un déclencheur Gerrit avec deux événements sur trois pipelines différents.

### Étape suivante

Après avoir validé une modification de code dans le projet Gerrit, vérifiez l'onglet **Activité** de l'événement Gerrit dans vRealize Automation Code Stream. Vérifiez que la liste d'activités inclut les entrées correspondant à chaque exécution de pipeline configurée dans le déclencheur. Lorsqu'un événement se produit, seuls les pipelines du déclencheur Gerrit qui sont associés au type d'événement particulier s'exécutent. Dans cet exemple, si un jeu de correctifs est créé, seuls **Pipeline-Déclencheur-Gerrit** et **Pipeline-Démo-Gerrit** s'exécutent.

Les informations contenues dans les colonnes de l'onglet **Activité** décrivent chaque événement déclencheur Gerrit. Vous pouvez choisir les colonnes à afficher.

- Les colonnes **Modifier l'objet** et **Exécution** sont vides si le déclencheur était un transfert Git direct.
- La colonne **Déclencheur Gerrit** affiche le déclencheur qui a créé l'événement.
- La colonne **Écouteur** est désactivée par défaut. Lorsqu'elle est sélectionnée, cette colonne affiche l'écouteur Gerrit qui a reçu l'événement. Un écouteur peut être associé à plusieurs déclencheurs.
- La colonne **Type de déclencheur** est désactivée par défaut. Lorsqu'elle est sélectionnée, cette colonne indique si le déclencheur a été déclenché manuellement ou automatiquement.

## Gerrit

Activity Triggers Listeners

TRIGGER MANUALLY ⓘ

	Commit Time	Change#	Change Subject	Execution	Status	Message	Action taken	User	Gerrit project
⋮	Nov 12, 2019, 12:47:53 PM	19570 /4	111Dummy	Gerrit-Pipeline #1	COMPLETED	Execution Completed.	Verified +1	Pravin Kumar spnyank@vm	test1
⋮	Nov 12, 2019, 12:50:04 PM	19570 /6	11111Dummy	Gerrit-Pipeline #2	WAITING	Stage0.Task0: Execution Waiting for User Action.		Pravin Kumar spnyank@vm	test1
			11111Dummy	Gerrit-Demo-Pipeline #1	FAILED	Stage0.Task0: User Operation request has been rejected by fritz.	Verified -1	Pravin Kumar spnyank@vm	test1
			11111Dummy	Gerrit-Trigger-Pipeline #1	WAITING	Stage0.Task0: Execution Waiting for User Action.		Pravin Kumar spnyank@vm	test1

Show columns

- ☒ Change#
- ☒ Change Subject
- ☒ Execution
- ☒ Status
- ☒ Message
- ☒ Action taken
- ☒ User
- ☒ Gerrit project
- ☒ Gerrit Trigger
- ☐ Listener
- ☒ Branch
- ☒ Event
- ☐ Trigger Type

SELECT ALL

Pour contrôler l'activité correspondant à une exécution terminée ou ayant échoué, cliquez sur les trois points à gauche d'une entrée, sur l'écran Activité.

- Si le pipeline ne parvient pas à s'exécuter en raison d'une erreur dans le modèle de pipeline ou d'un autre problème, corrigez l'erreur et sélectionnez **Exécuter à nouveau** pour exécuter le pipeline à nouveau.
- Si le pipeline ne parvient pas à s'exécuter en raison d'un problème de connectivité réseau ou d'un autre problème, sélectionnez **Reprendre** pour redémarrer la même exécution de pipeline. Cela permet de gagner en temps d'exécution.
- Utilisez **Afficher l'exécution** pour basculer vers l'écran Exécution. Reportez-vous à la section [Comment exécuter un pipeline et afficher les résultats](#).
- Utilisez **Supprimer** pour supprimer l'entrée de l'écran Activité.

Si un événement Gerrit ne parvient pas à déclencher un pipeline, vous pouvez cliquer sur le bouton **Déclencher manuellement**, et entrer le nom du déclencheur Gerrit et l'ID de modification.

# Surveillance des pipelines dans vRealize Automation Code Stream



En tant qu'administrateur ou développeur de vRealize Automation Code Stream, vous utilisez des tableaux de bord vRealize Automation Code Stream pour surveiller les tendances et les résultats de l'exécution d'un pipeline. Vous pouvez utiliser les tableaux de bord de pipeline par défaut pour surveiller un seul pipeline ou créer des tableaux de bord personnalisés pour surveiller plusieurs pipelines.

## Présentation des tableaux de bord de pipeline

Un tableau de bord de pipeline est une vue des résultats de l'exécution d'un pipeline spécifique, tels que les tendances, les principaux échecs et les modifications réussies. vRealize Automation Code Stream crée le tableau de bord de pipeline lorsque vous créez un pipeline.

## Présentation des tableaux de bord personnalisés

Un tableau de bord personnalisé est une vue des résultats de l'exécution d'un ou de plusieurs pipelines. Vous créez le tableau de bord personnalisé et ajoutez des widgets pour afficher les résultats que vous souhaitez voir. Par exemple, vous pouvez créer un tableau de bord à l'échelle du projet avec des KPI et des mesures collectés auprès de plusieurs pipelines. Si un avertissement ou un échec d'exécution est signalé, vous pouvez utiliser le tableau de bord pour résoudre l'échec.

Ce chapitre contient les rubriques suivantes :

- [Suivi des indicateurs de performance clés d'un pipeline dans vRealize Automation Code Stream](#)

## Suivi des indicateurs de performance clés d'un pipeline dans vRealize Automation Code Stream

En tant qu'administrateur ou développeur de vRealize Automation Code Stream, vous avez besoin de visualiser les performances de vos pipelines dans vRealize Automation Code Stream. Vous devez connaître l'efficacité de vos pipelines en matière de publication de code, du développement à la production en passant par les tests.

Pour une meilleure visibilité, utilisez le tableau de bord par défaut de votre pipeline ou un tableau de bord personnalisé.

- Les mesures de pipeline incluent des statistiques, telles que les durées moyennes, disponibles sur le tableau de bord des pipelines.
- Pour observer les mesures sur plusieurs pipelines, utilisez les tableaux de bord personnalisés.

vRealize Automation Code Stream peut mesurer pour vous les durées moyennes avant la récupération, la livraison ou l'échec d'un pipeline au fil du temps, et afficher les tendances associées.

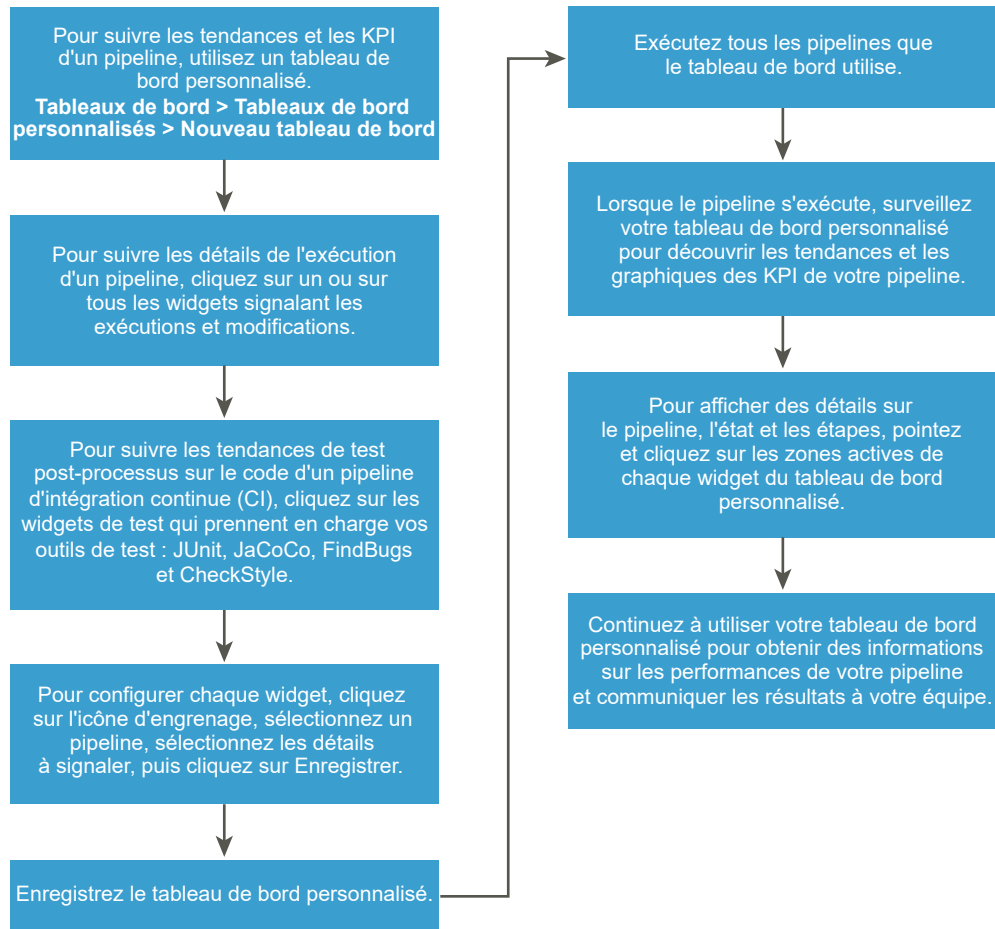
**Tableau 8-1. Mesure des durées moyennes**

Éléments mesurés...	Signification
CI moyenne	Durée moyenne passée en phase d'intégration continue, mesurée par heure dans le type de tâche CI.
Durée moyenne avant livraison d'un pipeline	Durée moyenne requise par le pipeline pour livrer une mise à jour après son déclenchement.
Durée moyenne entre deux déploiements réussis	Durée entre deux déploiements réussis, reflet de la fréquence de mise à jour d'un environnement de production.
Durée moyenne avant l'échec d'un pipeline	Durée avant l'échec d'un pipeline après son déclenchement.
Durée moyenne avant récupération suite à l'échec d'un pipeline	Durée moyenne avant livraison d'un pipeline réussi après son échec. Mesure la durée séparant l'échec d'un critère de build ou de test de la build suivante qui génère une exécution de pipeline réussie ; durée moyenne calculée sur une semaine ou un mois.

Vous pouvez également configurer vRealize Automation Code Stream de manière à afficher les premières tâches et étapes ayant échoué dans un pipeline. Cette mesure indique le nombre et le pourcentage d'échecs pour les environnements de développement et de post-développement de chaque pipeline et de chaque projet, en moyenne, sur une semaine ou un mois. Vous pouvez observer les principaux échecs pour résoudre les problèmes du processus d'automatisation de la publication.

Par exemple, vous pouvez configurer l'affichage pour une durée particulière, comme les sept derniers jours, et noter les premières tâches ayant échoué pendant ce laps de temps. Si vous apportez une modification dans votre environnement ou pipeline et que vous exécutez de nouveau ce dernier, vérifiez les premières tâches ayant échoué sur une durée plus longue, comme les 14 derniers jours : les premières tâches ayant échoué peuvent avoir changé. Avec ce résultat, vous aurez la certitude que la modification de votre processus d'automatisation de la publication a amélioré le taux de réussite de l'exécution de votre pipeline.

Pour suivre les tendances et les indicateurs de performance clés de vos pipelines à l'aide d'un tableau de bord personnalisé, ajoutez des widgets au tableau de bord et configurez-les pour générer des rapports sur vos pipelines.



#### Conditions préalables

- Vérifiez qu'au moins un pipeline existe. Dans l'interface utilisateur, cliquez sur **Pipelines**.
- Pour les pipelines que vous souhaitez surveiller, vérifiez qu'ils ont bien été exécutés. Cliquez sur **Exécutions**.

#### Procédure

- 1 Pour créer un tableau de bord personnalisé, cliquez sur **Tableaux de bord > Tableaux de bord personnalisés > Nouveau tableau de bord**.

- 2 Pour personnaliser le tableau de bord afin qu'il génère des rapports sur des tendances et des indicateurs de performance clés spécifiques pour votre pipeline, cliquez sur un widget.

Par exemple, pour afficher des détails sur l'état, les étapes, les tâches, la durée d'exécution et l'utilisateur à l'origine de l'exécution du pipeline, cliquez sur le widget **Détails de l'exécution**. Pour un pipeline d'intégration continue (CI), vous pouvez également suivre les tendances en matière de post-traitement en utilisant les widgets correspondant à JUnit, à JaCoCo, à FindBugs et à CheckStyle.

IX KPIS ACTIONS -

Détails de l'exécution

Exécution#	État	Message d'état	Toutes les tâches	Task0 (Stage0)	Durée
#22	WAITING	Stage0.Task0: Execution Waiting for User Action.			3 heures, 12 secondes
#21	COMPLETED	Execution Completed.			17 secondes

Éléments par page 10 1 - 10 sur 22 éléments

- 3 Configurez chaque widget que vous ajoutez.
  - a Sur le widget, cliquez sur l'icône d'engrenage.
  - b Sélectionnez un pipeline, définissez les options disponibles et sélectionnez les colonnes à afficher.
  - c Pour enregistrer la configuration du widget, cliquez sur **Enregistrer**.
  - d Pour enregistrer le tableau de bord personnalisé, cliquez sur **Enregistrer**, puis sur **Fermer**.
- 4 Pour afficher plus d'informations sur le pipeline, cliquez sur les zones actives des widgets.

Par exemple, dans le widget **Détails de l'exécution**, cliquez sur une entrée de la colonne État pour afficher plus d'informations sur l'exécution du pipeline. Sur le widget **Dernière modification réussie**, pour afficher un résumé de l'étape et de la tâche du pipeline, cliquez sur le lien actif.

## Résultats

Félicitations ! Vous avez créé un tableau de bord personnalisé qui surveille les tendances et les KPI de vos pipelines.

## Étape suivante

Poursuivez la surveillance des performances de vos pipelines dans vRealize Automation Code Stream, et partagez les résultats avec votre responsable et vos équipes pour continuer à améliorer le processus de publication de vos applications.



# En savoir plus sur Code Stream

# 9

Il existe de nombreuses façons pour les administrateurs et les développeurs de vRealize Automation Code Stream d'en savoir plus sur vRealize Automation Code Stream et sur ce qu'il peut leur offrir.

Vous pouvez utiliser cette documentation pour en savoir plus sur les pipelines et leur exécution, et sur l'ajout de points de terminaison, de projets, etc.

Familiarisez-vous avec les autorisations que les rôles fournissent. Découvrez comment utiliser des ressources limitées et exiger des approbations sur les pipelines. Reportez-vous à la section [Gestion de l'accès et des approbations utilisateur dans vRealize Automation Code Stream](#).

Appréhendez la valeur de la recherche en déterminant l'emplacement de tâches ou de composants spécifiques dans vos pipelines, exécutions ou points de terminaison.

Ce chapitre contient les rubriques suivantes :

- [Présentation de la recherche dans vRealize Automation Code Stream](#)
- [Plus de ressources pour les administrateurs et les développeurs vRealize Automation Code Stream](#)

## Présentation de la recherche dans vRealize Automation Code Stream

Vous utilisez la recherche pour localiser des éléments spécifiques ou d'autres composants. Par exemple, vous pouvez rechercher des pipelines activés ou désactivés. Si un pipeline est désactivé, il ne peut pas s'exécuter.

### Objets de recherche

Vous pouvez effectuer une recherche dans :

- Projets
- Des points de terminaison
- Des pipelines
- Des exécutions
- Des tableaux de bord de pipeline, des tableaux de bord personnalisés

- Des déclencheurs et serveurs Gerrit
- Des Webhooks Git
- Des Webhooks Docker

Vous pouvez effectuer une recherche avec filtrage sur colonne dans :

- Des opérations d'utilisateur
- Des variables
- L'activité des déclencheurs pour Gerrit, Git et Docker

Vous pouvez effectuer une recherche avec filtrage sur grille sur la page **Activité** de n'importe quel déclencheur.

## Fonctionnement de la recherche

Les critères de recherche varient en fonction de la page sur laquelle vous vous trouvez. Chaque page a des critères de recherche différents.

Zone de recherche	Critères à utiliser pour la recherche
Tableaux de bord de pipeline	Projet, Nom, Description, Balises, Lien
Tableaux de bord personnalisés	Projet, Nom, Description, Lien (UUID d'un élément du tableau de bord)
Des exécutions	Nom, Commentaires, Motif, Balises, Index, État, Projet, Afficher, Exécuté par moi, Lien (UUID de l'exécution) et Paramètres d'entrée, Paramètres de sortie ou Message d'état par le biais du format suivant : <code>&lt;key&gt;:&lt;value&gt;</code>
Des pipelines	Nom, Description, État, Balises, Créé par, Créé par moi, Mis à jour par, Mis à jour par moi, Projet
Projets	Nom, Description
Des points de terminaison	Nom, Description, Type, Mis à jour par, Projet
Déclencheurs Gerrit	Nom, État, Projet
Serveurs Gerrit	Nom, URL du serveur, Projet
Des Webhooks Git	Nom, Type de serveur, Référentiel, Branche, Projet

Où :

- Le lien est l'UUID d'un pipeline, d'une exécution ou d'un widget sur un tableau de bord.
- Pour spécifier les critères Paramètre d'entrée, Paramètre de sortie et Message d'état, utilisez les notations suivantes :
  - Notation : `input.<inputKey>:<inputValue>`

Exemple : `input.GERRIT_CHANGE_OWNER_EMAIL:joe_user`

- Notation : `output.<outputKey>:<outputValue>`

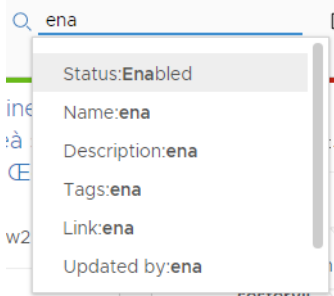

Exemple : `output.BuildNo:29`

- Notation : `statusMessage:<value>`

Exemple : `statusMessage:Execution failed`

- L'état dépend de la page de recherche.
  - Pour les exécutions, les valeurs possibles sont : Terminé, Échec, Échec de la restauration ou Annulé.
  - Pour les pipelines, les valeurs d'état possibles incluent : activé, désactivé ou publié.
  - Pour les déclencheurs, les valeurs d'état possibles incluent : activé ou désactivé.
- Exécuté, Créé ou Mis à jour par moi se rapporte à l'utilisateur connecté.

La recherche s'affiche dans la partie supérieure droite de chaque page valide. Lorsque vous commencez à taper dans la zone de recherche, vRealize Automation Code Stream connaît le contexte de la page et suggère des options de recherche.

Méthodes de recherche	Modalités de saisie
<p>Tapez une partie du paramètre de recherche.</p> <p>Par exemple, saisissez <b>act</b> pour ajouter un filtre d'état répertoriant tous les pipelines activés.</p>	 <p>The screenshot shows a search bar with the text 'ena'. A dropdown menu is open, displaying several suggestions: 'Status:Enabled', 'Name:ena', 'Description:ena', 'Tags:ena', 'Link:ena', and 'Updated by:ena'.</p>
<p>Ajoutez un filtre pour réduire le nombre d'éléments trouvés.</p> <p>Par exemple, tapez <b>Tes</b> pour ajouter un filtre de nom. Le filtre fonctionne comme un <b>ET</b> avec le filtre <b>État:désactivé</b> existant pour afficher uniquement les pipelines désactivés dont le nom comporte <b>Tes</b>.</p>	 <p>The screenshot shows the search bar with the text 'Tes'. Above the search bar, there are two filter buttons: 'Status' and 'Disabled'. Below the search bar, a list of search results is displayed, including 'Name:Tes', 'Description:Tes', 'Tags:Tes', 'Link:Tes', 'Project:218 test result', and 'Updated by:Tes'.</p>

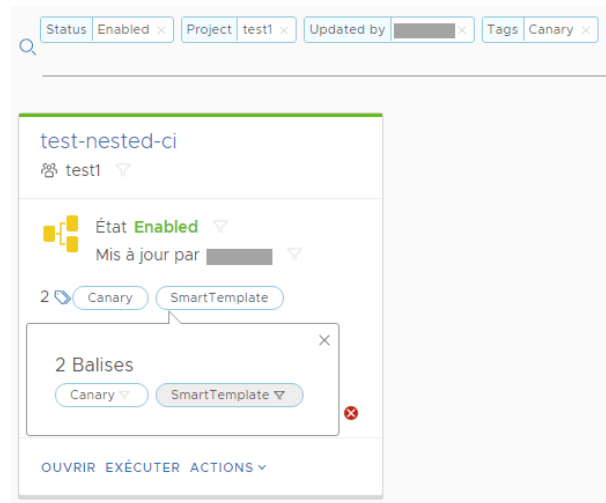
## Méthodes de recherche

Cliquez sur l'icône de filtre dans les propriétés d'un pipeline ou d'une exécution pour réduire le nombre d'éléments affichés.

- Les propriétés **État**, **Balises**, **Projet** et **Mis à jour par** des pipelines disposent d'une icône de filtre.
- Les propriétés **Balises**, **Exécuté par** et **Message d'état** des exécutions disposent d'une icône de filtre.

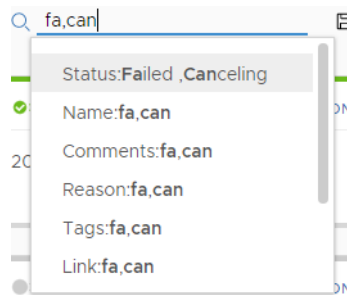
Par exemple, sur la fiche de pipeline, cliquez sur l'icône pour ajouter la balise **SmartTemplate** aux filtres existants : **État:Activé**, **Projet:test**, **Mis à jour par:utilisateur** et **Balises:Canary**.

## Modalités de saisie



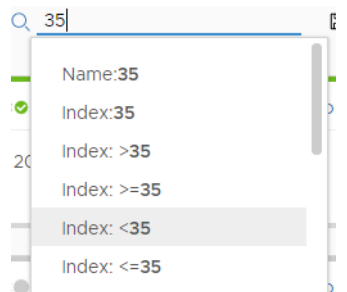
Utilisez une virgule de séparation pour inclure tous les éléments dans deux états d'exécution.

Par exemple, tapez **éc,ann** pour créer un filtre d'état qui fait office de **ou** pour répertorier toutes les exécutions en échec ou annulées.



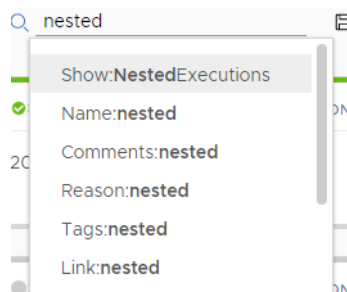
Tapez un nombre pour inclure tous les éléments dans une plage d'index.

Par exemple, tapez **35** et sélectionnez **<** pour répertorier toutes les exécutions dont le numéro d'index est inférieur à 35.



Les pipelines modélisés en tant que tâches deviennent des exécutions imbriquées et ne sont pas répertoriés avec toutes les exécutions par défaut.

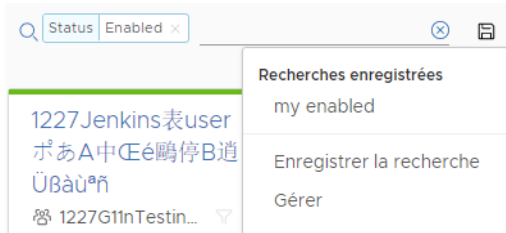
Pour afficher les exécutions imbriquées, tapez **imbriqué** et sélectionnez le filtre **Afficher**.



## Enregistrement d'une recherche favorite

Vous pouvez enregistrer des recherches favorites à utiliser sur chaque page en cliquant sur l'icône de disque en regard de la zone de recherche.

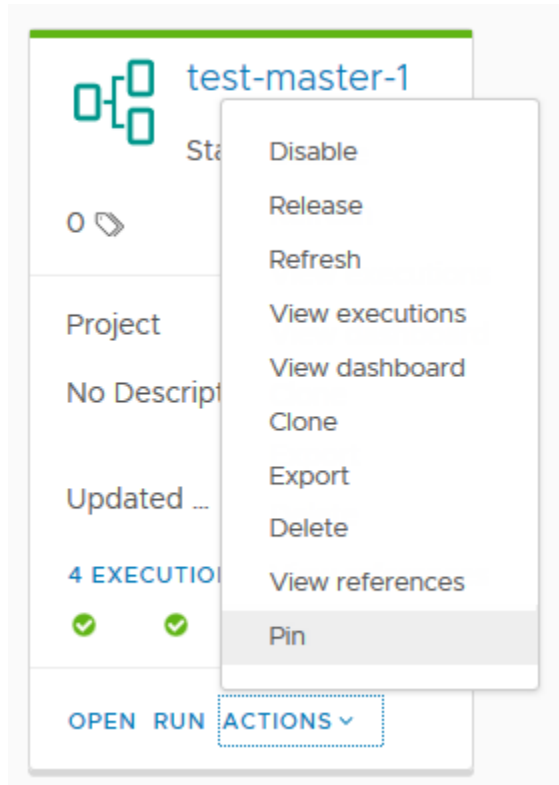
- Pour enregistrer une recherche, saisissez les paramètres de recherche et cliquez sur l'icône pour donner un nom à la recherche, comme **Mes pipelines activés**.
- Après avoir enregistré une recherche, cliquez sur l'icône pour accéder à la recherche. Vous pouvez également sélectionner **Gérer** pour renommer, supprimer ou déplacer la recherche dans la liste des recherches enregistrées.



Les recherches sont liées à votre nom d'utilisateur et ne s'affichent que sur les pages auxquelles la recherche s'applique. Par exemple, si vous avez enregistré une recherche nommée **Mes pipelines activés** correspondant à **État:activé** sur la page Pipelines, la recherche **Mes pipelines activés** n'est pas disponible sur la page des déclencheurs Gerrit, même si **État:activé** est une recherche valide pour un déclencheur.

## Enregistrement d'un pipeline favori

Si vous disposez d'un pipeline ou d'un tableau de bord favori, vous pouvez l'épingler de manière à ce qu'il figure toujours en haut de votre page Pipelines ou Tableaux de bord. Sur la fiche de pipeline, cliquez sur **Actions > Épingler**.



## Plus de ressources pour les administrateurs et les développeurs vRealize Automation Code Stream

En tant qu'administrateur ou développeur de vRealize Automation Code Stream, vous pouvez approfondir vos connaissances sur vRealize Automation Code Stream.

Tableau 9-1. Plus de ressources pour les administrateurs

Pour en savoir plus sur...	Consultez ces ressources...
<p>Les autres méthodes mises à la disposition des administrateurs pour utiliser vRealize Automation Code Stream :</p> <ul style="list-style-type: none"> <li>■ Configurer les pipelines pour automatiser les tests et la libération des applications natives de cloud.</li> <li>■ Automatiser et tester le code source des développeurs, par le biais des tests, jusqu'à la production.</li> <li>■ Configurer les pipelines pour que les développeurs puissent tester les modifications avant de les valider dans la branche principale.</li> <li>■ Effectuer le suivi des mesures clés de pipeline.</li> </ul>	<p>vRealize Automation Code Stream</p> <ul style="list-style-type: none"> <li>■ <a href="#">Documentation de vRealize Automation</a></li> <li>■ <a href="#">Site Web produit vRealize Automation</a></li> </ul> <p>Ateliers VMware</p> <ul style="list-style-type: none"> <li>■ Utilisez la <a href="#">Communauté vRealize Automation</a>.</li> <li>■ Utilisez la <a href="#">Zone d'apprentissage VMware</a>.</li> <li>■ Consultez les <a href="#">Blogs VMware</a>.</li> <li>■ Inscrivez-vous aux <a href="#">Ateliers pratiques VMware</a>.</li> </ul>

Tableau 9-2. Plus de ressources pour les développeurs

Pour en savoir plus sur...	Consultez ces ressources...
<p>Les autres méthodes mises à la disposition des développeurs pour utiliser vRealize Automation Code Stream :</p> <ul style="list-style-type: none"> <li>■ Utiliser les images de registre public et privé pour construire les environnements pour les nouvelles applications ou les nouveaux services.</li> <li>■ Configurer les environnements de développement pour pouvoir créer des branches à partir du dernier build stable.</li> <li>■ Mettre à jour les environnements de développement avec les modifications de code et les artefacts les plus récents.</li> <li>■ Tester les modifications du code non validées par rapport aux derniers builds stables des autres services dépendants.</li> <li>■ Recevoir une notification lorsqu'une modification validée dans un pipeline CI/CD principal rompt les autres services.</li> </ul>	<p>vRealize Automation Code Stream</p> <ul style="list-style-type: none"> <li>■ <a href="#">Documentation de vRealize Automation</a></li> <li>■ <a href="#">Site Web produit vRealize Automation</a></li> </ul> <p>Ateliers VMware</p> <ul style="list-style-type: none"> <li>■ Utilisez la <a href="#">Communauté vRealize Automation</a>.</li> <li>■ Utilisez la <a href="#">Zone d'apprentissage VMware</a>.</li> <li>■ Consultez les <a href="#">Blogs VMware</a>.</li> <li>■ Inscrivez-vous aux <a href="#">Ateliers pratiques VMware</a>.</li> </ul>