Utilisation et gestion de vRealize Automation Code Stream

14 décembre 2022

vRealize Automation 8.7

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

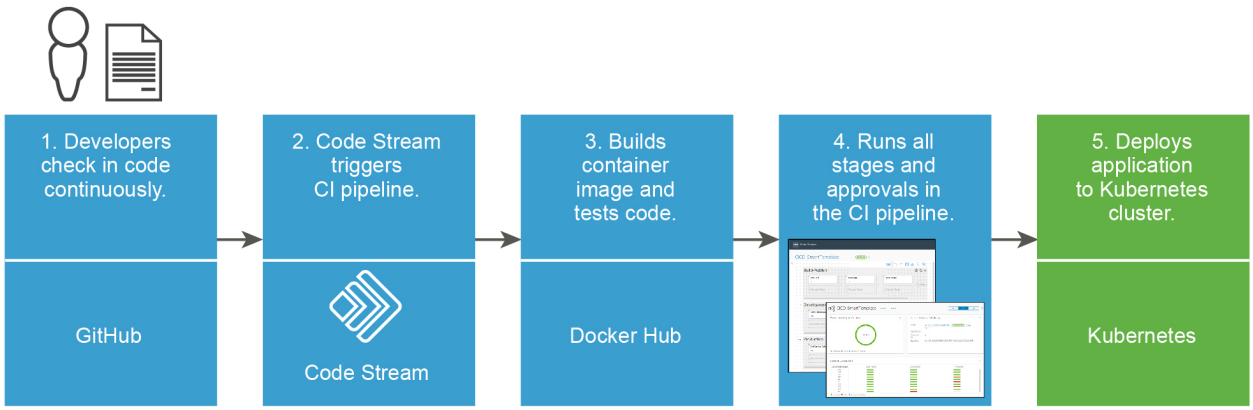
VMware France SAS.
Tour Franklin
100-101 Terrasse Boieldieu
92042 Paris La Défense 8 Cedex
France
www.vmware.com/fr

Table des matières

Présentation et fonctionnement de Code Stream

1

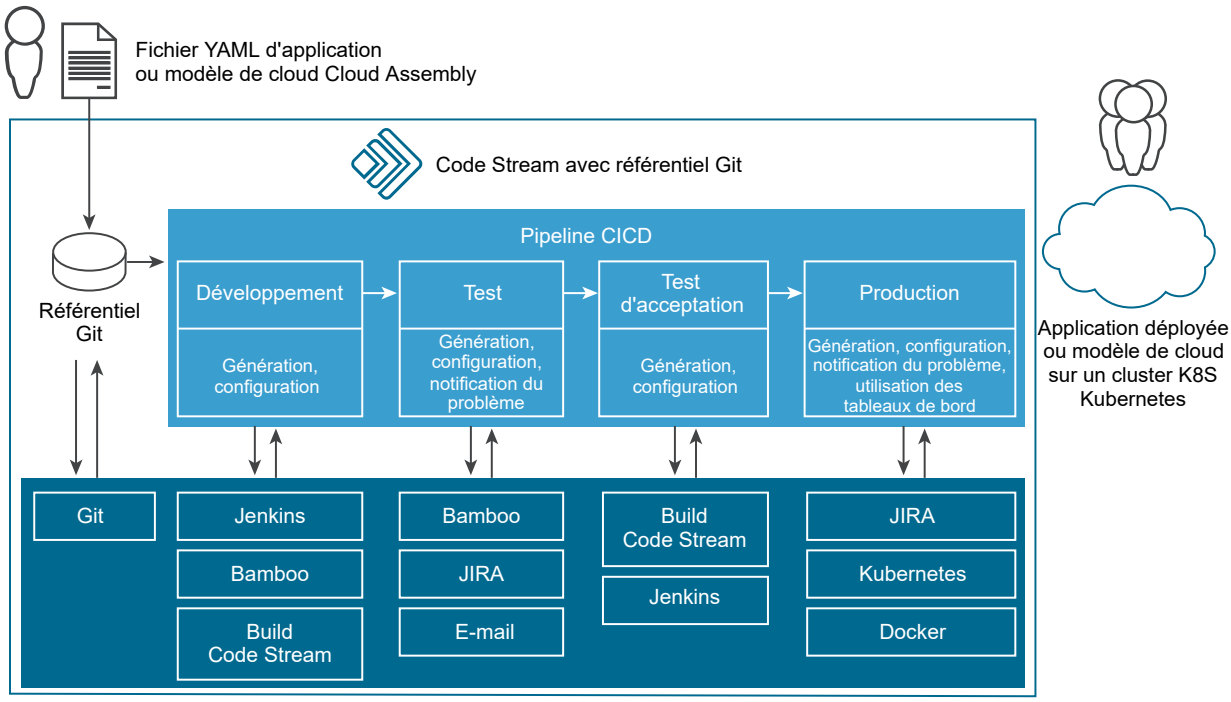
vRealize Automation Code Stream™ est un outil d'intégration continue et de prestation continue (CICD). En créant des pipelines qui modélisent le processus de publication logicielle dans votre cycle de vie DevOps, vous créez l'infrastructure de code qui livre votre logiciel rapidement et en continu.



Lorsque vous utilisez 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 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. 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 Code Stream .

Utilisation de Code Stream par les administrateurs de 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 Code Stream.

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

Pour aider les développeurs…	Voici ce que vous pouvez faire…
Vous fournissez et gérez des environnements.	Vous créez des environnements pour que les développeurs testent et déploient leur code. ■ Vous suivez l'état des environnement et envoyez des notifications par e-mail. ■ Vous assurez la production vos développeurs en veillant à ce que leurs environnements fonctionnent en permanence. Pour en savoir plus, reportez-vous à la section Plus de ressources pour les administrateurs et les développeurs Code Stream. Reportez-vous également à la section Chapitre 5 Didacticiels d'utilisation de Code Stream.
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.

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

Pour aider les développeurs...	Voici ce que vous pouvez faire...
Vous autorisez les intégrations à d'autres services.	Vous vous assurez que les intégrations à d'autres services fonctionnent. Pour en savoir plus, reportez-vous à la documentation VMware Cloud Services.
Créer des pipelines.	Vous créez des pipelines qui modélisent les processus de publication. Pour en savoir plus, reportez-vous à la section Chapitre 3 Création et utilisation de pipelines dans Code Stream.
Déclencher des pipelines.	Vous vous assurez que les pipelines s'exécutent lorsque des événements se produisent. ■ 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. ■ Pour déclencher un pipeline lorsqu'un développeur valide les modifications apportées à son code, vous utilisez le déclencheur Git. ■ Pour déclencher un pipeline lorsque les développeurs révisent le code, la fusion, etc., vous utilisez le déclencheur Gerrit. ■ 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. Pour en savoir plus, reportez-vous à la section Chapitre 7 Déclenchement des pipelines dans Code Stream.
Vous gérez les pipelines et les approbations.	Vous réalisez un suivi continu des pipelines. ■ Vous affichez l'état des pipelines et identifiez les utilisateurs qui ont exécuté les pipelines. ■ Vous affichez les approbations sur les exécutions de pipeline et gérez les approbations pour les exécutions de pipelines actifs et inactifs. Pour en savoir plus, reportez-vous à la section Description des opérations et des approbations d'utilisateur dans Code Stream. Reportez-vous également à la section Utilisation des tableaux de bord personnalisés pour suivre les indicateurs de performance clés d'un pipeline dans Code Stream.

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

Pour aider les développeurs...	Voici ce que vous pouvez faire...
Vous surveillez les environnements des développeurs.	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. Vous pouvez également afficher les éléments suivants : ■ Durée d'exécution d'un pipeline avant aboutissement. ■ Délai d'attente d'un pipeline avant approbation et notification de l'utilisateur qui doit l'approuver. ■ Étapes et tâches qui échouent le plus souvent. ■ Étapes et tâches dont l'exécution est la plus longue. ■ Publications en cours des équipes de développement. ■ Applications dont le déploiement et la publication ont abouti. Pour en savoir plus, reportez-vous à la section Chapitre 8 Surveillance des pipelines dans Code Stream.
Résoudre des problèmes.	Vous dépannez et résolvez les échecs de pipeline dans les environnements des développeurs. ■ Identifier et résoudre les problèmes dans les environnements d'intégration et de livraison continues (CICD). ■ Vous utilisez les tableaux de bord de pipeline et créez des tableaux de bord personnalisés pour en savoir plus. Reportez-vous à la section Chapitre 8 Surveillance des pipelines dans Code Stream. Reportez-vous également à la section Chapitre 2 Configuration de Code Stream pour modéliser le processus de publication.

Code Stream fait partie de VMware Cloud Services.

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

Pour plus d'informations sur les autres actions possibles, consultez la documentation VMware vRealize Automation.

Utilisation de Code Stream par les développeurs

En tant que développeur, vous utilisez 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 Code Stream.

Après avoir exécuté un pipeline, vous souhaiterez savoir :

- Si votre code a passé toutes les étapes du pipeline. Pour le savoir, observez les résultats dans les exécutions de pipeline.
- Les mesures à prendre en cas d'échec du pipeline et les causes de l'échec. Pour le savoir, observez les erreurs les plus fréquentes dans les tableaux de bord de pipeline.

Tableau 1-2. Utilisation de 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 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 normalement pour libérer votre logiciel. Code Stream automatise ensuite le processus qui crée, teste, approuve et déploie votre code.

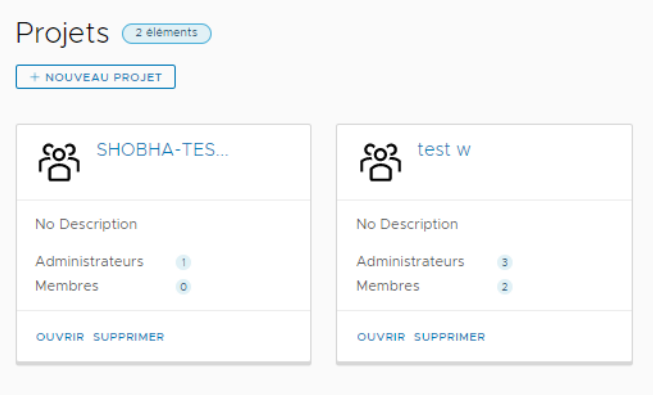
À présent que vous disposez de tout ce qui est nécessaire pour modéliser votre processus de publication logicielle, voici comment le faire dans Code Stream.

Conditions préalables

- Vérifiez si des points de terminaison sont déjà disponibles. Dans Code Stream, cliquez sur **Points de terminaison**.
- Découvrez les méthodes natives avec lesquelles vous pouvez créer et déployer 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 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 Code Stream.
- Si vous disposez du rôle d'utilisateur ou d'observateur au lieu du rôle d'administrateur, identifiez l'administrateur de votre instance de Code Stream.

Procédure

1. Examinez les projets disponibles dans Code Stream et sélectionnez celui qui vous convient.
   - Si aucun projet ne s'affiche, demandez à un administrateur de Code Stream de créer un projet et de vous rendre membre du projet. Reportez-vous à la section Ajout d'un projet dans Code Stream.
   - Si vous n'êtes membre d'aucun projet répertorié, demandez à un administrateur de 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.

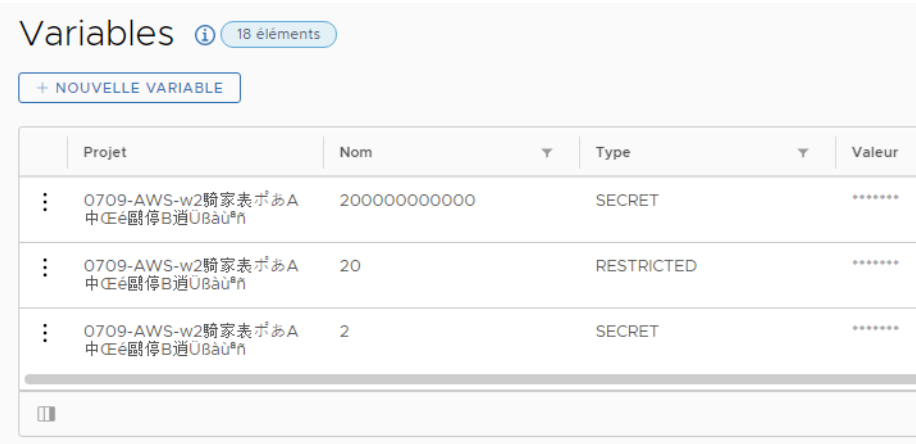
   Par exemple, vous pouvez avoir besoin de Git, Jenkins, Code Stream Build, Kubernetes et JIRA.

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

   Pour limiter les ressources utilisées dans vos pipelines, telles qu'une machine hôte, utilisez des variables limitées. 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 `HostIPAddress`. Pour utiliser la variable dans une tâche de pipeline, saisissez `${var.HostIPAddress}`.



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.

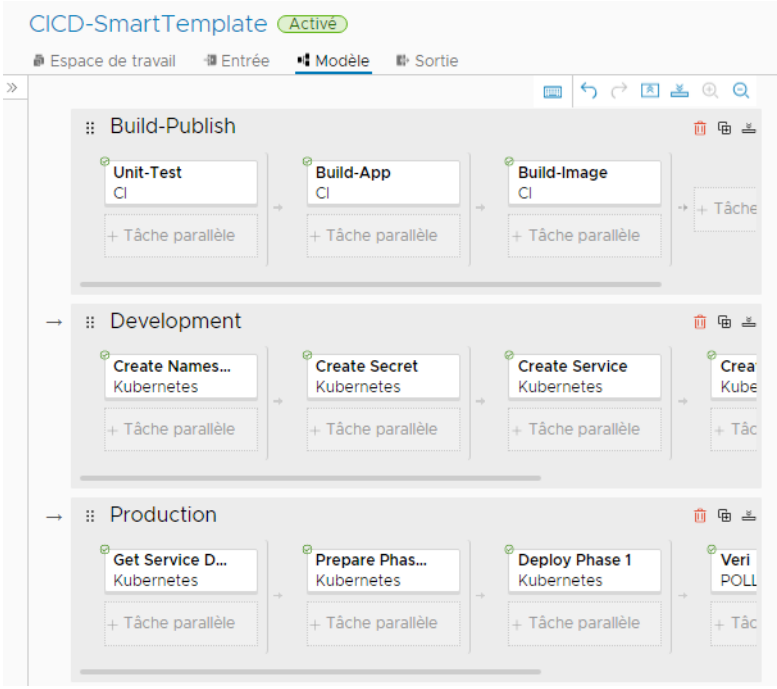
5 Planifiez la stratégie de build pour votre pipeline CICD, CI ou CD natif.

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 Code Stream a besoin afin de pouvoir générer, intégrer, tester et déployer votre code en mode natif.

Création d'une build Code Stream native	Résultats de cette stratégie de build
Utilisez l'un des modèles de pipeline intelligents.	■ 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.	Vous ajoutez des étapes, ajoutez des tâches et entrez les informations qui les remplit.

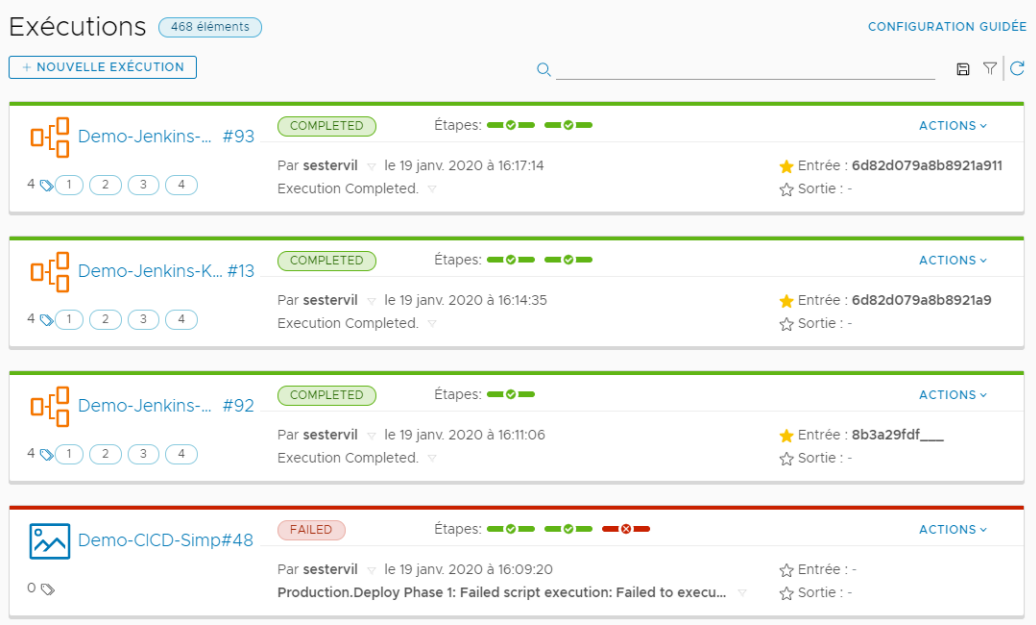
6 Créez votre pipeline à l'aide d'un modèle de cloud intelligent ou ajoutez manuellement des étapes et des tâches au pipeline.

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.

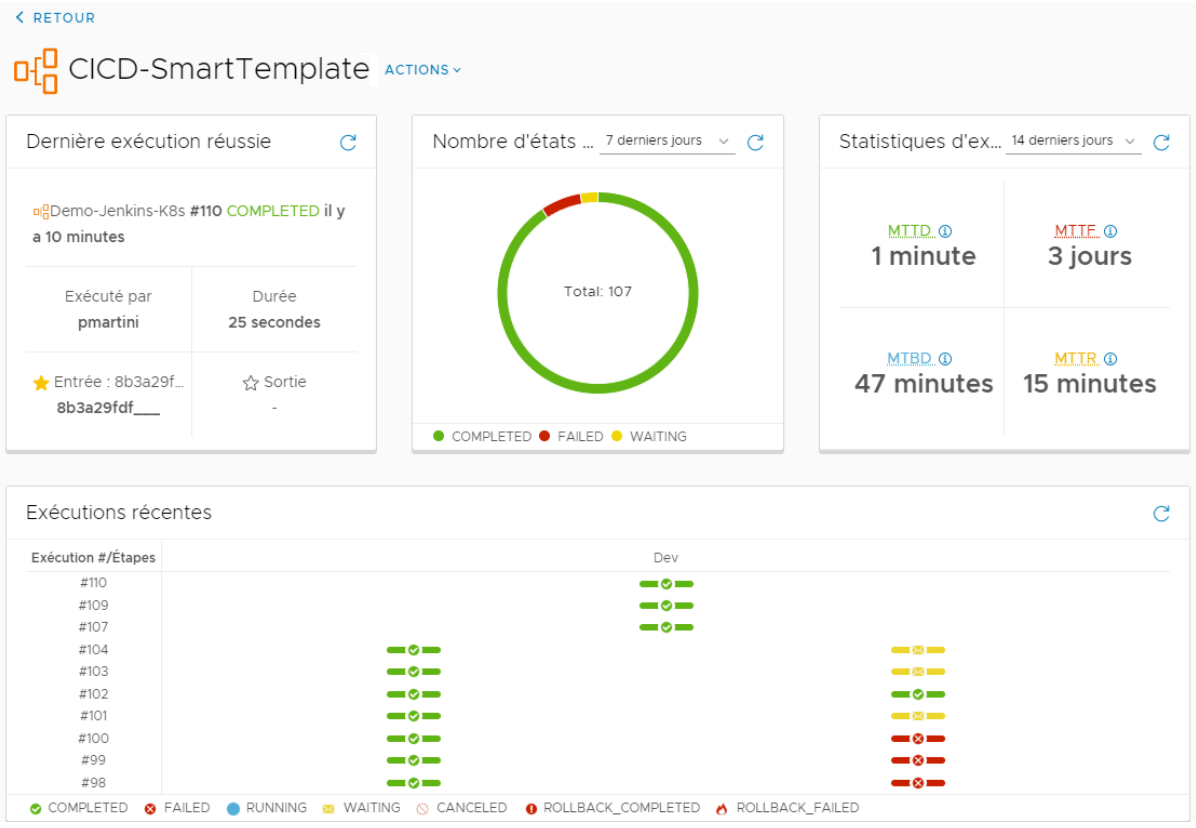


7 Validez, activez et exécutez votre pipeline.

8 Affichez les exécutions de pipeline.



9 Pour suivre l'état et les indicateurs de performance clés (KPI), utilisez les tableaux de bord de pipeline et créez des tableaux de bord personnalisés.



Résultats

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

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

Étape suivante

Découvrez les cas d'utilisation que vous souhaiterez peut-être appliquer dans votre environnement. Reportez-vous à la section Chapitre 5 Didacticiels d'utilisation de Code Stream.

Ajout d'un projet dans Code Stream

Vous créez un projet et y ajoutez des administrateurs et des membres. Les membres du projet peuvent utiliser des fonctionnalités telles que la création d'un pipeline et l'ajout d'un point de terminaison. Pour créer, supprimer ou mettre à jour un projet pour une équipe de développement, vous devez être un administrateur de Code Stream.

Vous ne pouvez créer un pipeline que s'il existe un projet. Lorsque vous créez un pipeline, vous sélectionnez un projet qui rassemble toutes vos informations de 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 Code Stream. Reportez-vous à la section Présentation des rôles dans Code Stream.

  Si vous ne disposez pas du rôle d'administrateur de Code Stream, mais que vous êtes un administrateur dans Cloud Assembly, vous pouvez utiliser l'interface utilisateur de Cloud Assembly pour créer, mettre à jour ou supprimer des projets. Reportez-vous à Ajout d'un projet pour mon équipe de développement 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**, ajoutez des utilisateurs et attribuez des rôles.
   - 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 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 Code Stream à des points de terminaison et au Chapitre 3 Création et utilisation de pipelines dans Code Stream.

Une fois que vous avez créé un pipeline, le nom du projet qui rassemble toutes vos informations de pipeline s'affiche sur les cartes de pipeline et les cartes d'exécution de pipeline.

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

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 Code Stream

En fonction de votre rôle dans 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 Code Stream

	Rôles 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 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 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 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			

Tableau 2-2. Fonctionnalités des rôles de service 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 Code Stream	Rôle d'utilisateur de Code Stream
	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	
	Créer des exécutions	Oui	Oui			
	Mettre à jour les exécutions	Oui	Oui			
	Supprimer les exécutions	Oui	Oui			
Marquer les ressources comme étant limitées						
	Marquer une variable ou un point de terminaison comme étant limité(e)	Oui				
Tableaux de bord						
	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 Code Stream

Vous pouvez créer des rôles personnalisés dans Cloud Assembly qui étendent des privilèges aux utilisateurs qui utilisent des pipelines. Lorsque vous créez un rôle personnalisé pour des pipelines 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	■ Créer, mettre à jour, supprimer et cloner des pipelines. ■ Publier et annuler la publication des pipelines pour VMware Service Broker. ■ Créer, mettre à jour et supprimer des points de terminaison. ■ Créer, mettre à jour et supprimer des variables régulières et secrètes. ■ Créer, cloner, mettre à jour et supprimer un écouteur Gerrit. ■ Connecter et déconnecter un écouteur Gerrit. ■ Créer, cloner, mettre à jour et supprimer un déclencheur Gerrit. ■ Créer, mettre à jour et supprimer un Webhook Git. ■ Créer, mettre à jour et supprimer un Webhook Docker. ■ Utiliser des modèles de pipeline intelligents pour créer des pipelines. ■ Importer des pipelines à partir de YAML et les exporter dans YAML. ■ Créer, mettre à jour et supprimer des tableaux de bord personnalisés. ■ Lire toutes les intégrations personnalisées. ■ Lire tous les points de terminaison et variables limités, mais pas afficher leurs valeurs.
Gérer les pipelines limités	■ 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. ■ Toutes les autorisations que vous pouvez effectuer avec l'autorisation Gérer les pipelines.
Gérer les intégrations personnalisées	■ Créer et mettre à jour des intégrations personnalisées. ■ Gérer la version et publier des intégrations personnalisées. ■ Supprimer et déconseiller des versions d'intégrations personnalisées. ■ Supprimer des intégrations personnalisées.
Exécuter les pipelines	■ Exécuter des pipelines. ■ Suspendre, reprendre et annuler les exécutions de pipeline. ■ Relancer des exécutions de pipeline. ■ Reprendre, relancer et déclencher manuellement un événement déclencheur Gerrit. ■ Approuver une opération d'utilisateur et effectuer des approbations par lot sur les opérations d'utilisateur.

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

Autorisation	Actions possibles
Exécuter les pipelines limités	■ Exécuter des pipelines. ■ Suspendre, reprendre, annuler et supprimer des exécutions de pipeline. ■ Relancer des exécutions de pipeline. ■ Synchroniser l'exécution d'un pipeline en cours d'exécution. ■ Forcer la suppression de l'exécution d'un pipeline en cours d'exécution. ■ Reprendre, relancer, supprimer et déclencher manuellement un événement déclencheur Gerrit. ■ Résoudre les éléments limités et poursuivre l'exécution du pipeline. ■ Changer le contexte utilisateur et poursuivre l'exécution du pipeline après l'approbation d'une tâche d'opération d'utilisateur. ■ Toutes les autorisations que vous pouvez effectuer avec l'autorisation Exécuter les pipelines.
Gérer les exécutions	■ Exécuter des pipelines. ■ Suspendre, reprendre, annuler et supprimer des exécutions de pipeline. ■ Relancer des exécutions de pipeline. ■ Reprendre, relancer, supprimer et déclencher manuellement un événement déclencheur Gerrit. ■ Toutes les autorisations que vous pouvez effectuer avec l'autorisation Exécuter les pipelines.

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 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 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	**Exécuter les pipelines**	
Deux autorisations	**Gérer les pipelines** et **Exécuter les pipelines**	

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

Nombre d'autorisations attribuées au rôle personnalisé	Exemples d'autorisations combinées	Utilisation de ce tableau de bord
Trois autorisations	**Gérer les pipelines**, **Exécuter les pipelines** et **Exécuter les pipelines limités**	
	Gérer les pipelines, **Gérer les intégrations personnalisées** et **Exécuter les pipelines limités**	Cette combinaison peut s'appliquer à un rôle de développeur de Code Stream, mais elle est limitée aux projets dont l'utilisateur est membre.
	Gérer les pipelines, **Gérer les intégrations personnalisées** et **Gérer les exécutions**	Cette combinaison peut s'appliquer à un administrateur de Code Stream, mais elle limitée aux projets dont l'utilisateur est membre.
	Gérer les pipelines, **Gérer les pipelines limités** et **Gérer les intégrations personnalisées**	Avec cette combinaison, un utilisateur dispose d'autorisations complètes et peut créer et supprimer tous les éléments dans 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 Code Stream ou un administrateur de projet doit reprendre l'exécution du pipeline.

Si vous disposez du rôle d'utilisateur

Vous pouvez accéder à 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

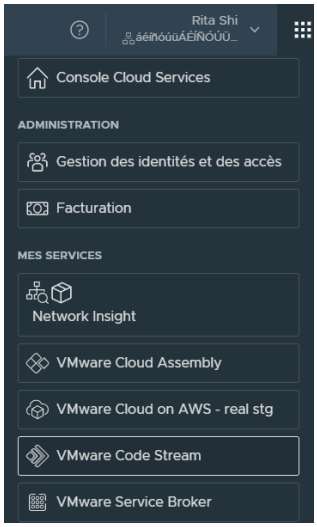
Vous pouvez exécuter des pipelines et effectuer des actions sur les tâches d'opération d'utilisateur. Vous pouvez également reprendre, suspendre et annuler les exécutions de pipeline, mais vous ne pouvez pas modifier les pipelines.

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 Code Stream

La zone Opérations d'utilisateur affiche les exécutions de pipeline requérant une approbation. L'approuveur requis peut approuver ou rejeter l'exécuter 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.

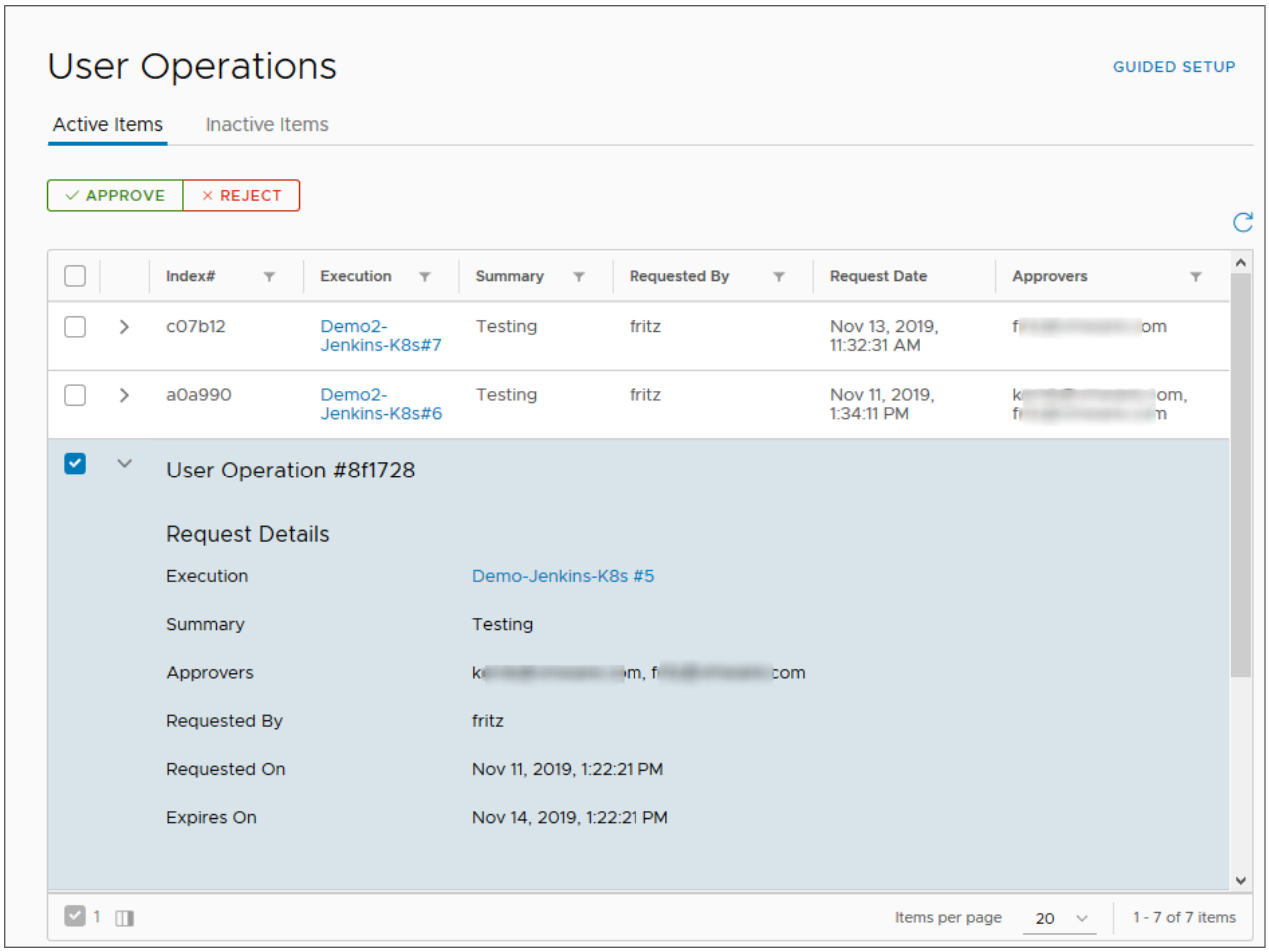
Pour déterminer s'il faut approuver une tâche de pipeline, l'approbateur requis doit avoir l'autorisation et l'expertise requises.

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.

Si vous activez l'envoi de notifications par e-mail, la tâche Opération d'utilisateur envoie uniquement des notifications aux approbateurs disposant d'adresses e-mail complètes et non aux noms d'approbateurs qui ne sont pas au format d'e-mail.

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

- 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.



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 examine la tâche, puis l'approuve ou la rejette. Si vous êtes un utilisateur figurant dans la liste d'approbateurs, 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 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 s'exécute lors des 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 qui inclut 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 CICD dans 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 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 publication de votre application. 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

Exemple d'étape	Exemples de ce que vous pouvez faire
Développement	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. Par exemple : ■ 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 Planification d'une build native d'intégration continue dans Code Stream avant d'utiliser le modèle de pipeline intelligent.
Test	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. Par exemple : ■ 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 Intégration de Code Stream à Jenkins. ■ 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 Intégration d'outils de génération, de test et de déploiement propres avec Code Stream. ■ Suivez les tendances d'un pipeline d'intégration continue (CI) en matière de post-traitement. Reportez-vous à la section Utilisation des tableaux de bord personnalisés pour suivre les indicateurs de performance clés d'un pipeline dans Code Stream.
Production	À une étape de production, vous pouvez intégrer un modèle de cloud à Cloud Assembly qui provisionne votre infrastructure, déployer votre logiciel sur un cluster Kubernetes, etc. Par exemple : ■ 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 Déploiement d'une application dans Code Stream vers un déploiement Bleu-vert. ■ Pour intégrer un modèle de cloud à votre pipeline, reportez-vous à la section Automatisation de la publication d'une application déployée à partir d'un modèle de cloud YAML dans Code Stream. Vous pouvez également ajouter une tâche de déploiement qui exécute un script pour déployer l'application. ■ Pour automatiser le déploiement de vos applications logicielles sur un cluster Kubernetes, reportez-vous à la section Automatisation de la publication d'une application dans Code Stream sur un cluster Kubernetes. ■ Pour intégrer du code dans votre pipeline et déployer votre image de build, reportez-vous à la section Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans Code Stream.

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 Code Stream en cas d'échec d'une tâche de pipeline.

Déclenchement de pipelines

Les pipelines peuvent se déclencher lorsque les développeurs placent leur code dans le référentiel ou examinent le code, ou lorsqu'ils identifient un nouvel artefact de build ou un artefact de build 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 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 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 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 Code Stream.

Ce chapitre contient les rubriques suivantes :

- Comment exécuter un pipeline et afficher les résultats
- Types de tâches disponibles dans Code Stream
- Utilisation des liaisons de variables dans les pipelines Code Stream
- Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline dans Code Stream

- Quelles variables et expressions peuvent être utilisées lors de la liaison de tâches de pipeline dans Code Stream
- Envoi de notifications relatives à un pipeline dans Code Stream
- Création d'un ticket JIRA dans Code Stream en cas d'échec d'une tâche de pipeline
- Restauration d'un déploiement dans 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 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 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` ou la commande `wget` peut demander à Jenkins d'exécuter un pipeline à la fin d'une génération Jenkins.

Pour plus d'informations sur l'utilisation de déclencheurs, reportez-vous à la section Chapitre 7 Déclenchement des pipelines dans 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 Code Stream.

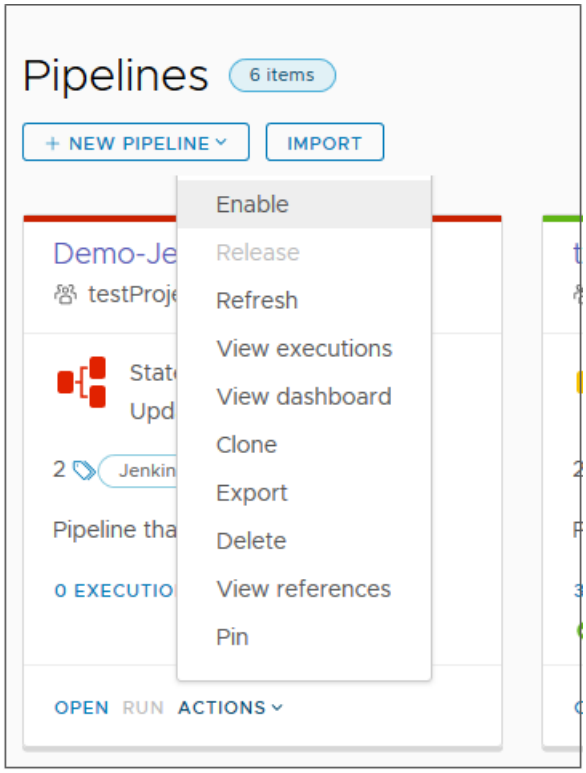
Procédure

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**.

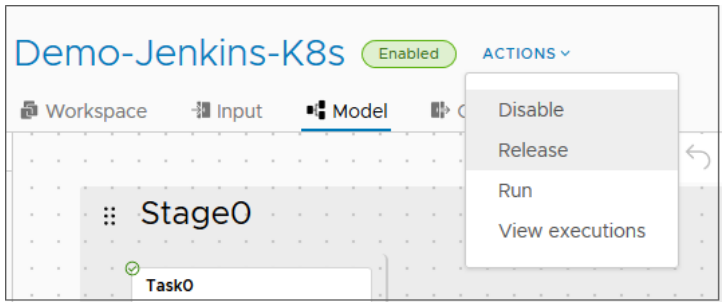
2 (Facultatif) Publiez votre pipeline.

Si vous souhaitez mettre à disposition votre pipeline en tant qu'élément du catalogue dans vRealize Automation Service Broker, vous devez le publier dans 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 Service Broker pour ajouter le pipeline en tant qu'élément du catalogue et l'exécuter. Reportez-vous à la section Ajouter des pipelines Code Stream au catalogue 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 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**.

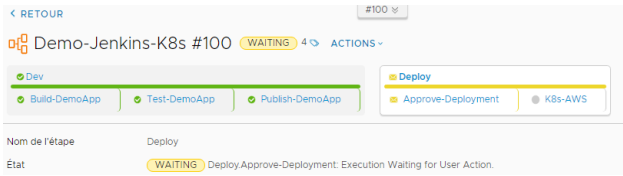
4 Pour afficher le pipeline au fil de son exécution, cliquez sur **Exécutions**.

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.

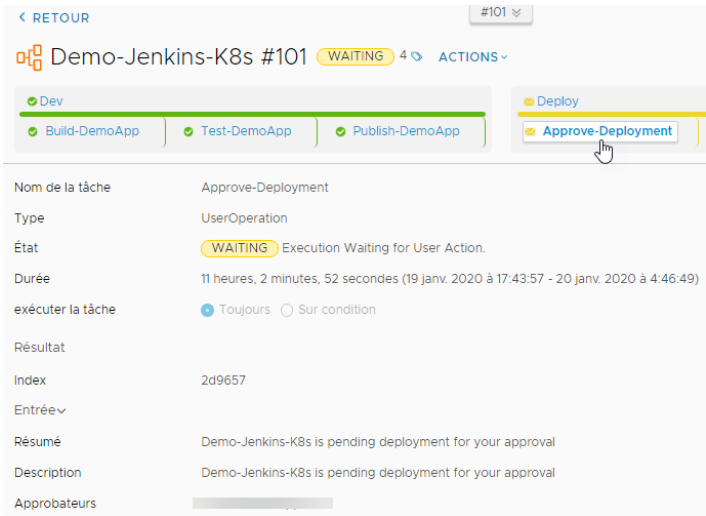


6 Pour afficher les détails de la tâche, cliquez sur celle-ci.

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 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. Cliquez ensuite sur l'onglet en haut et sélectionnez l'exécution du pipeline.

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 Code Stream pour automatiser votre cycle de publication logicielle, reportez-vous à la section Chapitre 5 Didacticiels d'utilisation de Code Stream.

Types de tâches disponibles dans Code Stream

Lorsque vous configurez votre pipeline, vous ajoutez des types de tâches spécifiques que le pipeline exécute pour les actions dont vous avez besoin. Chaque type de tâche s'intègre à une autre application et active votre pipeline au cours de la construction, des tests et de la livraison de vos applications.

Pour exécuter votre pipeline, que vous deviez extraire des artefacts d'un référentiel pour le déploiement, exécuter un script distant ou demander l'approbation d'une opération utilisateur à un membre de l'équipe, Code Stream a le type de tâche qu'il vous faut !

Code Stream prend en charge l'annulation d'une exécution de pipeline sur divers types de tâches. Lorsque vous cliquez sur **Annuler** sur l'exécution d'un pipeline, la tâche, l'étape ou l'intégralité du pipeline passe à l'état d'annulation et annule l'exécution du pipeline.

Code Stream vous permet d'annuler l'exécution du pipeline sur une tâche, une étape ou l'intégralité du pipeline lors de l'utilisation de ces tâches :

- Jenkins
- SSH
- PowerShell
- Opération de l'utilisateur

- Pipeline
- Modèle de cloud
- vRO
- POLL

Code Stream ne propage pas le comportement d'annulation aux systèmes tiers pour ces tâches : CI, Intégration personnalisée ou Kubernetes. Code Stream marque la tâche comme annulée et cesse immédiatement d'extraire l'état sans attendre la fin de la tâche. La tâche peut se terminer ou échouer sur le système tiers, mais cesse immédiatement de s'exécuter dans Code Stream lorsque vous cliquez sur **Annuler**.

Avant d'utiliser une 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	Fonctionnement	Exemples et détails
Opération de l'utilisateur	Active une tâche d'opération d'utilisateur qui contrôle le moment où un pipeline s'exécute et doit s'arrêter pour approbation.	Reportez-vous à la section Comment exécuter un pipeline et afficher les résultats. et Gestion de l'accès et des approbations utilisateur dans Code Stream.
Condition	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 Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline dans Code Stream.

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

Type de tâche	Fonctionnement	Exemples et détails
Modèle de cloud	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.	Reportez-vous à la section Automatisation de la publication d'une application déployée à partir d'un modèle de cloud YAML dans Code Stream. Les paramètres du modèle de cloud s'affichent lorsque vous sélectionnez **Créer** ou **Mettre à jour**, puis sélectionnez **Modèle de cloud** et **Version**. Vous pouvez ajouter ces éléments, qui adaptent les liaisons de variables aux zones de texte d'entrée dans la tâche de modèle de cloud : ■ Entier ■ Chaîne d'énumération ■ Valeur booléenne ■ Variable de tableau Lorsque vous utilisez la liaison de variable dans l'entrée, vous devez connaître ces exceptions. 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. Le paramètre du modèle de cloud apparaît dans la tâche de modèle de cloud lorsqu'un modèle de cloud de Cloud Assembly inclut des variables d'entrée. Par exemple, si un modèle de cloud dispose d'un type d'entrée `Integer`, vous pouvez entrer l'entier directement ou sous la forme d'une variable à l'aide d'une liaison de variables.
CI	La tâche CI 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. La tâche CI affiche 100 lignes du journal en tant que sortie et affiche 500 lignes lorsque vous téléchargez les journaux. Les tâches CI nécessitent les ports éphémères 32768 à 61000.	Reportez-vous à la section Planification d'une build native CICD dans Code Stream avant d'utiliser le modèle de pipeline intelligent.
Personnalisé	La tâche personnalisée intègre Code Stream à vos propres outils de générations, de tests et de déploiement.	Reportez-vous à la section Intégration d'outils de génération, de test et de déploiement propres avec Code Stream.

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

Type de tâche	Fonctionnement	Exemples et détails
Kubernetes	Automatisez le déploiement de vos applications logicielles sur des clusters Kubernetes sur AWS.	Reportez-vous à la section Automatisation de la publication d'une application dans Code Stream sur un cluster Kubernetes.
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 **Exécutions**, entrez **`imbriqué`** 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 Intégration de Code Stream à Jenkins.

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

Type de tâche	Effets	Exemples et détails
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.	Pour connaître les versions de Team Foundation Server prises en charge par Code Stream, consultez Présentation des points de terminaison dans Code Stream .
vRO	Étend la capacité de Code Stream en exécutant des workflows prédéfinis ou personnalisés dans vRealize Orchestrator. Code Stream prend en charge l'authentification de base et l'authentification basée sur des jetons pour vRealize Orchestrator. Code Stream utilise le jeton d'API pour authentifier et valider le cluster vRealize Orchestrator. Avec l'authentification par jeton, Code Stream prend en charge les points de terminaison vRealize Orchestrator qui utilisent un proxy d'extensibilité de cloud. Par conséquent, dans Code Stream, vous pouvez déclencher des workflows avec un point de terminaison vRealize Orchestrator qui utilise le proxy d'extensibilité de cloud.	Reportez-vous à la section Intégration de Code Stream à vRealize Orchestrator.

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

Type de tâche	Effets	Exemples et détails
REST	Intègre 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 Utilisation d'une REST API pour intégrer Code Stream à d'autres applications.
Interrogation	Appelle une REST API et l'interroge jusqu'à ce que la tâche de pipeline réponde aux critères de sortie et se termine. Un administrateur Code Stream peut définir le nombre d'interrogations sur un maximum de 10 000. L'intervalle d'interrogation doit être supérieur ou égal à 60 secondes. Lorsque vous cochez la case **Continuer en cas d'échec**, si le nombre ou l'intervalle dépasse ces valeurs, la tâche d'interrogation continue à s'exécuter. `POLL Iteration Count` : figure dans l'exécution du pipeline et affiche le nombre de fois que la tâche POLL a demandé une réponse à partir de l'URL. Par exemple, si l'entrée POLL est 65 et que la demande POLL est exécutée 4 fois, le nombre d'itérations dans la sortie d'exécution du pipeline est 4 (sur 65).	Reportez-vous à la section Utilisation d'une REST API pour intégrer Code Stream à d'autres applications.

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

Type de tâche	Fonctionnement	Exemples et détails
PowerShell	Avec la tâche PowerShell, Code Stream peut 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. Le script peut être distant ou défini par l'utilisateur. Il peut se connecter via HTTP ou HTTPS, et peut utiliser TLS. L'hôte Windows doit disposer du service `winrm` configuré, et `winrm` doit disposer de `MaxShellsPerUser` et `MaxMemoryPerShellMB` configurés. Pour exécuter une tâche PowerShell, vous devez disposer d'une session active sur l'hôte Windows distant. **Longueur de la ligne de commande PowerShell** Si vous entrez une commande PowerShell en base 64, sachez que vous devez calculer la longueur de la commande globale. Le pipeline Code Stream encode et encapsule une commande PowerShell en base 64 dans une autre commande, ce qui augmente la longueur globale de la commande. La longueur maximale autorisée pour une commande PowerShell `winrm` est de 8 192 octets. La limite de longueur de la commande est inférieure pour la tâche PowerShell lorsqu'elle est codée et encapsulée. Vous devez donc calculer la longueur de la commande avant d'entrer la commande PowerShell. La limite de longueur de la commande pour la tâche PowerShell Code Stream dépend de la longueur codée en base 64 de la commande d'origine. La longueur de la commande est calculée de la manière suivante. `3 * (length of original command / 4)) - (numberOfPaddingCharacters) + 77 (Length of Write-output command)` La longueur de la commande pour Code Stream doit être inférieure à la limite maximale de 8 192.	Lorsque vous configurez `MaxShellsPerUser` et `MaxMemoryPerShellMB` : ■ La valeur acceptable pour `MaxShellsPerUser` est de `500` pour 50 pipelines simultanés, avec 5 tâches PowerShell par pipeline. Pour définir la valeur, exécutez : `winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'` ■ La valeur de mémoire acceptable pour `MaxMemoryPerShellMB` est de `2 048`. Pour définir la valeur, exécutez : `winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'` Le script écrit la sortie dans un fichier de réponse qu'un autre pipeline peut utiliser.
SSH	La tâche SSH permet à la 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. 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.	Le script peut être distant ou défini par l'utilisateur. Par exemple, un script peut ressembler à ce qui suit : `message="Hello World" echo $message` Le script écrit la sortie dans un fichier de réponse qu'un autre pipeline peut utiliser.

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

Type de tâche	Fonctionnement	Exemples et détails
	Le service SSH doit être configuré sur l'hôte Linux et la configuration SSHD de `MaxSessions` doit être définie sur `50`. Si vous exécutez plusieurs tâches SSH simultanément, augmentez `MaxSessions` et `MaxOpenSessions` sur l'hôte SSH. N'utilisez pas votre instance de vRealize Automation en tant qu'hôte SSH si vous devez modifier les paramètres de configuration MaxSessions et MaxOpenSessions.	

Utilisation des liaisons de variables dans les pipelines 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 Code Stream. La façon dont vous modifiez les variables dans Code Stream dépend du codage des propriétés de la variable dans le modèle de cloud.

Si vous devez 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 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 Cloud Assembly.

- Vous devez avoir le même rôle que la personne qui a créé le déploiement du modèle de cloud dans 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 Code Stream et modélise le pipeline, le développeur doit également disposer du même rôle dans Cloud Assembly que la personne qui a déployé le modèle de cloud. Par exemple, le rôle requis peut être Administrateur de Cloud Assembly.

- Seule la personne qui modélise le pipeline peut créer le pipeline et créer le déploiement, car elle en a l'autorisation.

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 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 le jeton d'API dans la tâche de modèle de cloud, il crée les informations d'identification requises par le 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 définies se trouvent 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 Code Stream et y ajouter des liaisons dollar, procédez comme suit.

1 Dans 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 fournit des informations d'identification pour le pipeline. Pour créer une variable qui chiffre le jeton d'API dans la tâche de modèle de cloud, cliquez sur **Créer une variable**.

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 Cloud Assembly. Par exemple :

   a Définissez `flavor` de façon qu'il utilise une propriété de type `array`. 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`.

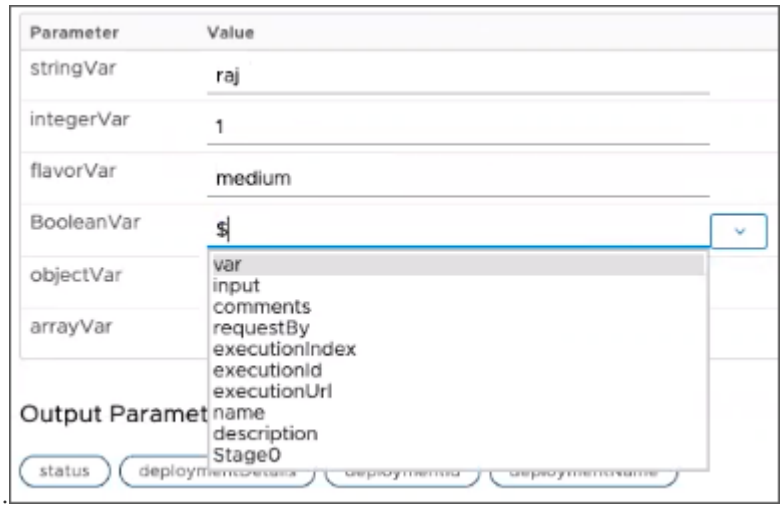
   e Notez que la page Entrées de déploiement inclut toutes les variables définies dans votre code de modèle de cloud et qui apparaîtront comme définies dans le code du modèle de cloud. Par exemple : `Integer Var`, `String Var`, `Flavor Var`, `Boolean Var`, `Object Var` et `Array Var`. `String Var` et `Flavor Var` sont des valeurs de chaîne et `Boolean Var` est une case à cocher.

   f Cliquez sur **Déployer**.

7 Dans 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, qui activent les liaisons Code Stream à l'aide de variables à signe dollar. De légères différences existent entre l'interface utilisateur de la tâche du modèle de cloud Code Stream et l'interface utilisateur du modèle de cloud Cloud Assembly. En fonction du codage d'un modèle de cloud dans Cloud Assembly, l'entrée de valeurs dans la tâche du modèle de cloud dans Code Stream peut ne pas être autorisée.

   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.



   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 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 Code Stream les transmette au pipeline. Ensuite, lorsque le pipeline s'exécute, un utilisateur doit entrer la valeur du paramètre d'entrée. Lorsque vous ajoutez des paramètres de sortie à votre pipeline, les tâches du pipeline peuvent utiliser la valeur de sortie d'une tâche. 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 :

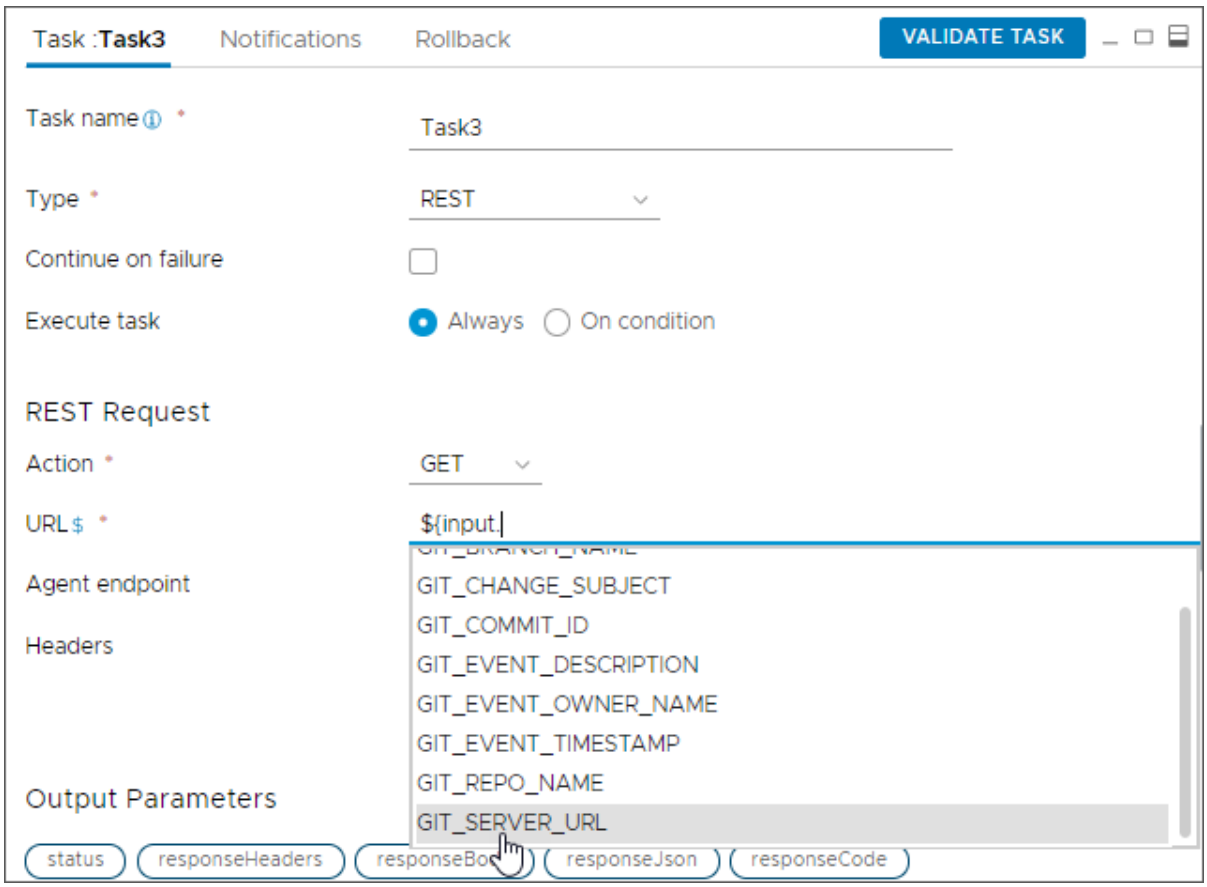
1 Dans votre pipeline, cliquez sur l'onglet **Entrée**.

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 devez utiliser une valeur par défaut pour l'URL du serveur Git, modifiez 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**.



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**.

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.

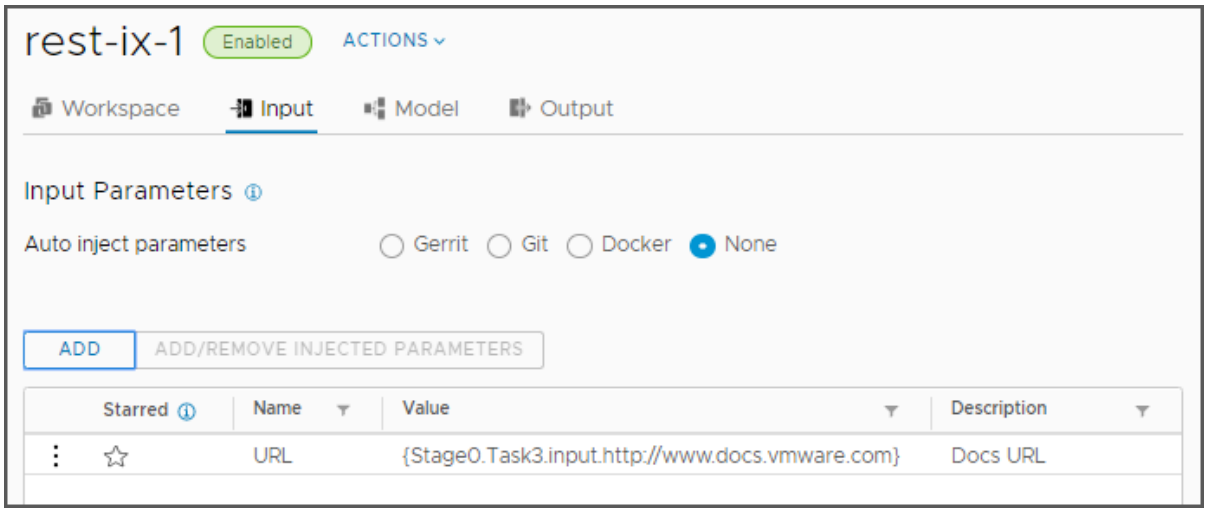
Liaison de deux tâches de pipeline par la création de paramètres d'entrée et de sortie

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. Cet exemple montre comment procéder.

Supposons que vous ayez besoin de votre pipeline pour appeler une URL dans une tâche REST et pour envoyer une réponse. Pour appeler l'URL et afficher la réponse, vous incluez les paramètres d'entrée et de sortie dans votre tâche REST. Vous avez également besoin d'un utilisateur pouvant approuver la tâche, et inclure une tâche Opérations d'utilisateur pour un autre utilisateur qui peut l'approuver 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**.



2 Laissez les **Paramètres d'insertion automatique** sur **Aucun**.

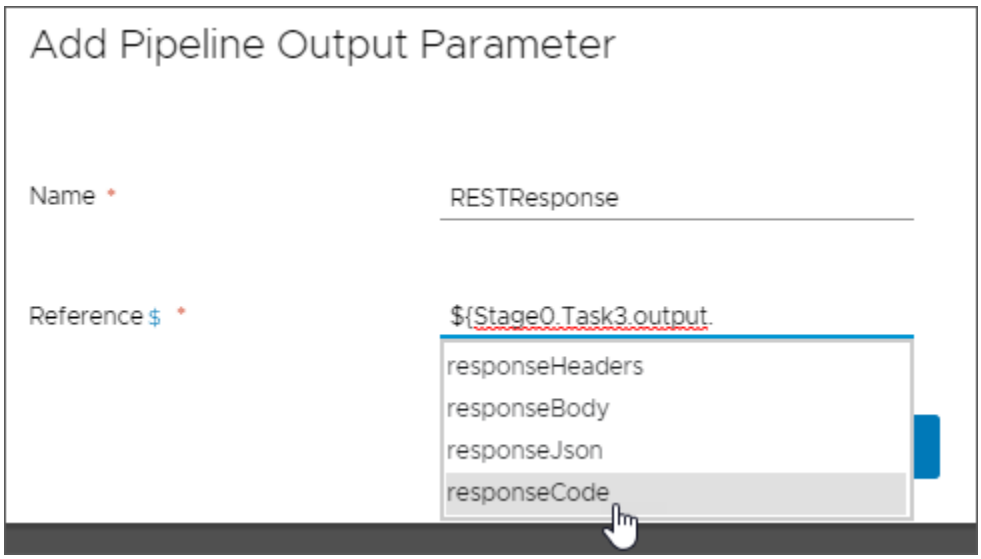
3 Cliquez sur **Ajouter**, entrez le nom, la valeur et la description du paramètre, puis cliquez sur **OK**. Par exemple :

   a Entrez un nom d'URL.

   b Entrez la valeur : `{Stage0.Task3.input.http://www.docs.vmware.com}`

   c Entrez une description.

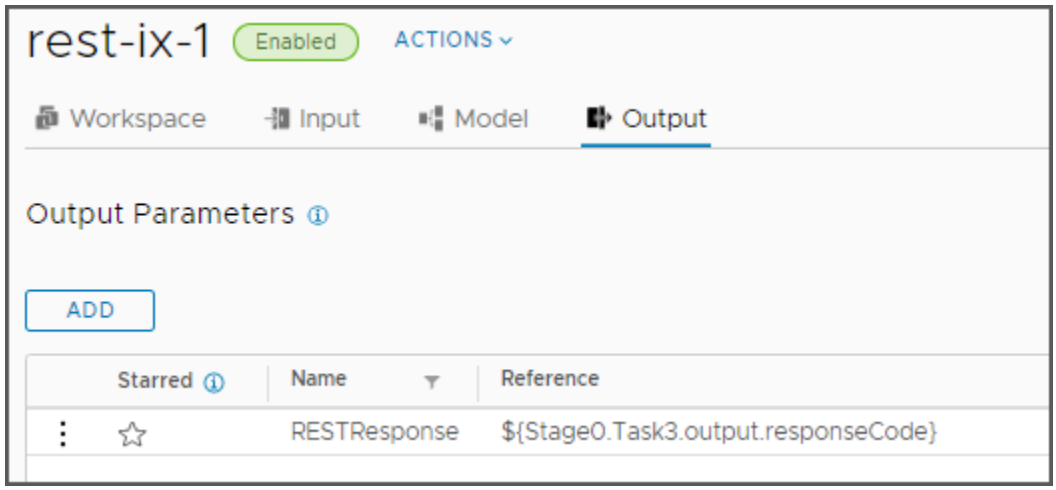
4 Cliquez sur l'onglet **Sortie**, cliquez sur **Ajouter**, puis entrez le nom et le mappage du paramètre de sortie.



a Entrez un nom de paramètre de sortie unique.

b Cliquez dans la zone de **Référence** et entrez `$`.

c 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**.

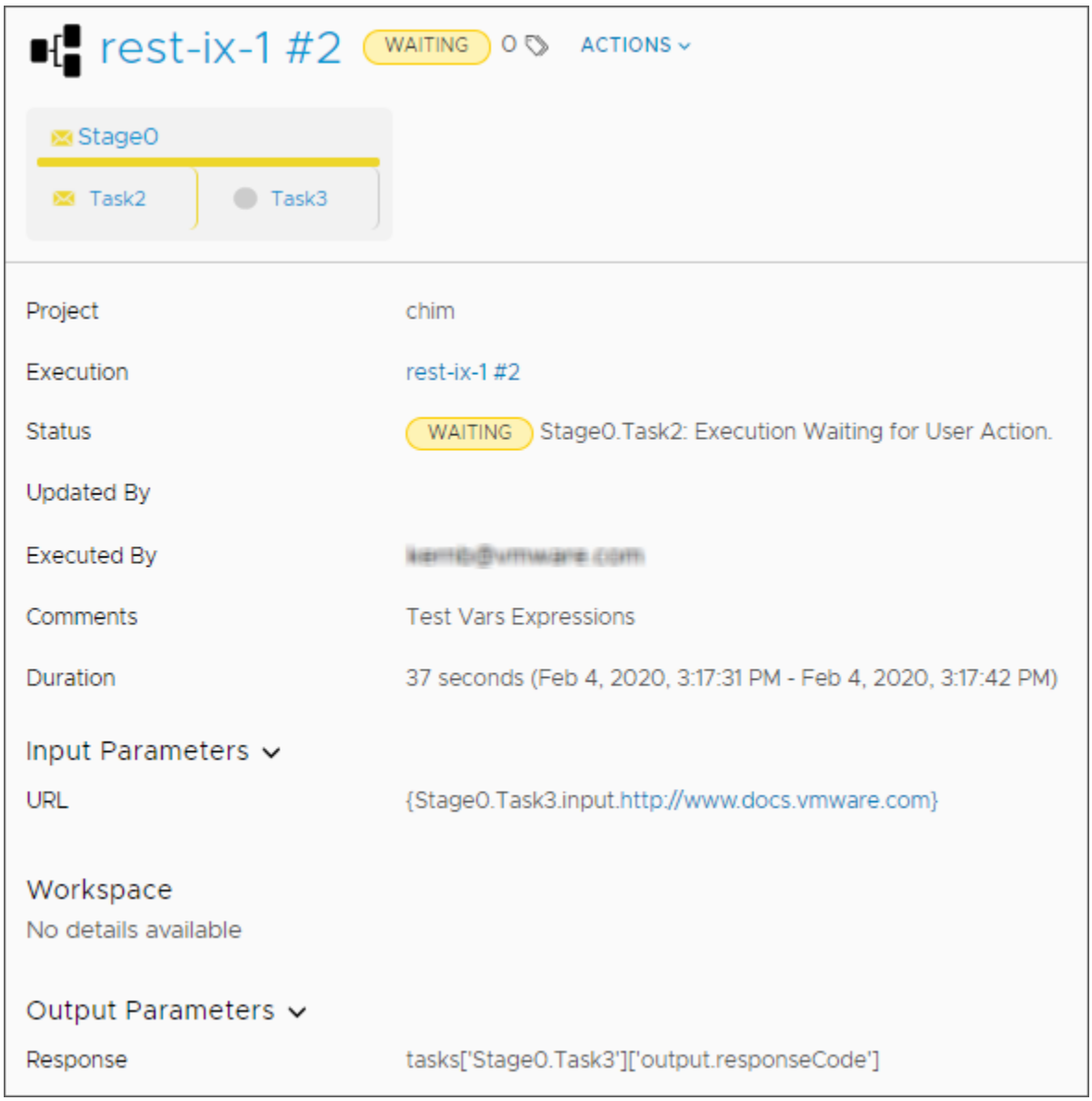


5 Enregistrez votre pipeline.

6 Dans le menu **Actions**, cliquez sur **Exécuter**.

7 Cliquez sur **Actions > Afficher les exécutions**.

8 Cliquez sur l'exécution du pipeline et examinez les paramètres d'entrée et de sortie que vous avez définis.

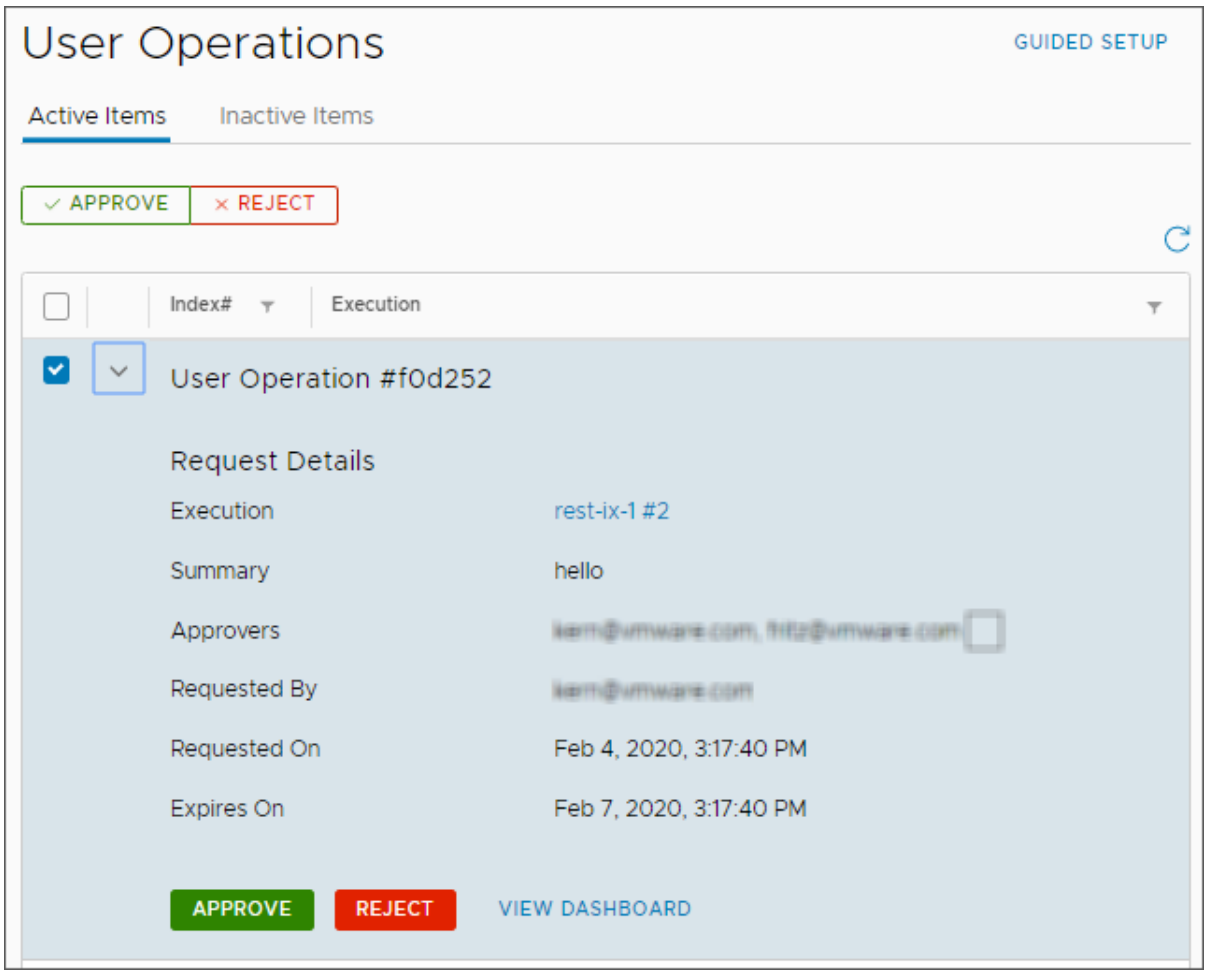


9 Pour approuver le pipeline, cliquez sur **Opérations de l'utilisateur** et affichez la liste des approbations dans l'onglet **Éléments actifs**. Sinon, restez dans Exécutions, cliquez sur la tâche, puis sur **Approuver**.

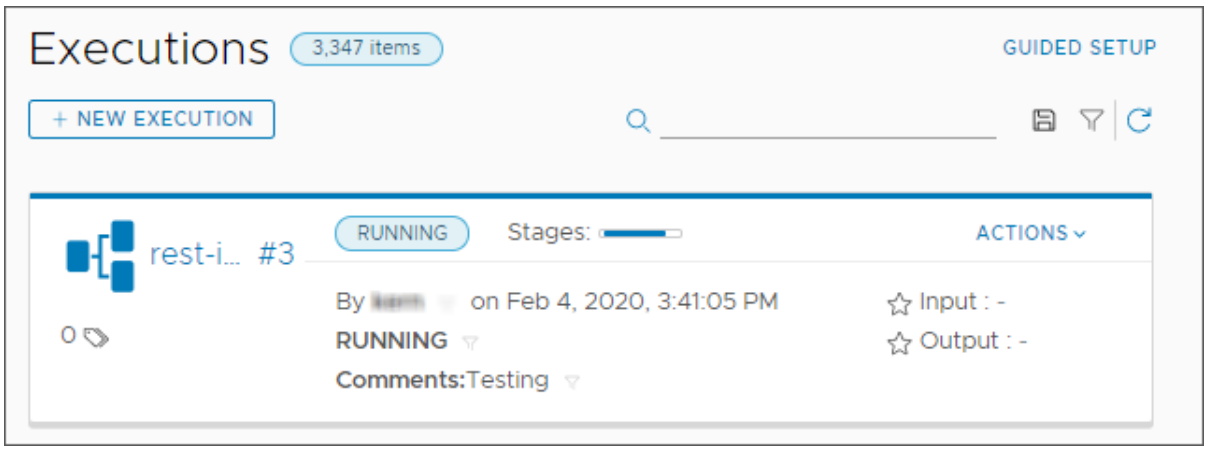
10 Pour activer les boutons **Approuver** et **Refuser**, cochez la case en regard de l'exécution.

11 Pour afficher les détails, développez la flèche déroulante.

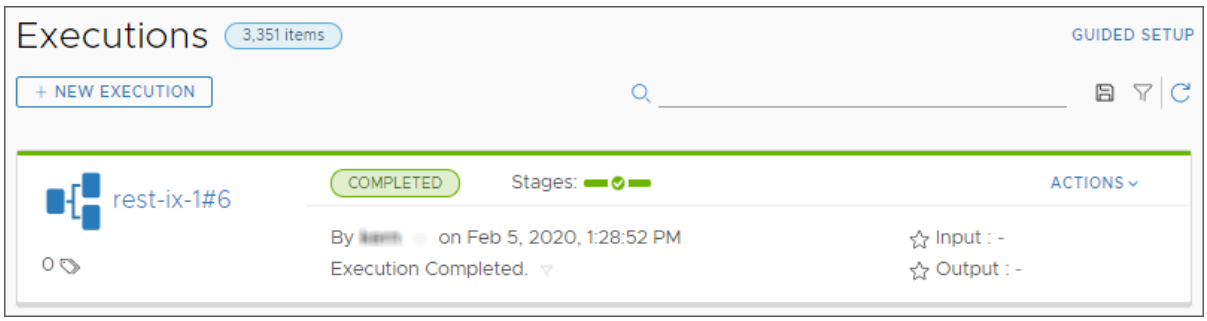
12 Pour approuver la tâche, cliquez sur **APPROUVER**, entrez un motif et cliquez sur **OK**.



13 Cliquez sur **Exécutions** et observez la suite de l'exécution du pipeline.



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 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 Code Stream.

Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline dans 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 du résultat de la tâche, vous utilisez la tâche Condition.

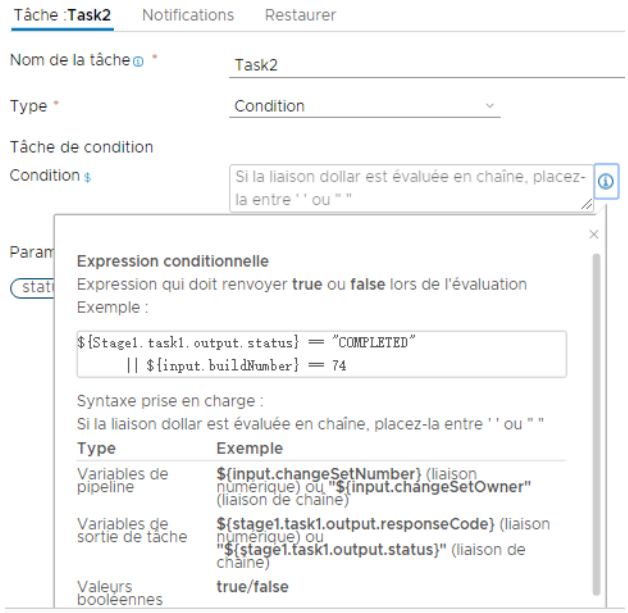
Vous utilisez la tâche **Condition** comme point de décision dans votre pipeline. En utilisant la tâche Condition avec une expression de condition que vous indiquez, vous pouvez évaluer toutes les propriétés de votre pipeline, de vos étapes et de vos tâches.

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

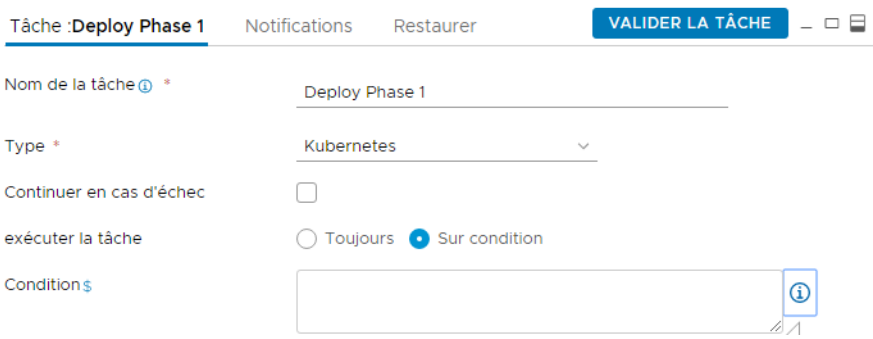
- Une condition true permet au pipeline de continuer à s'exécuter.
- Une condition false arrête le pipeline.

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 Code Stream.

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

Tâche Condition	Domaine d'influence	Fonctionnement
Tâche Condition	Pipeline	La tâche **Condition** détermine si le pipeline s'exécute ou s'arrête à ce stade, selon que la sortie de la tâche est true ou false.
Expression de condition	Sortie de la tâche Condition	Lorsque le pipeline s'exécute, l'expression de condition que vous incluez dans la tâche **Condition** 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 `Terminé` ou utiliser un numéro de build `74`. L'expression de condition s'affiche dans l'onglet Tâche, dans la tâche Condition. 

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



Dans d'autres types de tâches, **Sur condition** détermine si la tâche actuelle s'exécute, plutôt que les tâches successives, en fonction de l'évaluation de son expression de condition préalable 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. Le paramètre **Sur condition** s'affiche dans l'onglet Tâche avec sa propre expression de condition.

Cet exemple utilise la tâche Condition.

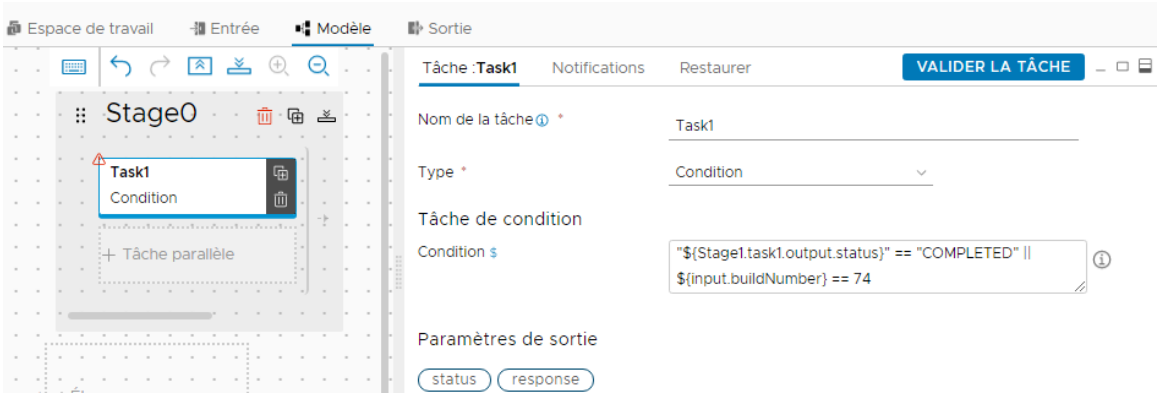
Conditions préalables

- 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

Si vous restaurez un déploiement de pipeline, vous pouvez également utiliser la tâche Condition. Par exemple, dans un pipeline de restauration, la tâche Condition permet à 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 Code Stream.

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

Avec des variables et des expressions, vous pouvez utiliser des paramètres d'entrée et des paramètres 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.

Les pipelines peuvent exécuter des solutions de livraison de logiciels simples ou complexes

Lorsque vous liez des tâches de pipeline ensemble, vous pouvez inclure des expressions par défaut et complexes. Par conséquent, votre pipeline peut 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 résoudre 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 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>}`.

Pour déterminer le comportement d'une tâche de pipeline, dans chaque expression, `SCOPE` est le contexte que Code Stream utilise. 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 :
	`${input.input1}`
		Nom de la propriété d'entrée	Pour faire référence à la propriété d'entrée d'un pipeline dans une tâche, utilisez ce format :
			`tasks:`
			`  mytask:`
			`  type: REST`
			`  input:`
			`  url: ${input.url}`
			`  action: get`
			
			`input:`
			`  url: https://www.vmware.com`

output	Propriétés de sortie d'un pipeline :
	`${output.output1}`
		Nom de la propriété de sortie	Pour faire référence à une propriété de sortie pour l'envoi d'une notification, utilisez ce format :
			`notifications:`
			`  email:`
			`  - endpoint: MyEmailEndpoint`
			`  subject: "Deployment Successful"`
			`  event: COMPLETED`
			`  to:`
			`  - user@example.org`
			`  body: |`
			`  Pipeline deployed the service successfully. Refer ${output.serviceURL}`

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	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 :
			`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.input.job} started for commit id ${input.COMMITID}.`

task output	Sortie d'une tâche :
	`${MY_STAGE.MY_TASK.output.SOMETHING}`
		Indique la sortie d'une tâche dans une tâche suivante	Pour faire référence à la sortie de la tâche de pipeline 1 dans la tâche 2, utilisez ce format :
			`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.example.org/api/activity`
			`  payload: ${MY_STAGE.task1.output.responseBody}`

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 :
	`${var.myVariable}`
		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 :
			`---`
			`project: MyProject`
			`kind: ENDPOINT`
			`name: MyJenkinsServer`
			`type: jenkins`
			`properties:`
			`url: https://jenkins.example.com`
			`username: jenkinsUser`
			`password: ${var.jenkinsPassword}`

var	Variable :
	`${var.myVariable}`
		Faire référence à une variable dans un pipeline	Pour faire référence à une variable dans une URL de pipeline, utilisez ce format :
			`tasks:`
			`task1:`
			`type: REST`
			`input:`
			`action: get`
			`url: ${var.MY_SERVER_URL}`

task status	État d'une tâche :
	`${MY_STAGE.MY_TASK.status}`
	`${MY_STAGE.MY_TASK.statusMessage}`		

stage status	État d'une étape :
	`${MY_STAGE.status}`
	`${MY_STAGE.statusMessage}`		

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
`${comments}`	Commentaires fournis lors de la demande d'exécution du pipeline.
`${duration}`	Durée de l'exécution du pipeline.
`${endTime}`	Heure de fin de l'exécution du pipeline en UTC, si conclue.
`${executedOn}`	Identique à l'heure de début, l'heure de début de l'exécution du pipeline en UTC.
`${executionId}`	ID de l'exécution du pipeline.
`${executionUrl}`	URL qui permet d'accéder à l'exécution du pipeline dans l'interface utilisateur.
`${name}`	Nom du pipeline.
`${requestBy}`	Nom de l'utilisateur qui a demandé l'exécution.
`${stageName}`	Nom de l'étape actuelle, lorsqu'elle est utilisée dans l'étendue d'une étape.
`${startTime}`	Heure de début de l'exécution du pipeline en UTC.
`${status}`	État de l'exécution.
`${statusMessage}`	Message d'état de l'exécution du pipeline.
`${taskName}`	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 tâches de pipeline

Vous pouvez utiliser des expressions avec n'importe quelle tâche de pipeline prise en charge. Ces exemples vous montrent comment définir `SCOPE` et `KEY`, et confirmer 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 tâches disponibles, consultez Types de tâches disponibles dans Code Stream.

Tableau 3-9. Assemblage de tâches

Tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
Opération de l'utilisateur			

	Input	`summary` : résumé de la demande d'opération d'utilisateur
		`description` : description de la demande d'opération d'utilisateur
		`approvers` : 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
		`approverGroups` : liste des adresses des groupes d'approbateurs pour la plate-forme et l'identité
		`sendemail` : envoie éventuellement une notification par e-mail lors de la demande ou de la réponse lorsqu'elle est définie sur true
		`expirationInDays` : nombre de jours qui représente le délai d'expiration de la demande
			`${MY_STAGE.MY_TASK.input.summary}`
			`${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}`

	Output	`index` : chaîne hexadécimale à 6 chiffres qui représente la demande
		`respondedBy` : nom du compte de la personne qui a approuvé/refusé l'opération d'utilisateur
		`respondedByEmail` : adresse e-mail de la personne qui a répondu
		`comments` : commentaires fournis lors de la réponse
			`${MY_STAGE.MY_TASK.output.index}`
			`${MY_STAGE.MY_TASK.output.respondedBy}`
			`${MY_STAGE.MY_TASK.output.respondedByEmail}`
			`${MY_STAGE.MY_TASK.output.comments}`

Condition			
	Input	`condition` : 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	`${MY_STAGE.MY_TASK.input.condition}`
	Output	`result` : résultat lors de l'évaluation	`${MY_STAGE.MY_TASK.output.response}`

Tableau 3-10. Tâches de pipeline

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	`${MY_STAGE.MY_TASK.input.name}` `${MY_STAGE.MY_TASK.input.inputProperties}` # Faire référence à toutes les propriétés `${MY_STAGE.MY_TASK.input.inputProperties.input1}` # Faire référence à la valeur de input1

	Output	`executionStatus` : état de l'exécution du pipeline
		`executionIndex` : index de l'exécution du pipeline
		`outputProperties` : propriétés de sortie de l'exécution du pipeline
			`${MY_STAGE.MY_TASK.output.executionStatus}`
			`${MY_STAGE.MY_TASK.output.executionIndex}`
			`${MY_STAGE.MY_TASK.output.outputProperties}` # Faire référence à toutes les propriétés
			`${MY_STAGE.MY_TASK.output.outputProperties.output1}` # Faire référence à la valeur d'output1

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

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
			`${MY_STAGE.MY_TASK.input.steps}`
			`${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

	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
			`${MY_STAGE.MY_TASK.output.exports}` # Faire référence à toutes les exportations
			`${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.processResponse}`
			`${MY_STAGE.MY_TASK.output.processResponse[0].result}` # Résultat de la configuration du premier processus

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

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
			`${MY_STAGE.MY_TASK.input.name}`
			`${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

	Output	`properties` : propriétés de sortie de la réponse d'intégration personnalisée	`${MY_STAGE.MY_TASK.output.properties}` # Faire référence à toutes les propriétés `${MY_STAGE.MY_TASK.output.properties.property1}` # Faire référence à la valeur de property1

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

Tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
Modèle de cloud			
	Input	`action` : un des éléments **createDeployment**, **updateDeployment**, **deleteDeployment**, **rollbackDeployment** `blueprintInputParams` : utilisé pour les actions **Créer un déploiement** et **Mettre à jour le déploiement** `allowDestroy` : des machines peuvent être détruites lors du processus de déploiement de la mise à jour. **CREATE_DEPLOYMENT** ■ `blueprintName` : nom du modèle de cloud ■ `blueprintVersion` : version du modèle de cloud OU ■ `fileUrl` : URL du modèle de cloud distant YAML, après la sélection d'un serveur GIT. **UPDATE_DEPLOYMENT** Une de ces combinaisons : ■ `blueprintName` : nom du modèle de cloud ■ `blueprintVersion` : version du modèle de cloud OU ■ `fileUrl` : URL du modèle de cloud distant YAML, après la sélection d'un serveur GIT. ------ ■ `deploymentId` : ID du déploiement OU ■ `deploymentName`: nom du déploiement ------ **DELETE_DEPLOYMENT** ■ `deploymentId` : ID du déploiement	

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

Tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
		OU ■ `deploymentName`: nom du déploiement **ROLLBACK_DEPLOYMENT** Une de ces combinaisons : ■ `deploymentId` : ID du déploiement OU ■ `deploymentName`: nom du déploiement `------` ■ `blueprintName` : nom du modèle de cloud ■ `rollbackVersion` : version à restaurer	
	Output		Paramètres pouvant être liés à d'autres tâches ou à la sortie d'un pipeline : ■ Le nom du déploiement est accessible en tant que **${Stage0.Task0.output.deploymentName}** ■ L'ID de déploiement est accessible en tant que **${Stage0.Task0.output.deploymentId}** ■ Les détails du déploiement constituent un objet complexe et les détails internes sont accessibles à l'aide des résultats JSON. 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 **$** est : **${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}** De même, pour le type, la liaison **$** est : **${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}** Dans l'interface utilisateur de Code Stream, vous pouvez obtenir les liaisons **$** pour n'importe quelle propriété. 1 Dans la zone de la propriété de sortie de la tâche, cliquez sur **EXAMINER LE RÉSULTAT JSON**. 2 Pour trouver la liaison **$**, entrez n'importe quelle propriété. 3 Cliquez sur l'icône de recherche, qui affiche la liaison **$** correspondante.

Exemple de sortie JSON :

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

Tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
Kubernetes			

	Input	`action` : une des opérations **GET**, **CREATE**, **APPLY**, **DELETE**, **ROLLBACK**
		■ `timeout` : délai d'expiration global d'une action
		■ `filterByLabel` : étiquette supplémentaire pour filtrer l'action **`GET`** avec K8S labelSelector
		GET, CREATE, DELETE, APPLY
		■ `yaml` : YAML en ligne à traiter et à envoyer à Kubernetes
		■ `parameters` : paire KEY, VALUE - Remplacer **`$$KEY`** par **`VALUE`** dans la zone d'entrée YAML en ligne
		■ `filePath` : chemin d'accès relatif à partir du point de terminaison Git SCM, s'il est fourni, à partir duquel extraire le YAML
		■ `scmConstants` : paire KEY, VALUE - Remplacer **`$${KEY}`** par **`VALUE`** dans le YAML extrait sur SCM.
		■ `continueOnConflict` : lorsqu'elle est définie sur true, si une ressource est déjà présente, la tâche se poursuit.
		ROLLBACK
		■ `resourceType` : type de ressource à restaurer
		■ `resourceName` : nom de la ressource à restaurer
		■ `namespace` : espace de noms dans lequel la restauration doit être effectuée
		■ `revision` : révision à restaurer
			`${MY_STAGE.MY_TASK.input.action}` #Détermine l'action à effectuer.
			`${MY_STAGE.MY_TASK.input.timeout}`
			`${MY_STAGE.MY_TASK.input.filterByLabel}`
			`${MY_STAGE.MY_TASK.input.yaml}`
			`${MY_STAGE.MY_TASK.input.parameters}`
			`${MY_STAGE.MY_TASK.input.filePath}`
			`${MY_STAGE.MY_TASK.input.scmConstants}`
			`${MY_STAGE.MY_TASK.input.continueOnConflict}`
			`${MY_STAGE.MY_TASK.input.resourceType}`
			`${MY_STAGE.MY_TASK.input.resourceName}`
			`${MY_STAGE.MY_TASK.input.namespace}`
			`${MY_STAGE.MY_TASK.input.revision}`

	Output	`response` : capture la réponse complète `response.<RESOURCE>` : ressource correspond à configMaps, deployments, endpoints, ingresses, jobs, namespaces, pods, replicaSets, replicationControllers, secrets, services, statefulSets, nodes, loadBalancers. `response.<RESOURCE>.<KEY>` : la clé correspond à apiVersion, kind, metadata, spec	`${MY_STAGE.MY_TASK.output.response}` `${MY_STAGE.MY_TASK.output.response.}`

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

Tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
Bamboo			

	Input	`plan` : nom du plan
		`planKey` : clé du plan
		`variables` : variables à transmettre au plan
		`parameters` : paramètres à transmettre au plan
			`${MY_STAGE.MY_TASK.input.plan}`
			`${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

	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
			`${MY_STAGE.MY_TASK.output.resultUrl}`
			`${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}`

Jenkins			
	Input	`job` : nom du travail Jenkins `parameters` : paramètres à transmettre à la tâche	`${MY_STAGE.MY_TASK.input.job}` `${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

	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
			`${MY_STAGE.MY_TASK.output.job}`
			`${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.jacocoRespose}` # Faire référence aux résultats JaCoCo
			`${MY_STAGE.MY_TASK.output.jobUrl}`

TFS			

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

Tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche

	Input	`projectCollection` : collection de projets depuis TFS
		`teamProject` : projet sélectionné dans la collection disponible
		`buildDefinitionId` : ID de définition de build à exécuter
			`${MY_STAGE.MY_TASK.input.projectCollection}`
			`${MY_STAGE.MY_TASK.input.teamProject}`
			`${MY_STAGE.MY_TASK.input.buildDefinitionId}`

	Output	`buildId` : ID de build résultant
		`buildUrl` : URL pour consulter le résumé de la build
		`logUrl` : URL pour consulter les journaux
		`dropLocation` : emplacement cible des artefacts, le cas échéant
			`${MY_STAGE.MY_TASK.output.buildId}`
			`${MY_STAGE.MY_TASK.output.buildUrl}`
			`${MY_STAGE.MY_TASK.output.logUrl}`
			`${MY_STAGE.MY_TASK.output.dropLocation}`

vRO			
	Input	`workflowId` : ID du workflow à exécuter `parameters` : paramètres à transmettre au workflow	`${MY_STAGE.MY_TASK.input.workflowId}` `${MY_STAGE.MY_TASK.input.parameters}`
	Output	`workflowExecutionId` : ID de l'exécution du workflow `properties` : propriétés de sortie de l'exécution du workflow	`${MY_STAGE.MY_TASK.output.workflowExecutionId}` `${MY_STAGE.MY_TASK.output.properties}`

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

Tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
REST			

	Input	`url` : URL à appeler
		`action` : méthode HTTP à utiliser
		`headers` : en-têtes HTTP à transmettre
		`payload` : charge utile de la demande
		`fingerprint` : empreinte digitale à faire correspondre si l'URL est https
		`allowAllCerts` : lorsque la valeur est définie sur true, il peut s'agir de n'importe quel certificat disposant de l'URL https
			`${MY_STAGE.MY_TASK.input.url}`
			`${MY_STAGE.MY_TASK.input.action}`
			`${MY_STAGE.MY_TASK.input.headers}`
			`${MY_STAGE.MY_TASK.input.payload}`
			`${MY_STAGE.MY_TASK.input.fingerprint}`
			`${MY_STAGE.MY_TASK.input.allowAllCerts}`

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

Tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche

	Output	`responseCode` : codes de réponse HTTP
		`responseHeaders` : en-têtes de réponse HTTP
		`responseBody` : format de chaîne de la réponse reçue
		`responseJson` : réponse transversale si content-type est **application/json**
			`${MY_STAGE.MY_TASK.output.responseCode}`
			`${MY_STAGE.MY_TASK.output.responseHeaders}`
			`${MY_STAGE.MY_TASK.output.responseHeaders.header1}` # Faire référence à l'en-tête de réponse « header1 »
			`${MY_STAGE.MY_TASK.output.responseBody}`
			`${MY_STAGE.MY_TASK.output.responseJson}` # Faire référence à la réponse JSON
			`${MY_STAGE.MY_TASK.output.responseJson.a.b.c}` # Faire référence à l'objet imbriqué suivant le chemin d'accès a.b.c JSON dans la réponse

Interrogation			

	Input	`url` : URL à appeler
		`headers` : en-têtes HTTP à transmettre
		`exitCriteria` : critères à respecter pour que la tâche aboutisse ou échoue. Une paire clé-valeur de « success » → expression, « failure » → Expression :
		`pollCount` : nombre d'itérations à effectuer. Un administrateur Code Stream peut définir le nombre d'interrogations sur un maximum de 10 000.
		`pollIntervalSeconds` : nombre de secondes à attendre entre chaque itération. L'intervalle d'interrogation doit être supérieur ou égal à 60 secondes.
		`ignoreFailure` : lorsque la valeur est définie sur true, ignore les échecs de réponse intermédiaire
		`fingerprint` : empreinte digitale à faire correspondre si l'URL est https
		`allowAllCerts` : lorsque la valeur est définie sur true, il peut s'agir de n'importe quel certificat disposant de l'URL https
			`${MY_STAGE.MY_TASK.input.url}`
			`${MY_STAGE.MY_TASK.input.headers}`
			`${MY_STAGE.MY_TASK.input.exitCriteria}`
			`${MY_STAGE.MY_TASK.input.pollCount}`
			`${MY_STAGE.MY_TASK.input.pollIntervalSeconds}`
			`${MY_STAGE.MY_TASK.input.ignoreFailure}`
			`${MY_STAGE.MY_TASK.input.fingerprint}`
			`${MY_STAGE.MY_TASK.input.allowAllCerts}`

	Output	`responseCode` : codes de réponse HTTP
		`responseBody` : format de chaîne de la réponse reçue
		`responseJson` : réponse transversale si content-type est **application/json**
			`${MY_STAGE.MY_TASK.output.responseCode}`
			`${MY_STAGE.MY_TASK.output.responseBody}`
			`${MY_STAGE.MY_TASK.output.responseJson}` # Refer to response as JSON

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

Tâche	Scope	Key	Utilisation de SCOPE et KEY dans la tâche
PowerShell Pour exécuter une tâche PowerShell, vous devez : ■ Disposer d'une session active vers un hôte Windows distant. ■ Si vous avez l'intention d'entrer une commande PowerShell en base 64, calculez d'abord la longueur de la commande globale. Pour plus d'informations, reportez-vous à la section Types de tâches disponibles dans Code Stream.			

	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
		`useTLS` : tentative de connexion https
		`trustCert` : lorsque la valeur est définie sur true, approuve les certificats auto-signés
		`script` : script à exécuter
		`workingDirectory` : chemin d'accès au répertoire vers lequel basculer avant d'exécuter le script
		`environmentVariables` : paire clé-valeur de variable d'environnement à définir
		`arguments` : arguments à passer au script
			`${MY_STAGE.MY_TASK.input.host}`
			`${MY_STAGE.MY_TASK.input.username}`
			`${MY_STAGE.MY_TASK.input.password}`
			`${MY_STAGE.MY_TASK.input.useTLS}`
			`${MY_STAGE.MY_TASK.input.trustCert}`
			`${MY_STAGE.MY_TASK.input.script}`
			`${MY_STAGE.MY_TASK.input.workingDirectory}`
			`${MY_STAGE.MY_TASK.input.environmentVariables}`
			`${MY_STAGE.MY_TASK.input.arguments}`

	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
			`${MY_STAGE.MY_TASK.output.response}`
			`${MY_STAGE.MY_TASK.output.responseFilePath}`
			`${MY_STAGE.MY_TASK.output.exitCode}`
			`${MY_STAGE.MY_TASK.output.logFilePath}`
			`${MY_STAGE.MY_TASK.output.errorFilePath}`

SSH			

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

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
			`${MY_STAGE.MY_TASK.input.host}`
			`${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}`

	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
			`${MY_STAGE.MY_TASK.output.response}`
			`${MY_STAGE.MY_TASK.output.responseFilePath}`
			`${MY_STAGE.MY_TASK.output.exitCode}`
			`${MY_STAGE.MY_TASK.output.logFilePath}`
			`${MY_STAGE.MY_TASK.output.errorFilePath}`

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	`${<Stage Key>.<Task Key>.output.<Task output key>}`
Pour faire référence à la valeur de sortie de la tâche précédente comme entrée de la tâche actuelle	`${<Previous/Current Stage key>.<Previous task key not in current Task group>.output.<task output key>}`

Pour en savoir plus

Pour en savoir plus sur la liaison de variables dans les tâches, consultez :

- Utilisation des liaisons de variables dans les pipelines Code Stream
- Utilisation des liaisons de variables dans une tâche de condition pour l'exécution ou l'arrêt d'un pipeline dans Code Stream
- Types de tâches disponibles dans Code Stream

Envoi de notifications relatives à un pipeline dans Code Stream

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

Pour envoyer des notifications lorsqu'un pipeline s'exécute, vous pouvez configurer des notifications Code Stream 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'un 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.

Pour créer un ticket Jira en cas d'échec d'une tâche de pipeline, vous pouvez configurer une notification. Pour envoyer une demande à un canal Slack concernant l'état d'un pipeline basé sur l'événement de pipeline, vous pouvez configurer une notification Webhook.

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 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 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 pour :	Actions :
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 Pour configurer la notification Slack, entrez les informations.

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 :

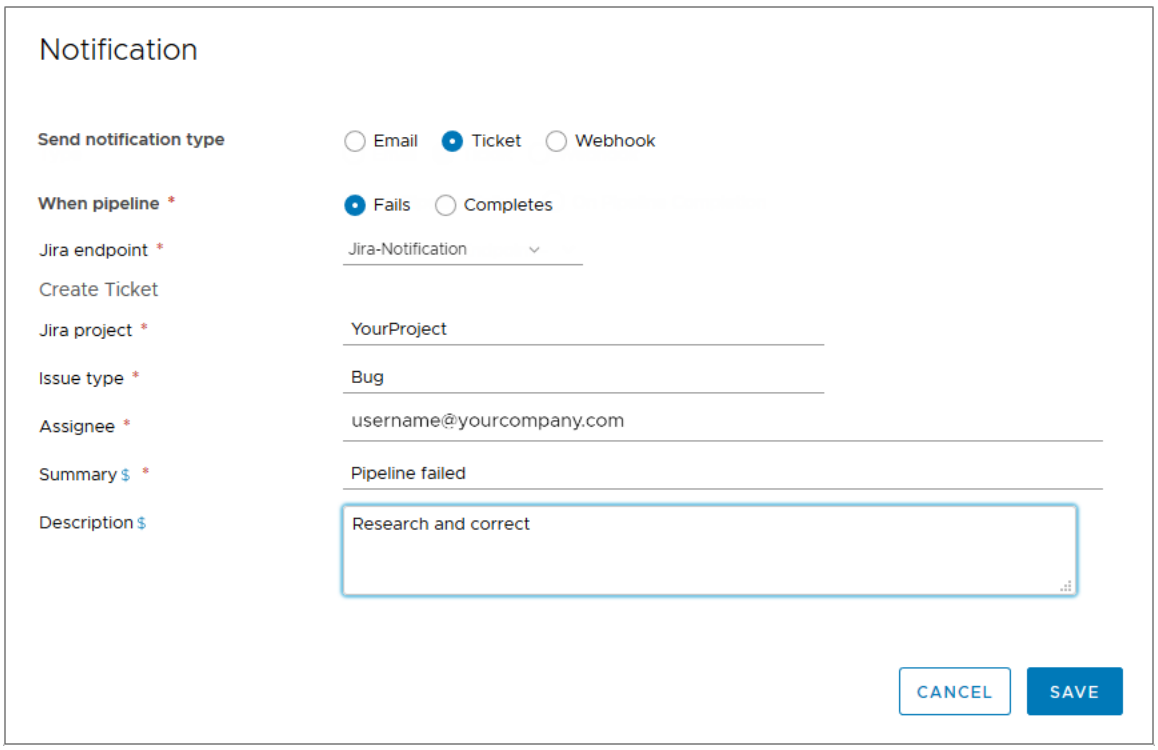
```
Codestream APP [12:01 AM]
Tested by User1 - Staging Pipeline 'User1-Pipeline', Pipeline ID
'e9b5884d809ce2755728177f70f8a' succeeded
```

6 Pour créer un ticket Jira, configurez les informations de ticket.

a Sélectionnez **Ticket**.

b Pour configurer la notification JIRA, entrez les informations.

c Cliquez sur **Enregistrer**.



Résultats

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

Étape suivante

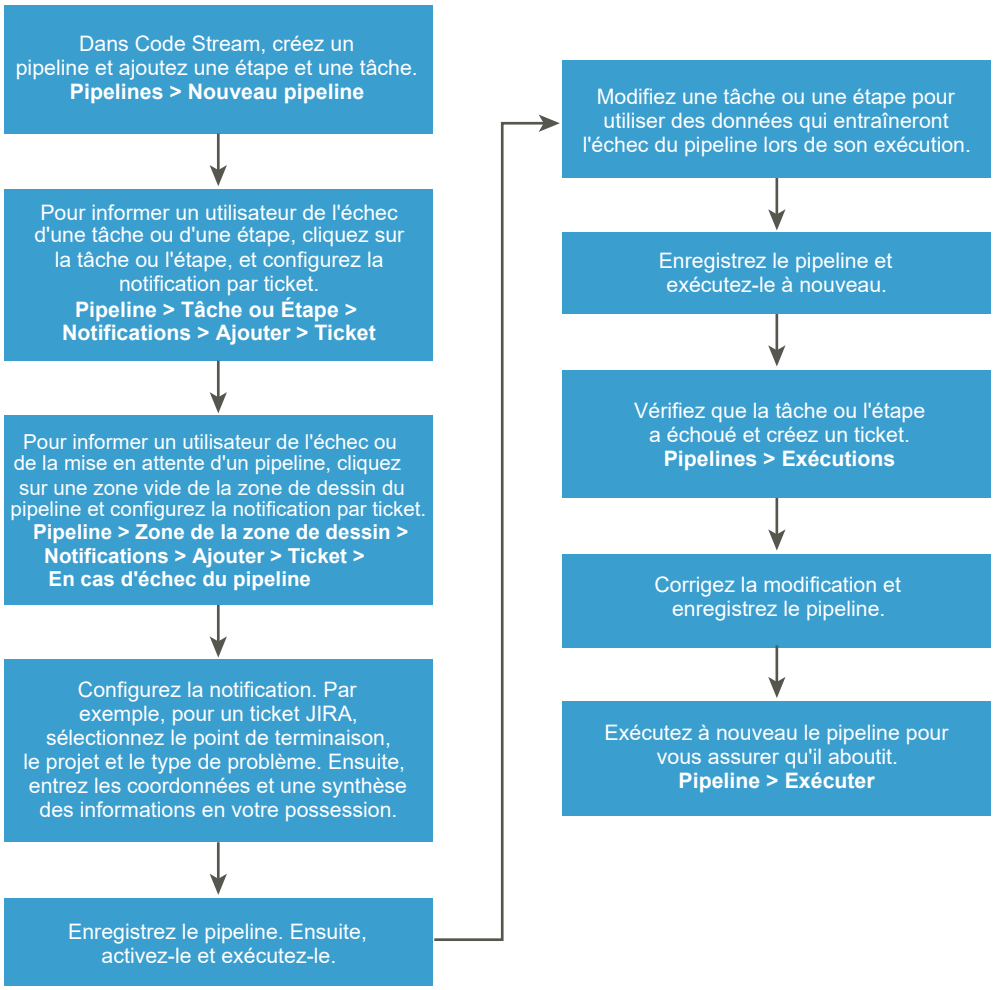
Pour obtenir un exemple détaillé de création d'une notification, reportez-vous à la section Création d'un ticket JIRA dans Code Stream en cas d'échec d'une tâche de pipeline.

Création d'un ticket JIRA dans 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 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. 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 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.

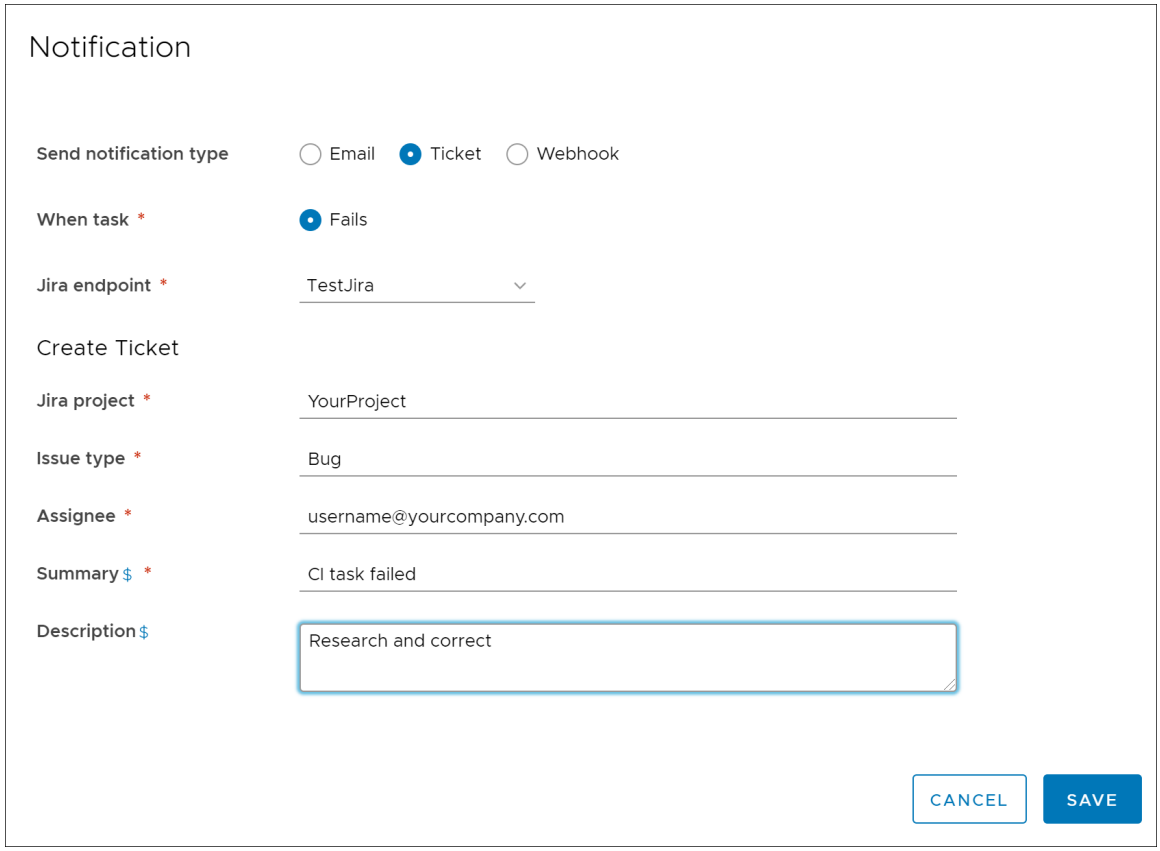
   a Cliquez sur **Ticket**.

   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**.



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 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 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 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. Pour restaurer en cas d'échec, vous associez le pipeline de restauration à des tâches ou des étapes.

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

- En tant qu'ingénieur en publication, je souhaite que 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 Code Stream.

Conditions préalables

- Assurez-vous d'être membre d'un projet dans Code Stream. Si ce n'est pas le cas, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans Code Stream.
- Configurez les clusters Kubernetes sur lesquels votre pipeline déploiera votre application. Configurez un cluster de développement et un cluster de production.
- Vérifiez que vous disposez d'une configuration de registre Docker.
- Identifier un projet qui regroupera tous vos travaux, y compris votre pipeline, vos points de terminaison et vos tableaux de bord.
- Familiarisez-vous avec le modèle intelligent CD comme décrit dans la partie CD de la section Planification d'une build native CICD dans Code Stream avant d'utiliser le modèle de pipeline intelligent. Par exemple :
  - Créez le développement Kubernetes et les points de terminaison de production qui déploient votre image d'application sur les clusters Kubernetes.
  - Préparez le fichier YAML Kubernetes qui crée l'espace de noms, le service et le déploiement. Si vous devez télécharger une image à partir d'un référentiel privé, le fichier YAML doit inclure une section avec le secret de configuration Docker.

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**.

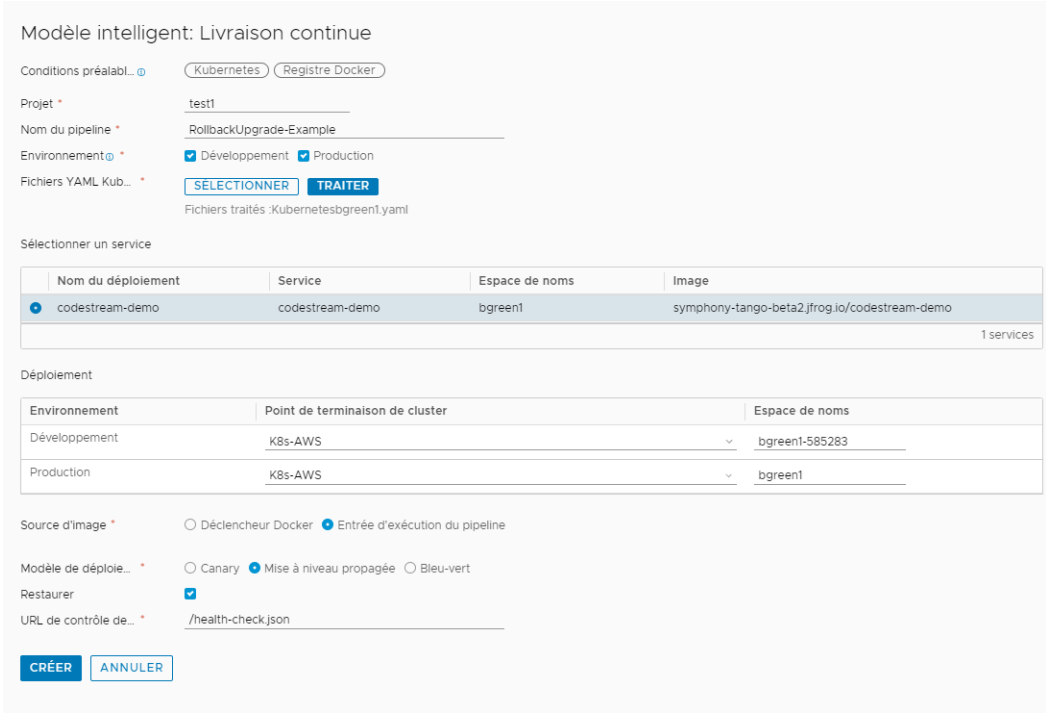
      Le modèle de pipeline intelligent affiche les services et les environnements de déploiement disponibles.
   e. Sélectionnez le service que le pipeline utilisera pour le déploiement.
   f. Sélectionnez les points de terminaison de cluster pour l'environnement Dev et l'environnement Prod.

g Pour la source Image, sélectionnez **Entrée d'exécution du pipeline**.

h Pour le modèle de déploiement, sélectionnez **Mise à niveau propagée**.

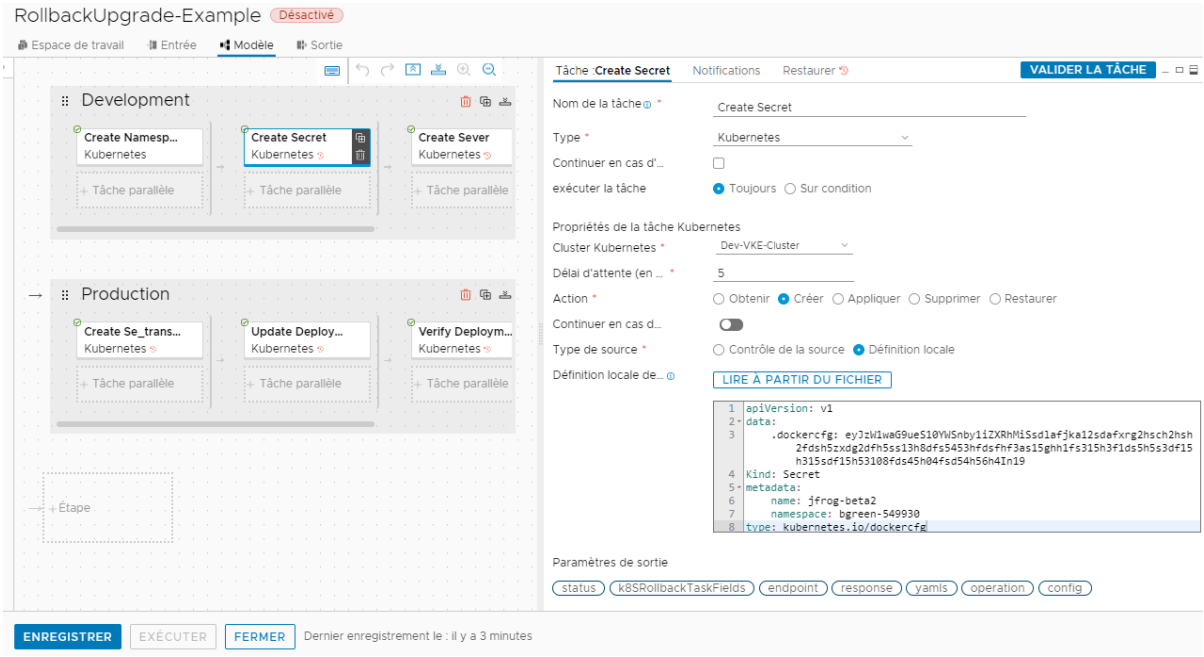
i Cliquez sur **Restaurer**.

j Fournissez l'**URL de contrôle de santé**.



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

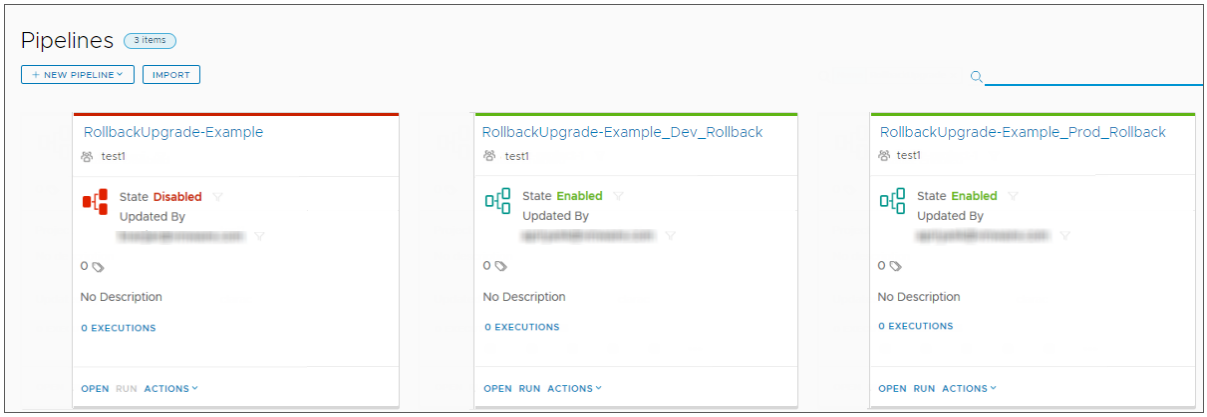
Le pipeline nommé RollbackUpgrade-Example s'affiche et l'icône de restauration s'affiche sur les tâches qui peuvent être restaurées à l'étape de développement et à l'étape de production.



4 Fermez le pipeline.

Sur la page Pipelines, le pipeline que vous avez créé s'affiche, et un nouveau pipeline pour chaque étape de votre pipeline s'affiche.

- MiseÀniveauPropagée-Exemple. 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. L'échec de tâches à l'étape de développement, telles que le service **Créer un service**, **Créer un secret**, **Créer un déploiement** et **Vérifier le déploiement** appelle ce pipeline de développement par restauration. Pour garantir la restauration des tâches de développement, Code Stream active le pipeline de développement par restauration par défaut.
- MiseÀniveauPropagée-Exemple_Prod_Restauration. L'échec de tâches à l'étape de production, telles que **Déployer la phase 1**, **Vérifier la phase 1**, **Déployer la phase de lancement Terminer la phase de lancement** et **Vérifier la phase de lancement** appelle ce pipeline de production par restauration. Pour garantir la restauration des tâches de production, Code Stream active le pipeline de production par restauration par défaut.

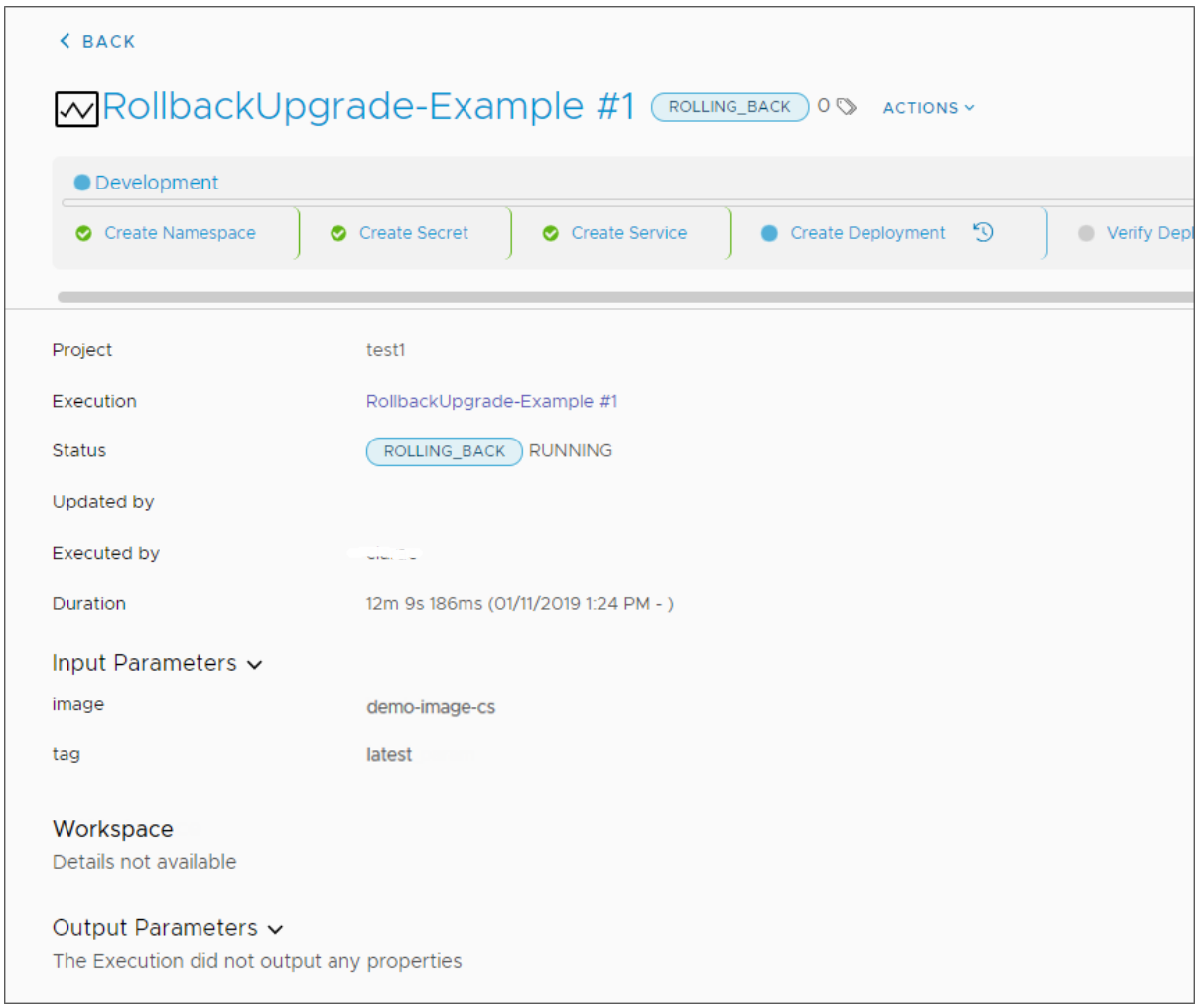


5 Activez et exécutez le pipeline que vous avez créé.

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

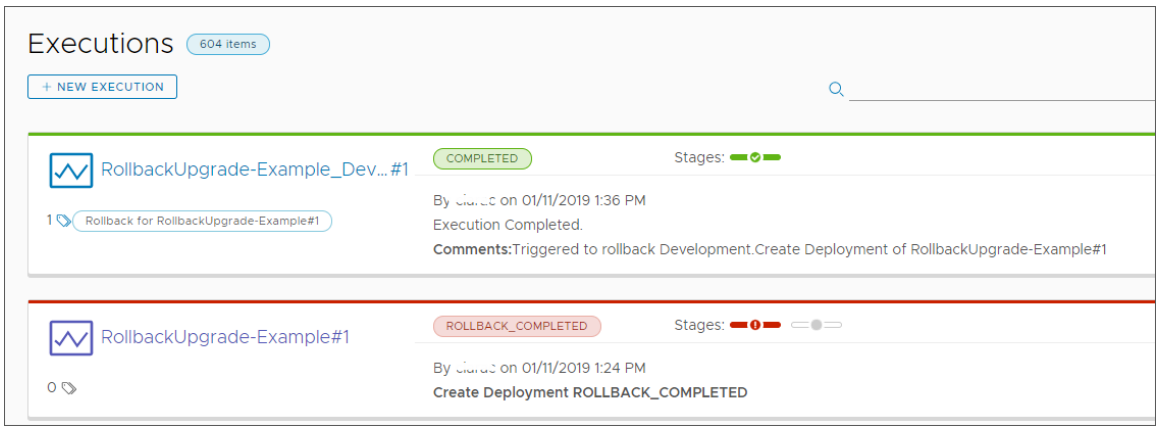
6 Sur la page Exécutions, sélectionnez **Actions > Afficher l'exécution** pour surveiller l'exécution du pipeline.

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 RollingUpgrade-Example_Dev_Rollback est déclenché et restaure le déploiement, puis l'état du pipeline devient **ROLLING_BACK**.



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

- Le pipeline que vous avez créé a été restauré et affiche **ROLLBACK_COMPLETED**.
- Le pipeline de développement par restauration qui a été déclenché et qui a effectué la restauration indique **COMPLETED**.



Résultats

Félicitations ! Vous avez défini un pipeline avec restauration et 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 Code Stream

4

Avant que 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.

Pour planifier votre build d'intégration continue et de livraison continue, nous avons inclus plusieurs exemples qui vous montrent comment procéder. Ces plans décrivent les conditions préalables et les présentations qui peuvent vous aider à préparer et à utiliser efficacement la capacité de build native lorsque vous créez vos pipelines.

Ce chapitre contient les rubriques suivantes :

Configuration de l'espace de travail de pipeline

Pour exécuter des tâches d'intégration continue et des tâches personnalisées, vous devez configurer un espace de travail pour votre pipeline Code Stream.

Dans l'espace de travail du pipeline, sélectionnez le **Type** Docker ou Kubernetes, puis fournissez le point de terminaison respectif. Les plates-formes Docker et Kubernetes gèrent l'intégralité du cycle de vie du conteneur que Code Stream déploie pour exécuter la tâche d'intégration continue (CI) ou la tâche personnalisée.

- L'espace de travail Docker requiert le point de terminaison de l'hôte Docker, l'URL de l'image du générateur, le registre d'image, le répertoire de travail, le cache, les variables d'environnement, la limite de CPU et la limite de mémoire. Vous pouvez également créer un clone du référentiel Git.
- L'espace de travail Kubernetes requiert le point de terminaison de l'API Kubernetes, l'URL de l'image du générateur, le registre d'image, l'espace de noms, le port de nœud, la réclamation de volume persistant (PVC), le répertoire de travail, les variables d'environnement, la limite de CPU et la limite de mémoire. Vous pouvez également créer un clone du référentiel Git.

La configuration de l'espace de travail de pipeline comporte de nombreux paramètres communs et d'autres paramètres spécifiques du type d'espace de travail, comme décrit dans le tableau suivant.

Tableau 4-1. Zones, détails et disponibilité de l'espace de travail

Sélection	Description	Détails et disponibilité
Type	Type d'espace de travail.	Disponible avec Docker ou Kubernetes.
Point de terminaison d'hôte	Point de terminaison d'hôte sur lequel l'intégration continue (CI) et les tâches personnalisées s'exécutent.	Disponible avec l'espace de travail Docker lorsque vous sélectionnez le point de terminaison de l'hôte Docker. Disponible avec l'espace de travail Kubernetes lorsque vous sélectionnez le point de terminaison d'API Kubernetes.
URL de l'image du générateur	Nom et emplacement de l'image du générateur. Un conteneur est créé à l'aide de cette image sur l'hôte Docker et le cluster Kubernetes. Les tâches d'intégration continue (CI) et les tâches personnalisées s'exécutent à l'intérieur de ce conteneur.	Exemple : **`fedora:latest`** L'image de générateur doit inclure `curl` ou `wget`.
Registre d'image	Si l'image du générateur est disponible dans un registre et si le registre nécessite des informations d'identification, vous devez créer un point de terminaison de registre d'image, puis le sélectionner ici afin que l'image puisse être extraite du registre.	Disponible avec les espaces de travail Docker et Kubernetes.
Répertoire de travail	Le répertoire de travail est l'emplacement à l'intérieur du conteneur où les étapes de la tâche d'intégration continue (CI) s'exécutent, et l'emplacement où le code est cloné lorsqu'un Webhook Git déclenche une exécution de pipeline.	Disponible avec Docker ou Kubernetes.
Espace de noms	Si vous n'entrez pas d'espace de noms, Code Stream crée un nom unique dans le cluster Kubernetes que vous avez fourni.	Spécifique à l'espace de travail Kubernetes.

Tableau 4-1. Zones, détails et disponibilité de l'espace de travail (suite)

Sélection	Description	Détails et disponibilité
Proxy	Pour communiquer avec l'espace de travail du cluster Kubernetes, Code Stream déploie une instance de proxy unique dans l'espace de noms `codestream-proxy` pour chaque cluster Kubernetes. Vous pouvez sélectionner le type **NodePort** ou **LoadBalancer** en fonction de la configuration du cluster. L'option que vous sélectionnez dépend de la nature du cluster Kubernetes déployé. ■ En général, si l'URL du serveur d'API Kubernetes spécifiée dans le point de terminaison est exposée via l'un des nœuds principaux, sélectionnez **NodePort**. ■ Si l'URL du serveur d'API Kubernetes est exposée par un équilibrage de charge, tel qu'Amazon EKS (Elastic Kubernetes Service), sélectionnez **LoadBalancer**.	
NodePort	Code Stream utilise le port de nœud pour communiquer avec le conteneur s'exécutant à l'intérieur du cluster Kubernetes. Si vous ne sélectionnez pas de port, Code Stream utilise un port éphémère attribué par Kubernetes. Vous devez vous assurer que la configuration des règles de pare-feu autorise l'entrée vers la plage de ports éphémères (30000-32767). Si vous entrez un port, vous devez vous assurer qu'aucun autre service du cluster ne l'utilise déjà et que le port est autorisé dans les règles de pare-feu.	Spécifique à l'espace de travail Kubernetes.
Réclamation de volume persistant	Fournit un moyen pour l'espace de travail Kubernetes de conserver les fichiers dans les exécutions de pipeline. Lorsque vous fournissez un nom de réclamation de volume persistant, il peut stocker les journaux, les artefacts et le cache. Pour plus d'informations sur la création d'une réclamation de volume persistant, consultez la documentation Kubernetes à l'adresse `https://kubernetes.io/docs/concepts/storage/persistent-volumes/`.	Spécifique à l'espace de travail Kubernetes.

Tableau 4-1. Zones, détails et disponibilité de l'espace de travail (suite)

Sélection	Description	Détails et disponibilité
Variables d'environnement	Les paires clé-valeur qui sont transmises ici seront disponibles pour toutes les tâches d'intégration continue (CI) et les tâches personnalisées d'un pipeline lors de son exécution.	Disponible avec Docker ou Kubernetes. Les références aux variables peuvent être transmises ici. Les variables d'environnement fournies dans l'espace de travail sont transmises à toutes les tâches d'intégration continue (CI) et aux tâches personnalisées dans le pipeline. Si les variables d'environnement ne sont pas transmises ici, elles doivent être explicitement transmises à chaque tâche d'intégration continue (CI) et tâche personnalisée dans le pipeline.
Limites de CPU	Limites des ressources de CPU pour le conteneur d'intégration continue (CI) ou le conteneur de tâches personnalisées.	La valeur par défaut est `1`.
Limites de mémoire	Limites de mémoire pour le conteneur d'intégration continue (CI) ou le conteneur de tâches personnalisées.	L'unité est `Mo`.

Tableau 4-1. Zones, détails et disponibilité de l'espace de travail (suite)

Sélection	Description	Détails et disponibilité
Clone Git	Lorsque vous sélectionnez **Clone Git** et qu'un Webhook Git appelle le pipeline, le code est cloné dans l'espace de travail (conteneur).	Si vous n'activez pas l'option **Clone Git**, vous devez configurer une autre tâche d'intégration continue (CI) explicite dans le pipeline pour d'abord cloner le code, puis effectuer d'autres étapes, telles que la génération et le test.

Cache	L'espace de travail Code Stream vous permet de mettre en cache un ensemble de répertoires ou de fichiers pour accélérer les exécutions de pipeline suivantes. Les exemples de ces répertoires incluent `.m2` et `npm_modules`. Si vous n'avez pas besoin de la mise en cache des données entre les exécutions de pipeline. Une réclamation de volume persistant n'est pas nécessaire.
	Les artefacts, tels que les fichiers ou les répertoires du conteneur, sont mis en cache pour une réutilisation avec les exécutions de pipeline. Par exemple, les dossiers `node_modules` ou `.m2` peuvent être mis en cache. **Cache** accepte une liste de chemins d'accès.
	Par exemple :
	`workspace:`
	`  type: K8S`
	`  endpoint: K8S-Micro`
	`  image: fedora:latest`
	`  registry: Docker Registry`
	`  path: ''`
	`  cache:`
	`  - /path/to/m2`
	`  - /path/to/node_modules`
		Spécifique du type d'espace de travail.
		Dans l'espace de travail Docker, le **cache** est obtenu en utilisant un chemin partagé dans l'hôte Docker pour conserver les données, les artefacts et les journaux mis en cache.
		Dans l'espace de travail Kubernetes, pour activer l'utilisation de **Cache**, vous devez fournir une réclamation de volume persistant. Dans le cas contraire, l'option **Cache** n'est pas disponible.

Lors de l'utilisation d'un point de terminaison d'API Kubernetes dans l'espace de travail de pipeline, Code Stream crée les ressources Kubernetes nécessaires, telles que ConfigMap, Secret et Pod, pour exécuter la tâche d'intégration continue (CI) ou la tâche personnalisée. Code Stream communique avec le conteneur à l'aide du port de nœud.

Pour partager des données entre des exécutions de pipeline, vous devez fournir une réclamation de volume persistant, et Code Stream montera la réclamation de volume persistant sur le conteneur pour stocker les données et l'utilisera pour les exécutions de pipeline suivantes.

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

Pour créer un pipeline d'intégration continue et de livraison continue (CICD) dans Code Stream, vous pouvez utiliser le modèle de pipeline intelligent CICD. Pour planifier votre build native CICD,

collectez les informations du modèle de pipeline intelligent avant de créer le pipeline dans cet exemple de plan.

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

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 la destination du déploiement de votre image en fonction des types d'environnement que vous sélectionnez, par exemple Dev (Développement) et Prod (Production). Le pipeline publiera votre image de conteneur et effectuera les actions requises pour l'exécuter. Après l'exécution de votre pipeline, vous pouvez surveiller les tendances en matière d'exécution de pipeline.

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

Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.

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

Pour planifier l'étape d'intégration continue de votre pipeline, configurez les conditions externes et internes requises, et déterminez les informations requises dans la partie d'intégration continue du modèle de pipeline intelligent. En voici un résumé.

Cet exemple utilise un espace de travail Docker.

Points de terminaison et référentiels dont vous aurez besoin :

- D'un référentiel de code source Git dans lequel vos développeurs archivent leur code. 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 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 Code Stream. Si ce n'est pas le cas, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans Code Stream.

Vous aurez besoin d'un Webhook Git, ce qui permet à 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 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 créer un point de terminaison de registre d'image, 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 livraison continue de votre pipeline, configurez les conditions externes et internes requises, et déterminez les informations à saisir dans la partie de livraison continue du modèle de pipeline intelligent.

Points de terminaison dont vous aurez besoin :

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

Types d'environnement et fichiers :

- Tous les types d'environnement dans lesquels 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 4-2. 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.

  Le fichier YAML Kubernetes comporte trois sections obligatoires pour l'espace de noms, le service et le déploiement, et une section facultative pour le secret. Si vous prévoyez de créer un pipeline en téléchargeant une image à partir d'un référentiel privé, vous devez inclure une section avec le secret de configuration Docker. Si le pipeline que vous créez utilise uniquement des images accessibles publiquement, aucun secret n'est requis. L'exemple de fichier YAML suivant comporte quatre sections.

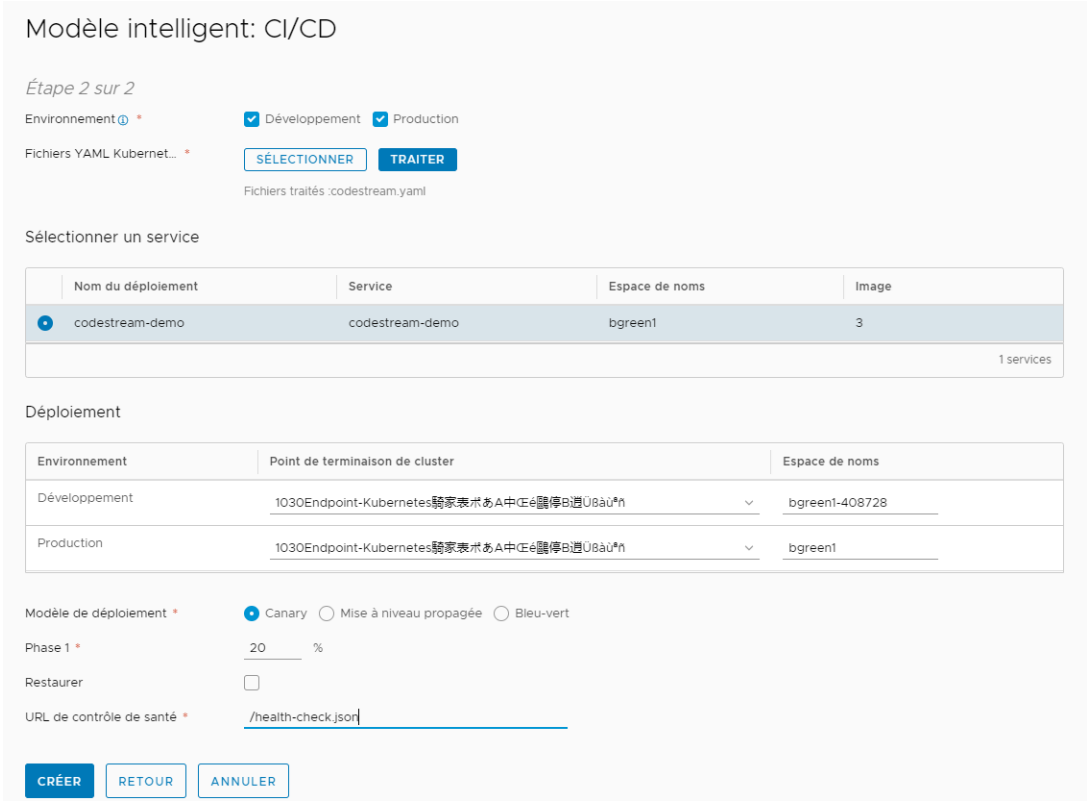
```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream
  namespace: codestream
---
apiVersion: v1
data:
  .dockerconfigjson:
eyJhdXRocyI6eyJodHRwczovL2luZ1234567890lci5pby92MS8iOnsidXNlcm5hbWUiOiJhdXRvbWF0aW9uYmV0YSI
sInBhc3N3b3JkIjoiVk13YXJlQDEyMyIsImVtYWlsIjoiYXV0b21hdGlvbmJldGF1c2VyQGdtYWlsLmNvbSIsImF1dG
giOiJZWFYwYjIxaGRHbHZibUpsZEdFNlZrMTNZWEpsUURFeU13PT0ifX19
kind: Secret
metadata:
  name: dockerhub-secret
  namespace: codestream
type: kubernetes.io/dockerconfigjson
---
apiVersion: v1
kind: Service
metadata:
  name: codestream-demo
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  ports:
    - port: 80
  selector:
    app: codestream-demo
```

```
    tier: frontend
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: codestream-demo
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  replicas: 10
  selector:
    matchLabels:
      app: codestream-demo
      tier: frontend
  template:
    metadata:
      labels:
        app: codestream-demo
        tier: frontend
    spec:
      containers:
      - name: codestream-demo
        image: automationbeta/codestream-demo:01
        ports:
        - containerPort: 80
          name: codestream-demo
      imagePullSecrets:
      - name: dockerhub-secret
```

Note Le fichier YAML Kubernetes est également utilisé dans le modèle de pipeline intelligent CD, comme dans les exemples de cas d'utilisation suivants :

- Déploiement d'une application dans Code Stream vers un déploiement Bleu-vert
- Restauration d'un déploiement dans Code Stream
- Utilisation du déclencheur Docker dans Code Stream pour exécuter un pipeline de livraison continue

Pour appliquer le fichier dans le modèle intelligent, cliquez sur **Sélectionner** et sélectionnez le fichier YAML Kubernetes. Cliquez ensuite 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.

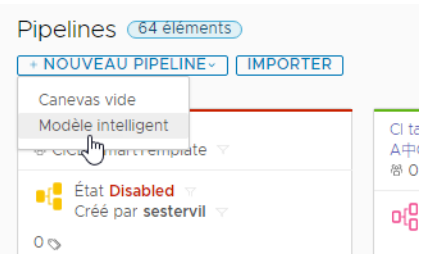


Pour découvrir un exemple d'utilisation du modèle de pipeline intelligent pour créer un pipeline en vue d'un déploiement Bleu-vert, reportez-vous à la section Déploiement d'une application dans Code Stream vers un déploiement Bleu-vert.

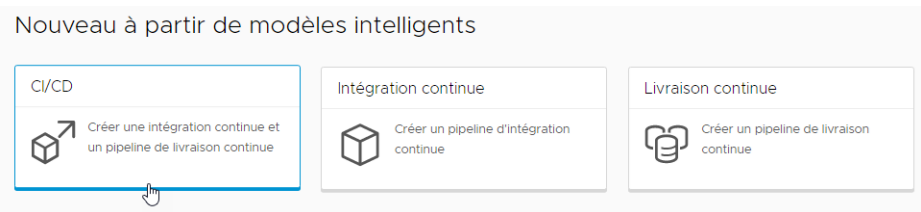
Création du pipeline CICD à l'aide du modèle de pipeline intelligent

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.

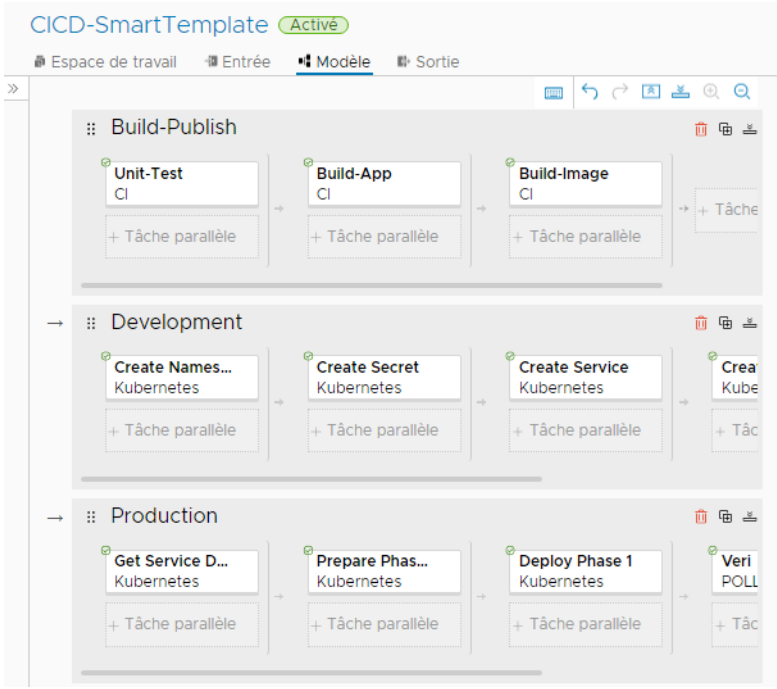
Sur la page Pipelines, sélectionnez **Nouveau pipeline > Modèles intelligents**.



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 Code Stream.

Planification d'une build native d'intégration continue dans 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 d'intégration continue. Pour planifier votre build native d'intégration continue, collectez les informations du modèle de pipeline intelligent avant de créer le pipeline dans cet exemple de plan.

Lorsque vous remplissez le modèle de pipeline intelligent, il crée un pipeline d'intégration continue dans votre référentiel et exécute les actions permettant l'exécution du pipeline. 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 d'intégration continue :

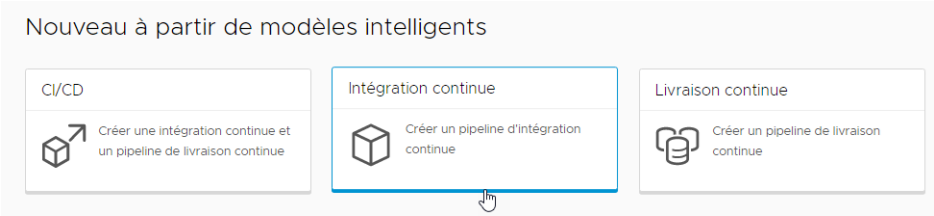
- Identifier un projet qui regroupera tous vos travaux, y compris votre pipeline, vos points de terminaison et vos tableaux de bord.
- Collectez les informations de votre build comme décrit dans la partie de livraison continue de la section Planification d'une build native CICD dans Code Stream avant d'utiliser le modèle de pipeline intelligent.

  Par exemple, ajoutez un point de terminaison Kubernetes où Code Stream déploiera le conteneur.

Ensuite, créez un pipeline à l'aide du modèle de pipeline intelligent d'intégration continue.

Dans Pipelines, sélectionnez **Modèles intelligents**.

Sélectionnez le modèle de pipeline intelligent d'intégration continue.



Pour enregistrer le pipeline avec les étapes qu'il crée, remplissez le modèle et entrez un nom pour le pipeline. Pour enregistrer le pipeline avec les étapes qu'il crée, cliquez sur **Créer**.

L'espace de travail de pipeline Code Stream prend en charge Docker et Kubernetes pour les tâches d'intégration continue et les tâches personnalisées.

Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.

Pour apporter des modifications finales, vous pouvez modifier le pipeline. Ensuite, vous pouvez activer le pipeline et l'exécuter. Après l'exécution du pipeline :

- 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. Pour générer des rapports sur d'autres indicateurs de performance clés, vous pouvez créer un tableau de bord personnalisé.

Pour obtenir un exemple détaillé, reportez-vous à la section Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans Code Stream.

Planification d'une build native de livraison continue dans Code Stream avant d'utiliser le modèle de pipeline intelligent

Pour créer un pipeline de livraison continue (CD) dans Code Stream, vous pouvez utiliser le modèle de pipeline intelligent de livraison continue. Pour planifier votre build native de livraison continue, collectez les informations du modèle de pipeline intelligent avant de créer le pipeline dans cet exemple de plan.

Lorsque vous remplissez le modèle de pipeline intelligent, il crée un pipeline de livraison continue dans votre référentiel et exécute les actions permettant l'exécution du pipeline. Après l'exécution de votre pipeline, vous pouvez surveiller les tendances en matière d'exécution de pipeline.

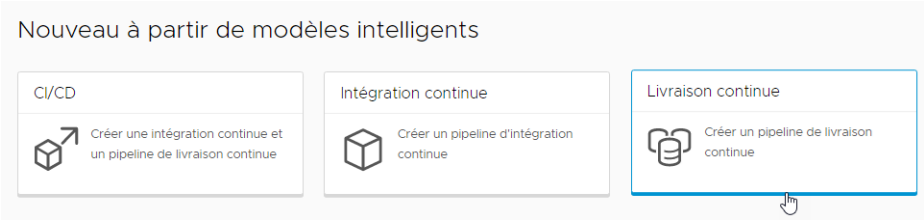
Pour planifier votre build avant d'utiliser le modèle de pipeline intelligent de livraison continue :

- Identifier un projet qui regroupera tous vos travaux, y compris votre pipeline, vos points de terminaison et vos tableaux de bord.
- Collectez les informations de votre build comme décrit dans la partie de livraison continue de la section Planification d'une build native CICD dans Code Stream avant d'utiliser le modèle de pipeline intelligent. Par exemple :
  - Ajouter un point de terminaison Kubernetes où Code Stream déploiera le conteneur.
  - Préparez le fichier YAML Kubernetes qui crée l'espace de noms, le service et le déploiement. Pour télécharger une image à partir d'un référentiel privé, le fichier YAML doit inclure une section avec le secret de configuration Docker.

Ensuite, vous créez un pipeline à l'aide du modèle de pipeline intelligent de livraison continue.

Dans Pipelines, sélectionnez **Modèles intelligents**.

Sélectionnez le modèle de pipeline intelligent de livraison continue.



Vous remplissez le modèle et entrez un nom pour le pipeline. Pour enregistrer le pipeline avec les étapes qu'il crée, cliquez sur **Créer**.

L'espace de travail de pipeline Code Stream prend en charge Docker et Kubernetes pour les tâches d'intégration continue et les tâches personnalisées.

Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.

Pour apporter des modifications finales, vous pouvez modifier le pipeline. Ensuite, vous pouvez activer le pipeline et l'exécuter. Après l'exécution du pipeline :

- 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. Pour générer des rapports sur d'autres indicateurs de performance clés, vous pouvez créer un tableau de bord personnalisé.

Pour obtenir un exemple détaillé, reportez-vous à la section Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans Code Stream.

Planification d'une build native CICD dans Code Stream avant d'ajouter des tâches manuellement

Pour créer un pipeline d'intégration continue et de prestation continue (CICD) dans Code Stream, vous pouvez manuellement ajouter des étapes et des tâches. Pour planifier votre build native CICD, 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 livraison 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.

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

L'espace de travail de pipeline Code Stream prend en charge Docker et Kubernetes pour les tâches d'intégration continue et les tâches personnalisées.

Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.

Planification des exigences externes et internes

Pour planifier les étapes d'intégration continue et de livraison continue, les exigences suivantes indiquent les opérations à effectuer avant de créer votre pipeline.

Cet exemple utilise un espace de travail Docker.

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 leur code. 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'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 Code Stream. Si ce n'est pas le cas, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans Code Stream.

Vous aurez besoin d'un Webhook Git, ce qui permet à 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 Code Stream pour exécuter un pipeline.

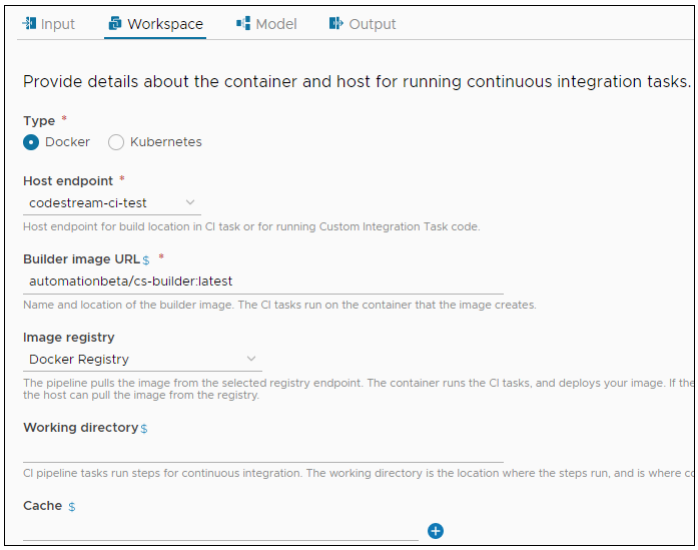
Création du pipeline CICD et configuration de l'espace de travail

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 :

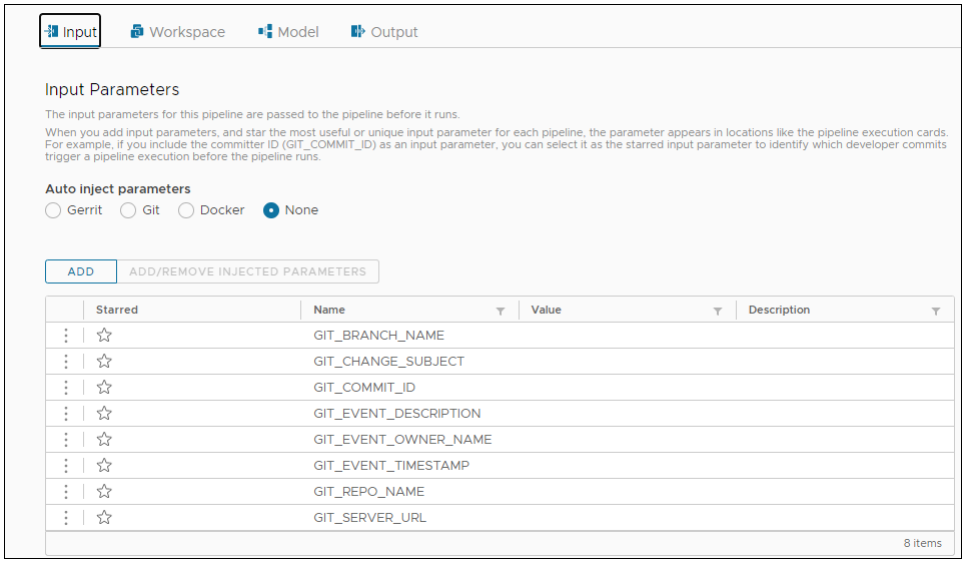
- Incluez votre hôte de build Docker.
- Entrez l'URL de votre image de générateur.
- Sélectionnez le point de terminaison de registre d'images afin que le pipeline puisse en extraire l'image. Le conteneur exécute les tâches de CI et déploie votre image. Si le registre requiert des informations d'identification,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

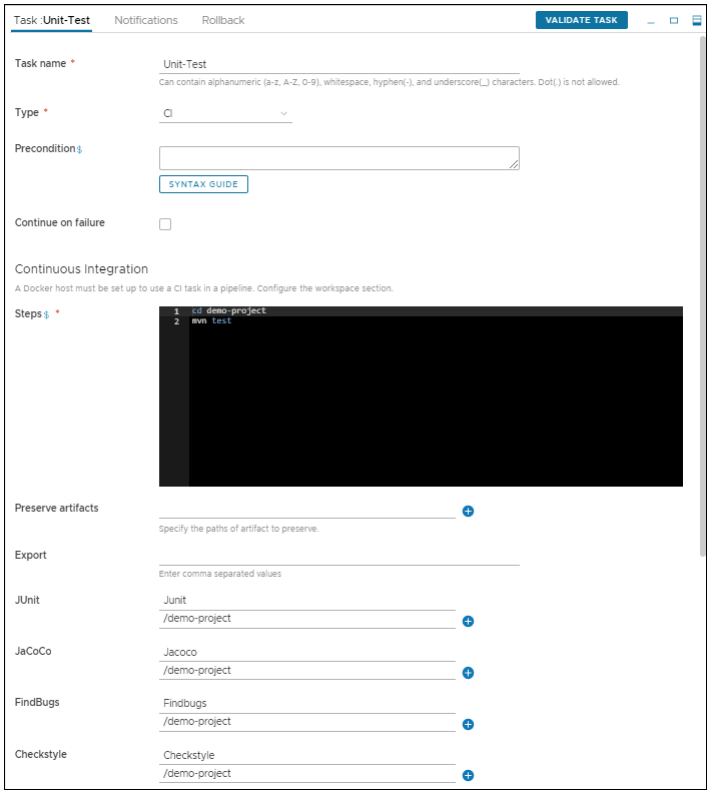
- Si votre pipeline utilise des paramètres d'entrée à partir d'un événement déclencheur Git, Gerrit ou Docker, sélectionnez le type de déclencheur pour les paramètres d'insertion automatique. Ces événements sont les suivants : Modifier l'objet de Gerrit ou Git ou Nom du propriétaire de l'événement pour 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.



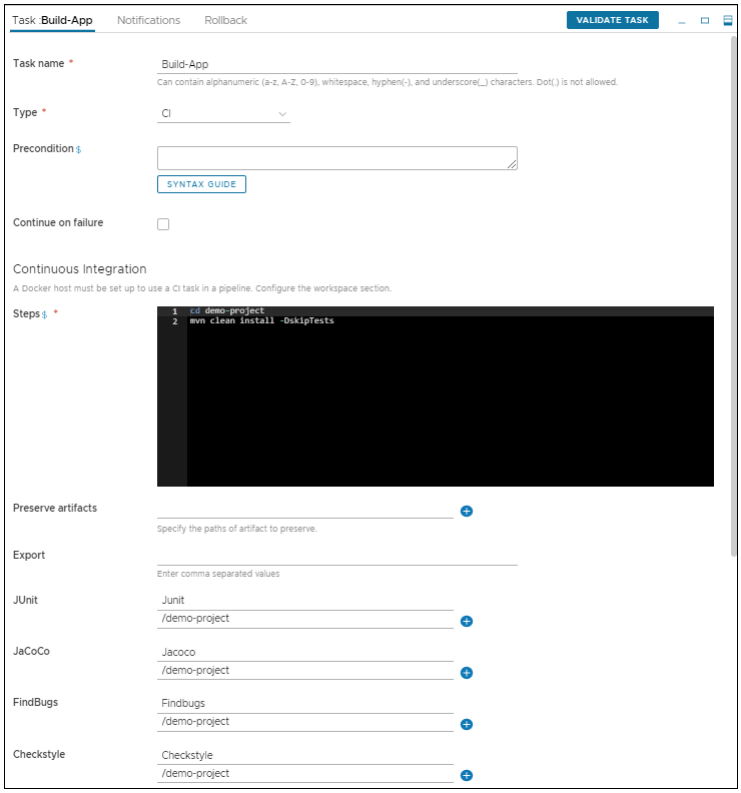
Configurez le pipeline pour tester votre code :

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



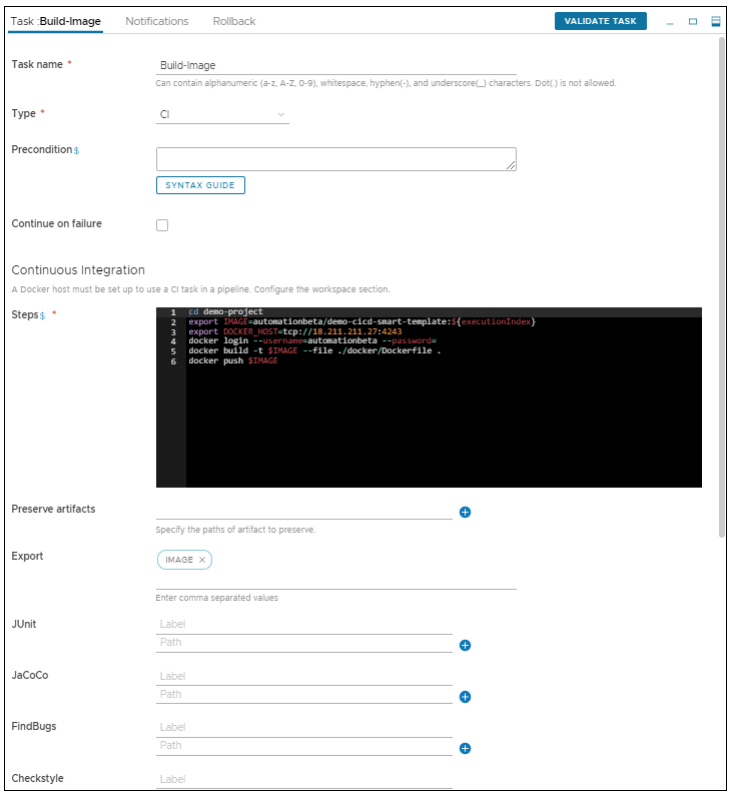
Configurez le pipeline pour générer votre code :

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



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.



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 souhaiterez 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 Code Stream, reportez-vous à la section Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans Code Stream.

Planification de la restauration dans Code Stream

Si l'exécution d'un pipeline échoue, vous pouvez utiliser la restauration pour rétablir votre environnement à un état précédemment stable. Pour utiliser la récupération, planifiez un flux de récupération et comprenez 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 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 créerez votre pipeline de restauration avec ces tâches.

Vous pouvez créer manuellement un pipeline de restauration ou 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, 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 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.

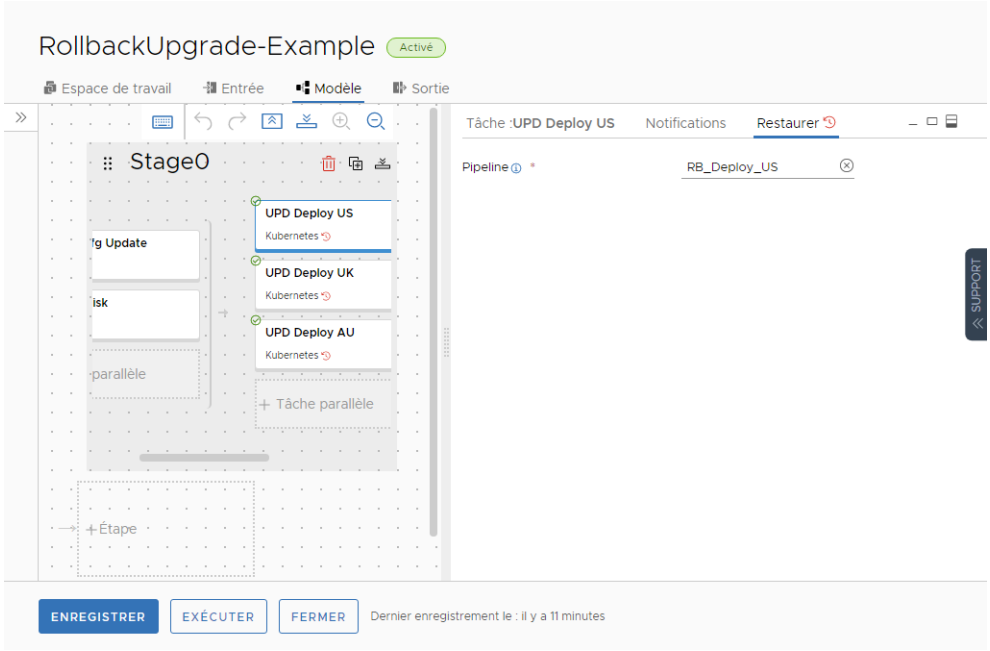
Tableau 4-3. Critères déterminant la séquence de restauration

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 Code Stream

5

Code Stream modélise et prend en charge votre cycle de vie de version DevOps, et teste et publie en continu vos applications dans des environnements de développement et de production.

Vous avez déjà configuré tout ce dont vous aviez besoin et pouvez maintenant utiliser Code Stream. Reportez-vous à la section Chapitre 2 Configuration de 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 Code Stream pour déployer des applications basée sur conteneur ou traditionnelles.

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

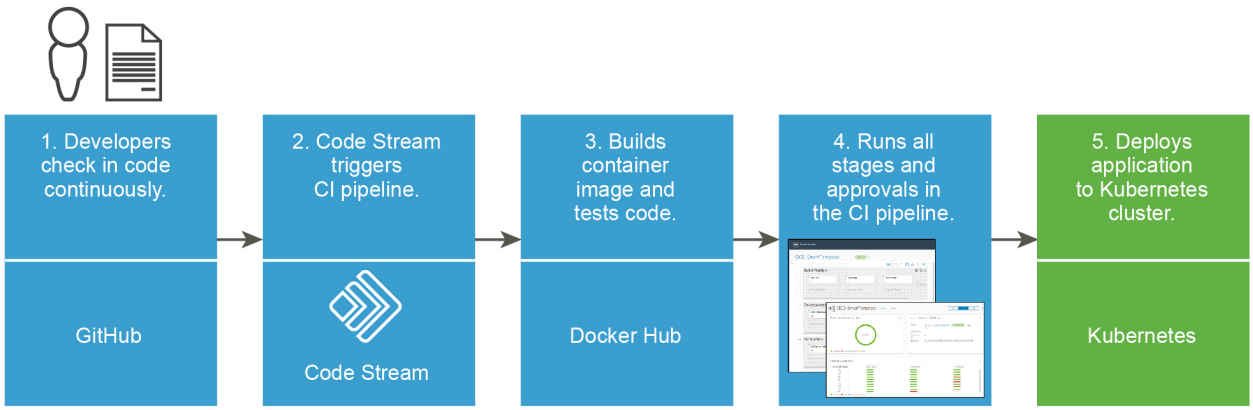
Fonctionnalités	Exemples de ce que vous pouvez faire
Utilisez la fonctionnalité de build en natif dans Code Stream.	Créez des pipelines d'intégration et de livraison continues (CICD), d'intégration continue (CI) et de livraison continue (CD) qui intègrent, placent dans un conteneur et livrent votre code en continu. ■ Utiliser un modèle de pipeline intelligent pour créer un pipeline pour vous. ■ Ajouter manuellement des étapes et des tâches à un pipeline.
Publier vos applications et automatiser les versions.	Intégrer et publier vos applications de différentes manières. ■ Intégrer continuellement votre code d'un référentiel GitHub ou GitLab à votre pipeline. ■ Intégrer un hôte Docker pour exécuter des tâches d'intégration continue, comme décrit dans cet article de blog Création d'un hôte Docker pour vRealize Automation Code Stream. ■ Automatiser le déploiement de votre application à l'aide d'un modèle de cloud YAML. ■ Automatiser le déploiement de votre application sur un cluster Kubernetes. ■ Publier votre application sur un déploiement Bleu-vert. ■ Intégrer Code Stream à vos propres outils de génération, de test et de déploiement. ■ Utiliser une REST API qui intègre Code Stream à d'autres applications.
Suivre les tendances, les mesures et les indicateurs clés de performance (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, demandez à Code Stream de créer un ticket JIRA.

Ce chapitre contient les rubriques suivantes :

- Intégration continue du code d'un référentiel GitHub ou GitLab à un pipeline dans Code Stream
- Automatisation de la publication d'une application déployée à partir d'un modèle de cloud YAML dans Code Stream
- Automatisation de la publication d'une application dans Code Stream sur un cluster Kubernetes
- Déploiement d'une application dans Code Stream vers un déploiement Bleu-vert
- Intégration d'outils de génération, de test et de déploiement propres avec Code Stream
- Utilisation des propriétés de ressource d'une tâche de modèle de cloud dans ma tâche suivante
- Utilisation d'une REST API pour intégrer Code Stream à d'autres applications
- Comment utiliser le pipeline en tant que code dans Code Stream

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

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



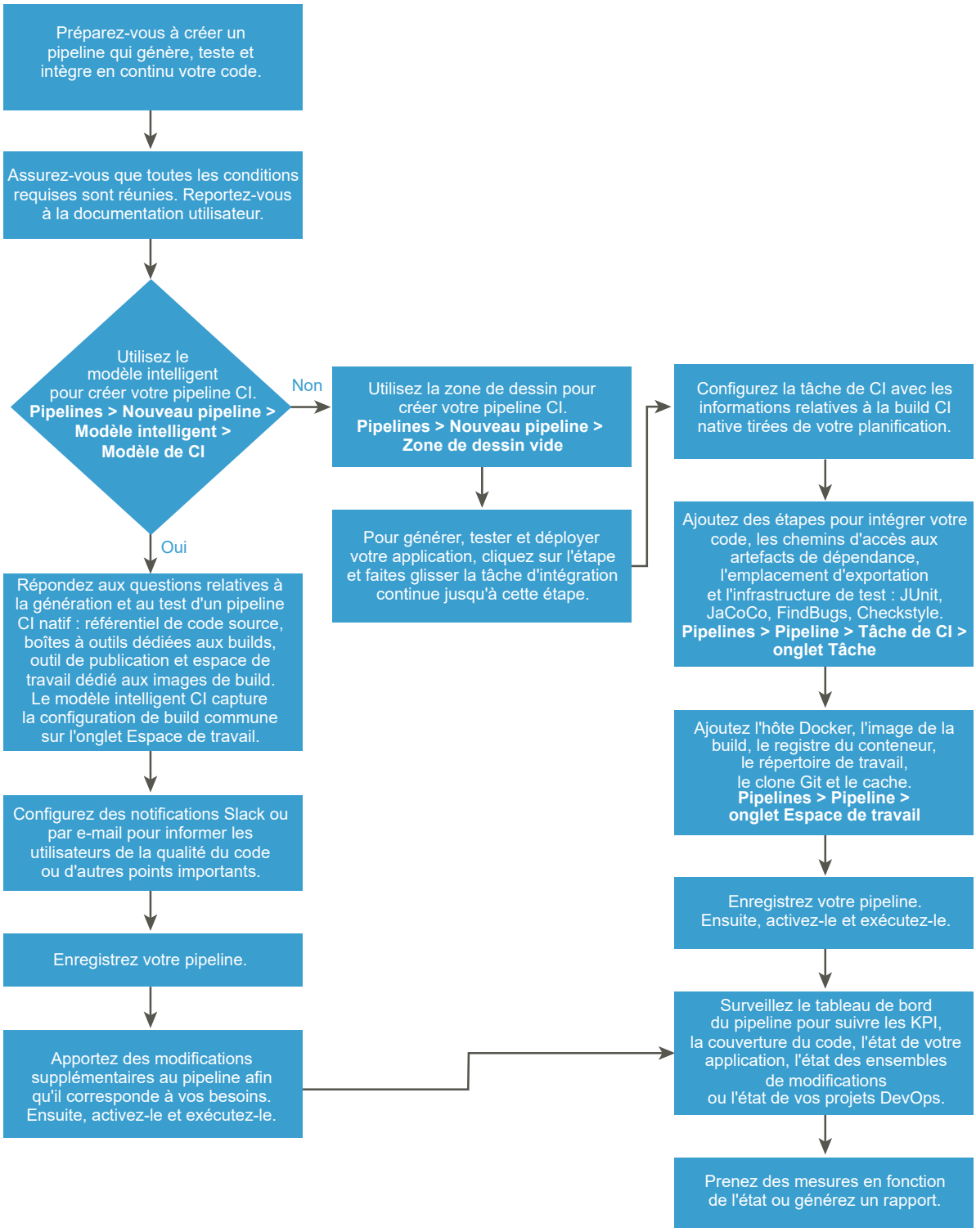
Pour que Code Stream déclenche votre pipeline dès que le code est modifié, vous utilisez le déclencheur Git. Code Stream déclenche ensuite votre pipeline à chaque fois que vous validez les modifications apportées à votre code.

L'espace de travail de pipeline Code Stream prend en charge Docker et Kubernetes pour les tâches d'intégration continue et les tâches personnalisées.

Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.

Le diagramme de flux suivant montre le workflow que vous pouvez prendre si vous utilisez un modèle de pipeline intelligent pour créer votre pipeline ou si vous créez le pipeline manuellement.

Figure 5-1. Workflow qui utilise un modèle de pipeline intelligent ou crée un pipeline manuellement



L'exemple suivant utilise un espace de travail Docker.

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 pouvez utiliser le modèle de pipeline intelligent d'intégration continue qui crée un pipeline d'intégration continue qui crée, teste et déploie votre code sur le cluster Kubernetes de votre équipe de projet sur AWS. Pour stocker les artefacts de dépendance de code pour votre tâche d'intégration continue, ce qui permet de gagner du temps dans les builds de code, vous pouvez utiliser un cache.

Dans la tâche de pipeline qui crée et teste votre code, vous pouvez inclure plusieurs étapes d'intégration continue. Ces étapes peuvent résider dans le même répertoire de travail que celui dans lequel Code Stream clone le code source lorsque le pipeline se déclenche.

Pour déployer votre code sur le cluster Kubernetes, vous pouvez utiliser une tâche Kubernetes dans votre pipeline. Vous devez ensuite activer et exécuter 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 l'exemple suivant, pour créer un pipeline d'intégration continue qui intègre continuellement votre code à votre pipeline, vous utilisez le modèle de pipeline intelligent d'intégration continue. Cet exemple utilise un espace de travail Docker.

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 CICD dans 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 d'intégration continue dans Code Stream avant d'utiliser le modèle de pipeline intelligent.
- Vérifiez qu'un référentiel de code source GitLab existe. Pour obtenir de l'aide, consultez votre administrateur Code Stream.
- Ajoutez un point de terminaison Git. Par exemple, reportez-vous à la section Utilisation du déclencheur Git dans Code Stream pour exécuter un pipeline.
- Pour que 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 Code Stream pour exécuter un pipeline.
- Ajoutez un point de terminaison hôte Docker, qui crée un conteneur pour la tâche d'intégration continue que plusieurs tâches d'intégration continue peuvent utiliser. Pour plus d'informations sur les points de terminaison, reportez-vous à la section Présentation des points de terminaison dans Code Stream .

- Obtenez l'URL de l'image, l'hôte du build et l'URL de l'image du build. Pour obtenir de l'aide, consultez votre administrateur Code Stream.
- Vérifiez que vous utilisez JUnit et JaCoCo pour vos outils de structure de test.
- Configurez une instance externe pour votre build d'intégration continue : Jenkins, TFS ou Bamboo. Le plug-in Kubernetes déploie votre code. Pour obtenir de l'aide, consultez votre administrateur Code Stream.

Procédure

1 Respectez les conditions préalables.

2 Pour créer le pipeline à l'aide du modèle de pipeline intelligent, ouvrez le modèle de pipeline intelligent d'intégration continue 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 par 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 incluez vos informations de construction d'intégration continue native à la tâche d'intégration continue.

   a Cliquez sur **Pipelines > Nouveau pipeline > Canevas vide**.

   b Cliquez sur l'étape, puis faites glisser les différentes tâches l'intégration continue du volet de navigation vers l'étape.

   c Pour configurer la tâche d'intégration continue, 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.

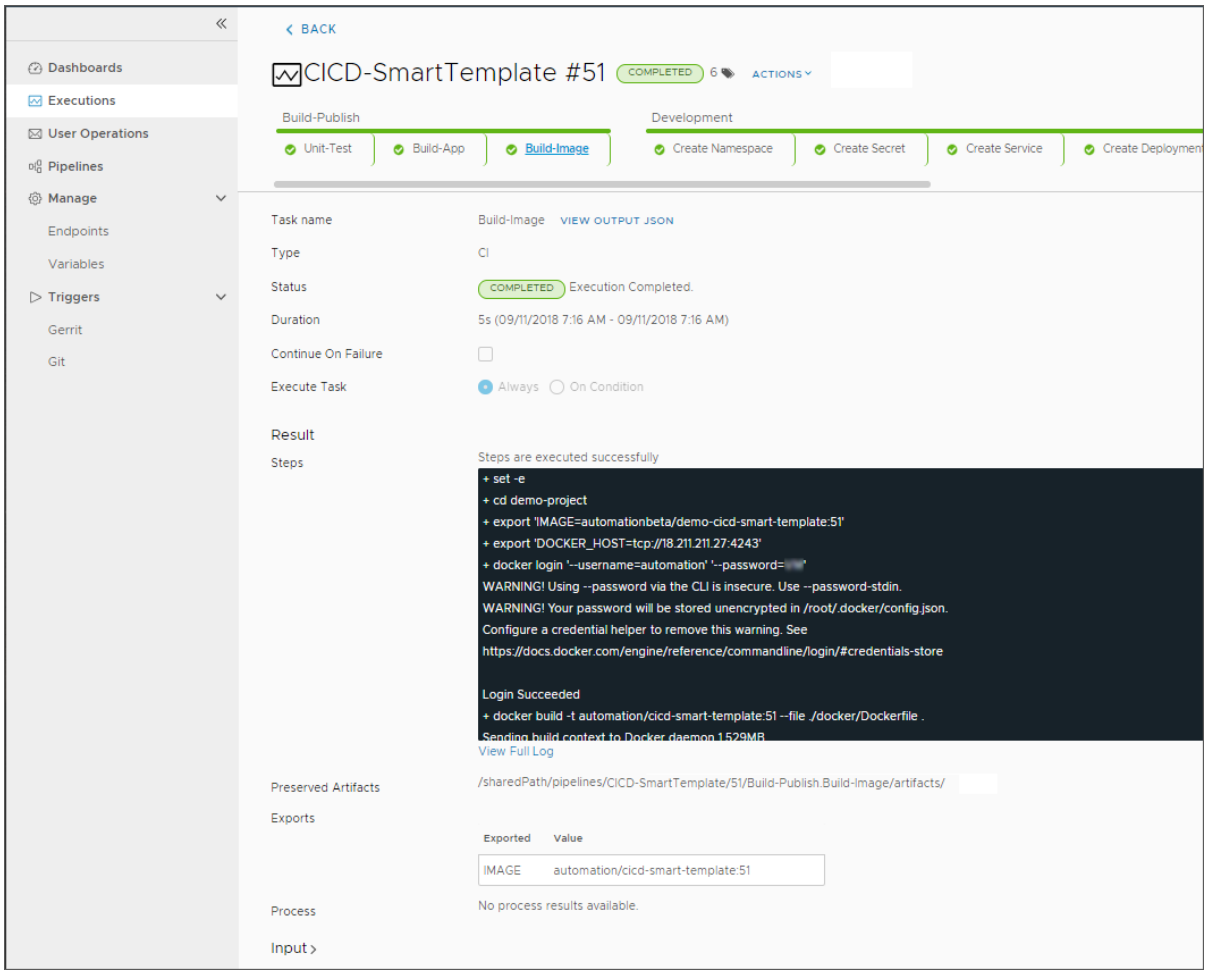
   j Enregistrez le pipeline, puis activez-le.

4 Modifiez votre code dans votre référentiel GitHub ou votre référentiel 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.



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 build.

Étape suivante

Pour plus d'informations, reportez-vous à la section Plus de ressources pour les administrateurs et les développeurs Code Stream.

Automatisation de la publication d'une application déployée à partir d'un modèle de cloud YAML dans 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. 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 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 Cloud Assembly, transférez le modèle de cloud vers votre référentiel GitHub.

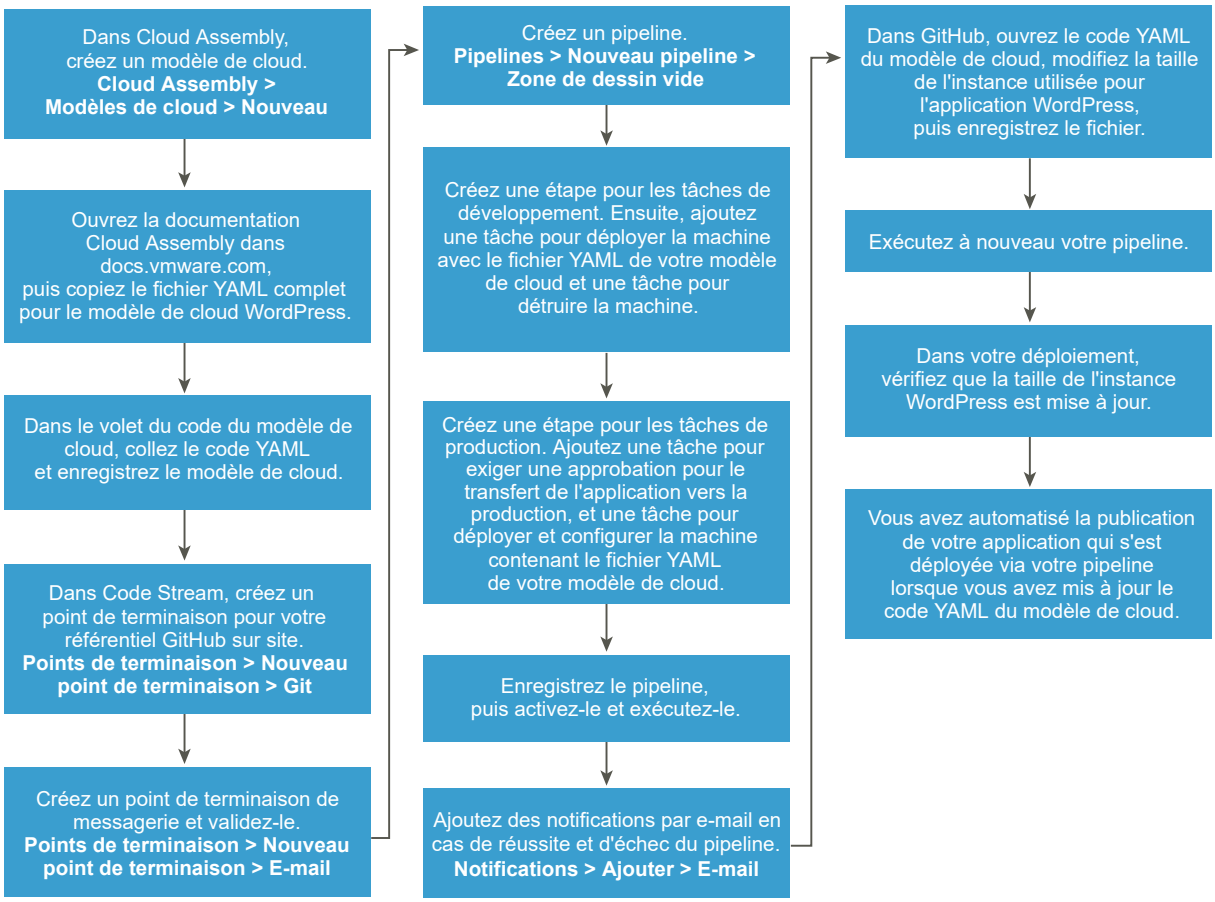
2 Dans Code Stream, créez un point de terminaison Git. Créez ensuite un Webhook Git qui utilise votre point de terminaison Git et votre pipeline.

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 Cloud Assembly, vous devez ajouter un compte de cloud, ajoutez une zone de cloud et créez le modèle de cloud.
- Dans 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

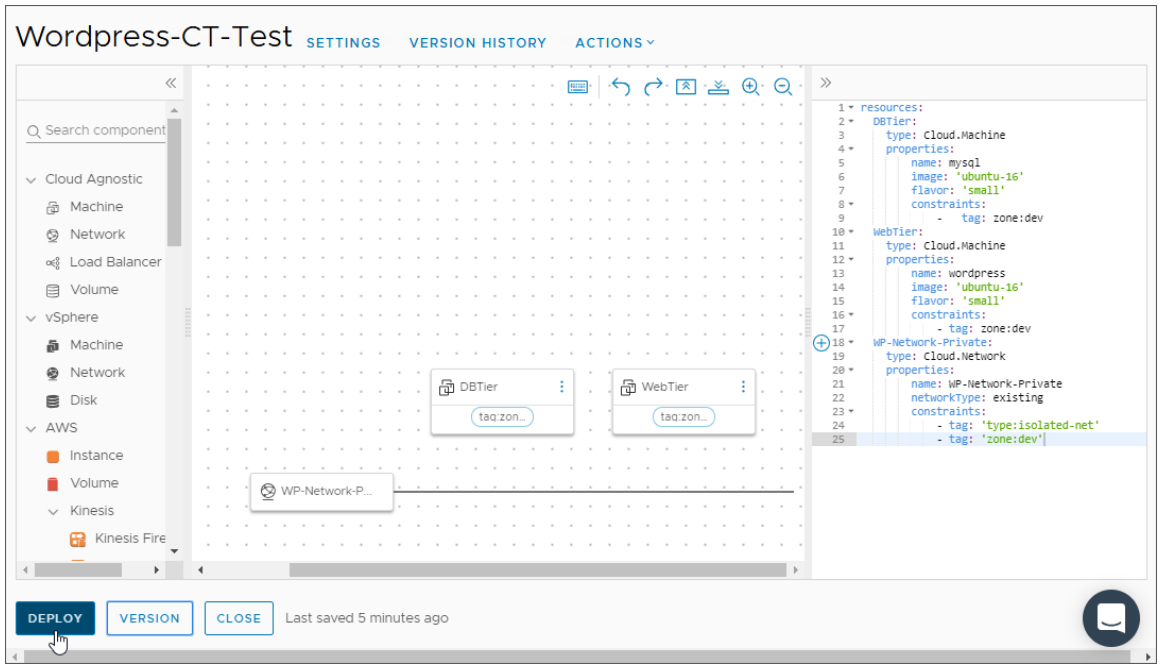
- Ajoutez un compte de cloud et une zone de cloud dans votre infrastructure vRealize Automation Cloud Assembly. Consultez la documentation de vRealize Automation Cloud Assembly.
- Pour créer votre modèle de cloud dans la procédure suivante, copiez le code YAML WordPress dans votre Presse-papiers. Consultez 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 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 Cloud Assembly.

Procédure

1 Dans Cloud Assembly, procédez comme suit.

a Cliquez sur **VMware Cloud Templates**, puis créez un modèle de cloud et un déploiement pour l'application WordPress.

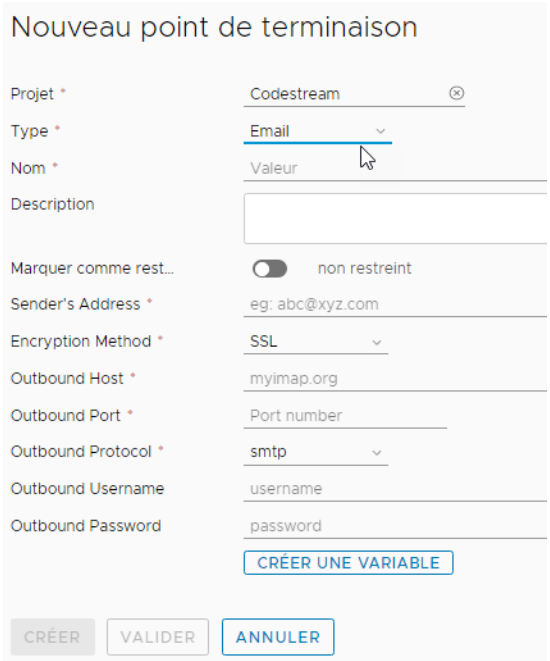
b Collez le code YAML WordPress que vous avez copié dans votre Presse-papiers dans votre modèle de cloud et déployez-le.



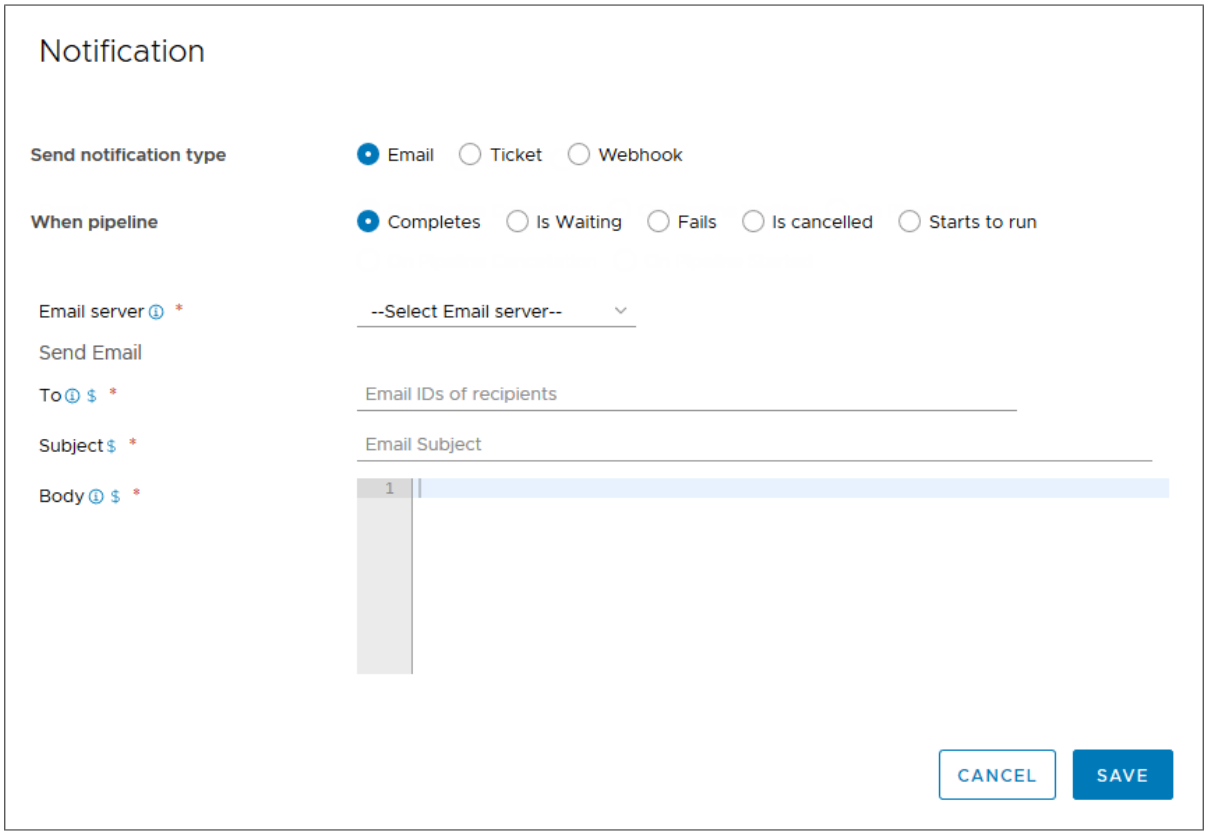
2 Dans Code Stream, créez des points de terminaison.

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 pouvant informer les utilisateurs de l'état du pipeline lors de son exécution.



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



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 utilisateur pour exiger une approbation pour le processus de production de l'application WordPress.

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.

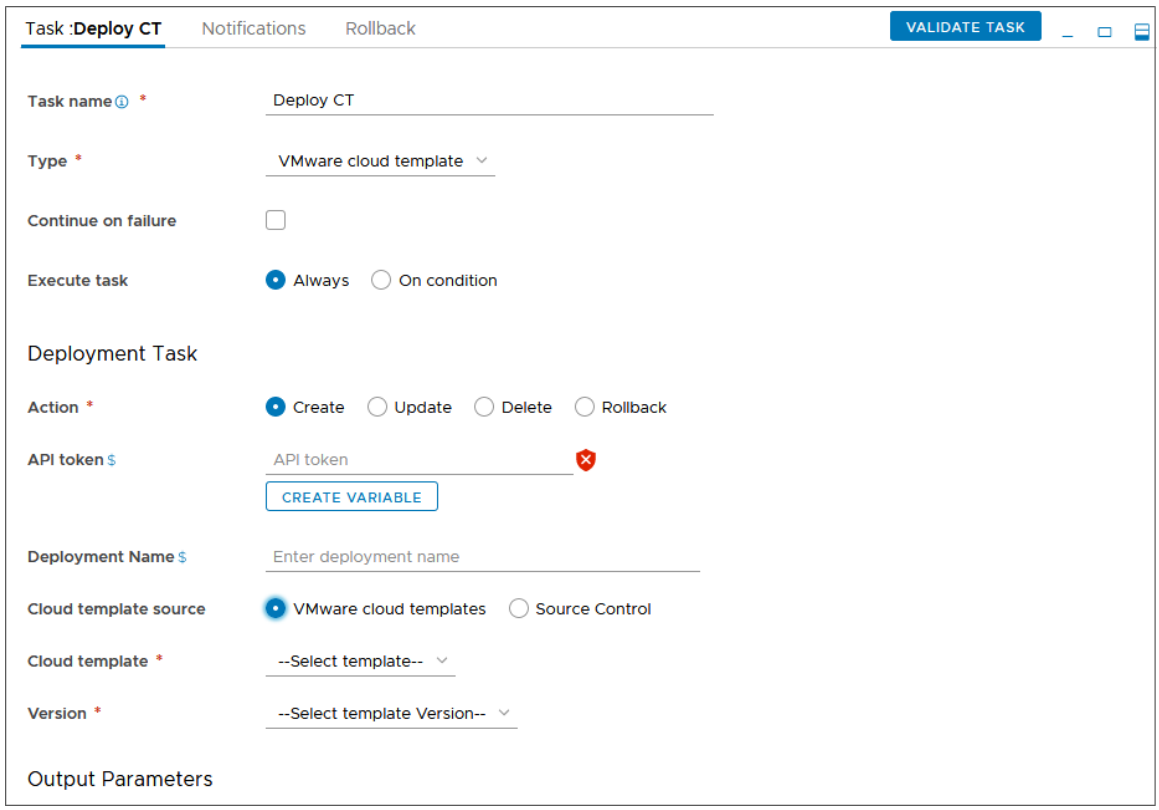
Lorsque vous sélectionnez **Créer**, le nom du déploiement doit être unique. Si vous ne renseignez pas ce champ, Code Stream lui attribuera un nom aléatoire unique.

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 Code Stream, la restauration est autorisée.

Lorsque vous vous connectez à 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 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 Code Stream, reportez-vous à la section Chapitre 5 Didacticiels d'utilisation de Code Stream.

Pour obtenir des références supplémentaires, reportez-vous à la section Plus de ressources pour les administrateurs et les développeurs Code Stream.

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

En tant qu'administrateur ou développeur Code Stream, vous pouvez utiliser 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.

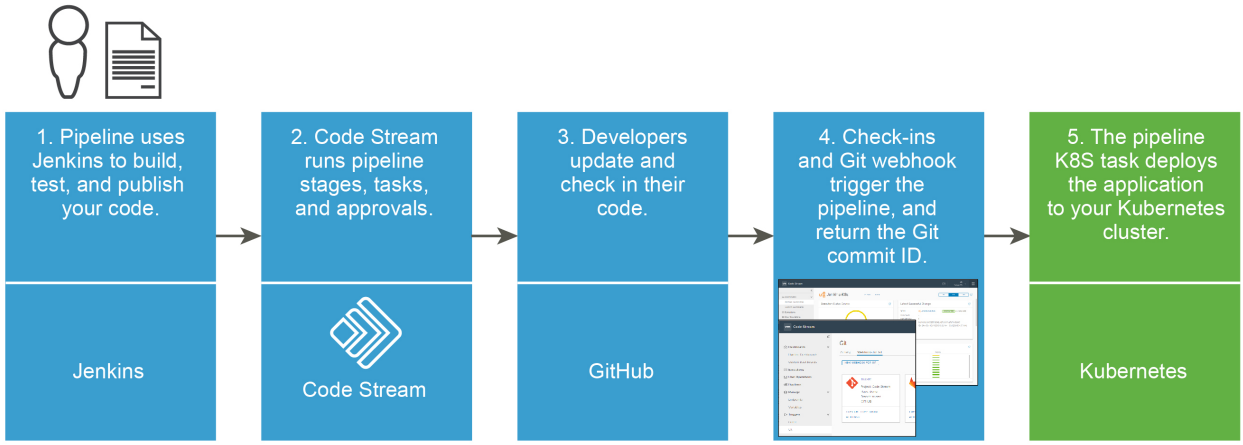
- La seconde étape concerne le déploiement. Elle exécute une tâche d'opération utilisateur qui nécessite l'approbation des utilisateurs clés pour que le pipeline puisse déployer votre application sur votre cluster Kubernetes.

Lors de l'utilisation d'un point de terminaison d'API Kubernetes dans l'espace de travail de pipeline, Code Stream crée les ressources Kubernetes nécessaires, telles que ConfigMap, Secret et Pod, pour exécuter la tâche d'intégration continue (CI) ou la tâche personnalisée. Code Stream communique avec le conteneur à l'aide du port de nœud.

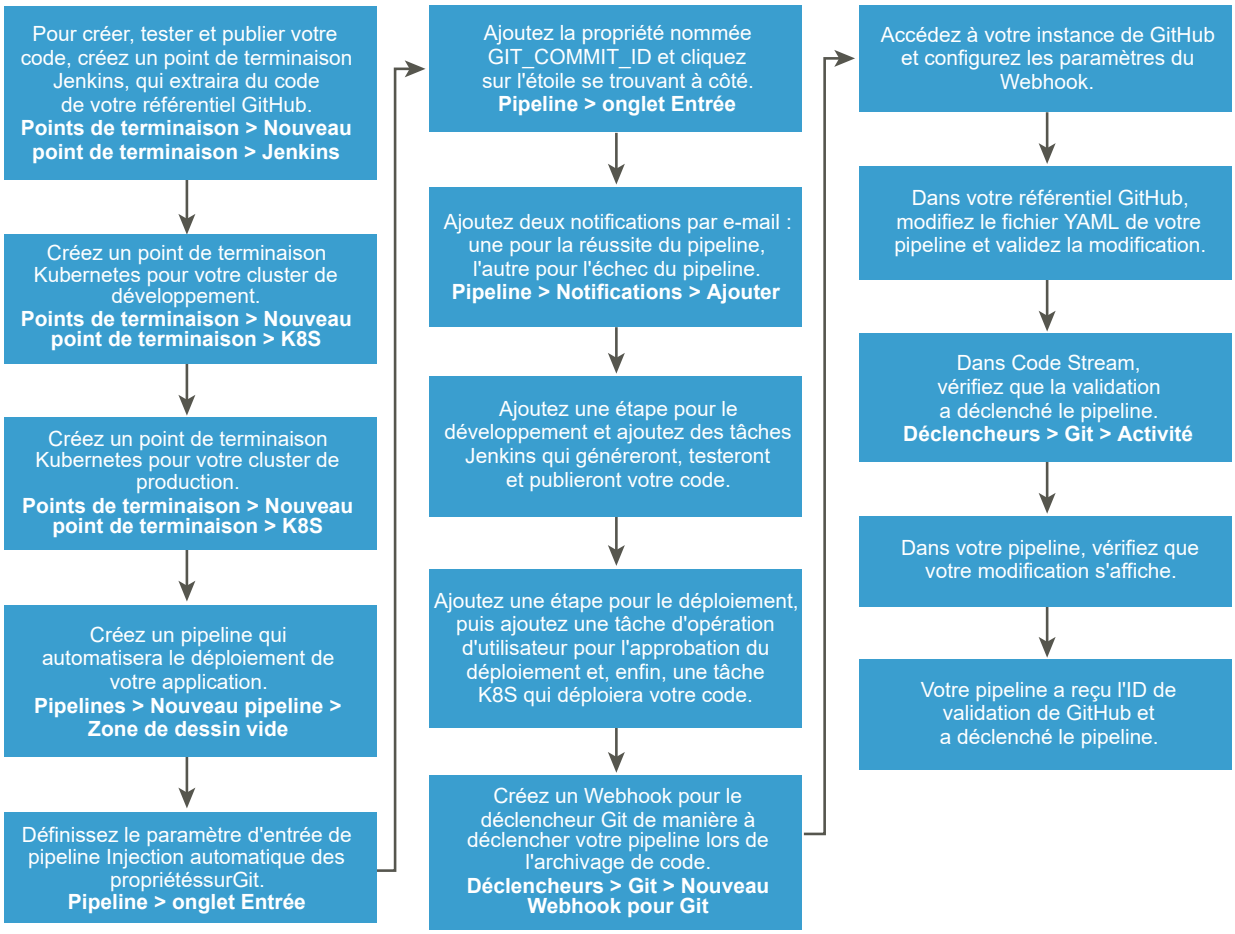
Pour partager des données entre des exécutions de pipeline, vous devez fournir une réclamation de volume persistant, et Code Stream montera la réclamation de volume persistant sur le conteneur pour stocker les données et l'utilisera pour les exécutions de pipeline suivantes.

L'espace de travail de pipeline Code Stream prend en charge Docker et Kubernetes pour les tâches d'intégration continue et les tâches personnalisées.

Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.



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 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 Code Stream et d'un hôte Docker, reportez-vous à :

- Planification d'une build native CICD dans Code Stream avant d'utiliser le modèle de pipeline intelligent
- Planification d'une build native CICD dans Code Stream avant d'ajouter des tâches manuellement

Conditions préalables

- Vérifiez que le code d'application à déployer réside 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 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 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.

   L'URL de votre cluster Kubernetes peut ou non inclure un numéro de port.

   Par exemple :

   `https://10.111.222.333:6443`

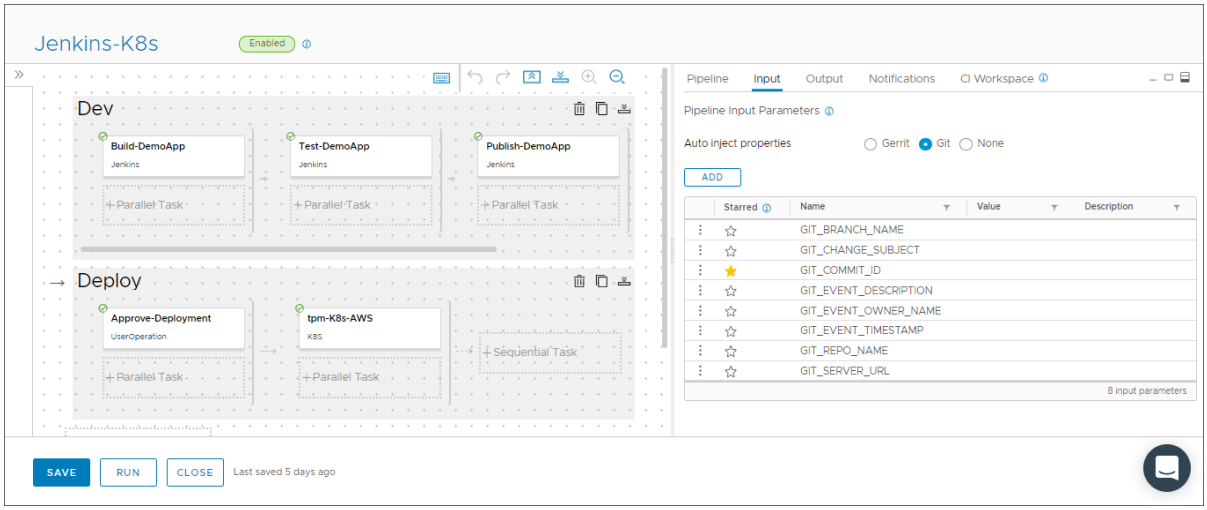
   `https://api.kubernetesserver.fa2c1d78-9f00-4e30-8268-4ab81862080d.k8s-user.com`

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 en regard de celle-ci.

   Au cours de l'exécution du pipeline, l'ID de validation renvoyé par le déclencheur Git est affiché.

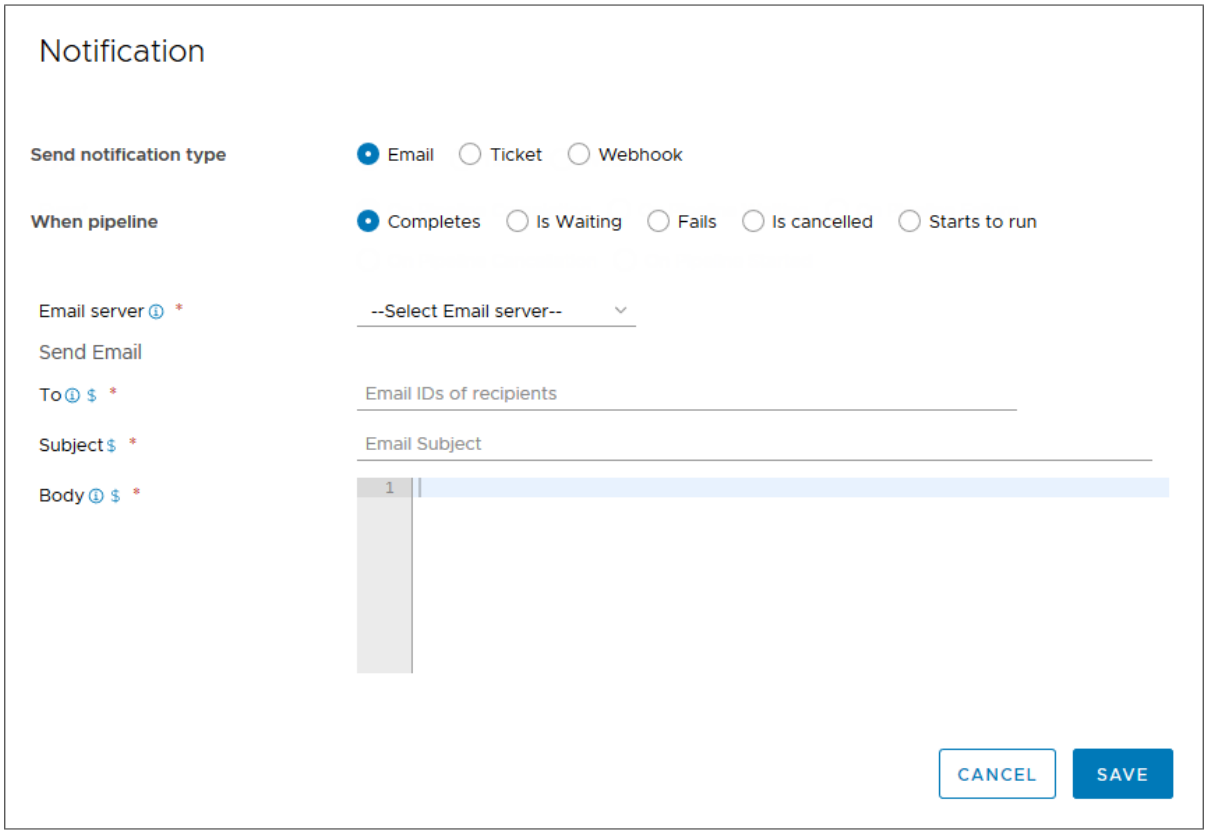


4 Ajoutez des notifications pour envoyer un e-mail en cas de réussite ou d'échec du pipeline.

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**.

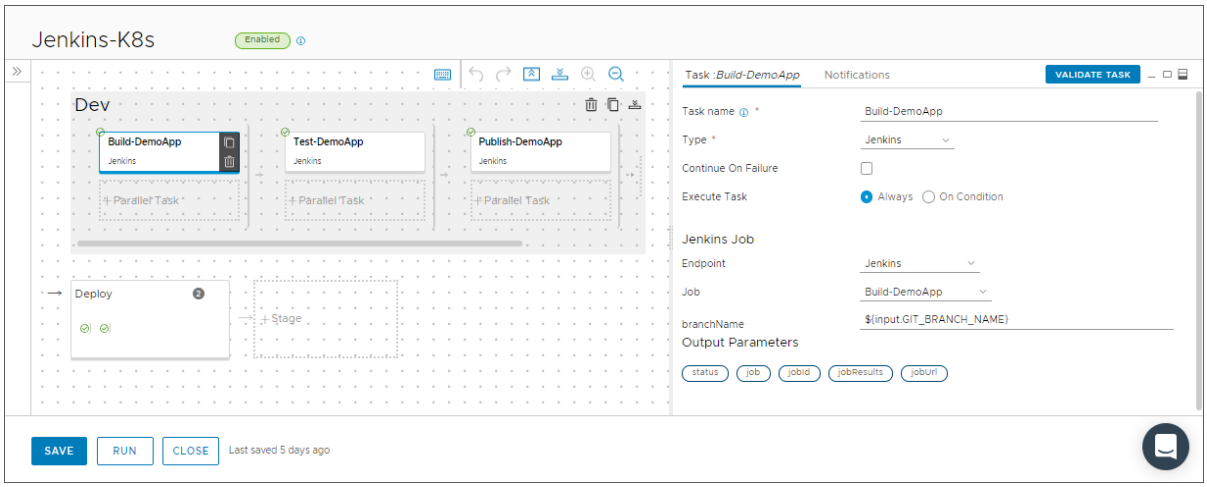


5 Ajoutez une étape de développement à votre pipeline et ajoutez des tâches de création, de test et de publication de votre application. Ensuite, validez chaque tâche.

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.

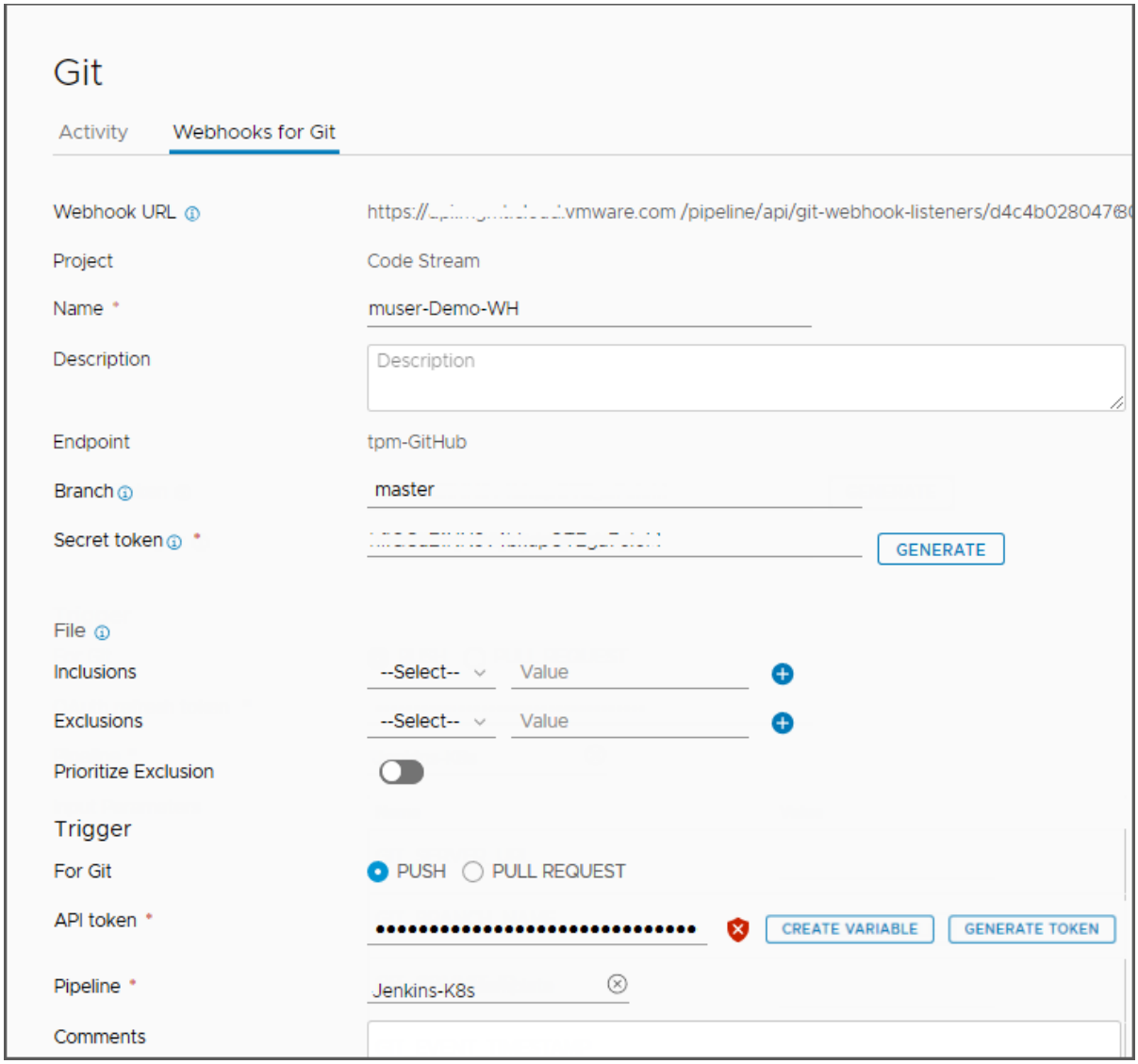


6 Ajoutez une étape de déploiement à votre pipeline. Ajoutez ensuite une tâche qui nécessite une approbation pour le déploiement de votre application, puis une autre tâche qui déploie l'application sur votre cluster Kubernetes. Ensuite, validez chaque tâche.

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 Code Stream d'utiliser le déclencheur Git, qui déclenche votre pipeline lorsque les développeurs valident leur code.



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 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:
eyJzeW1waG9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJ1c2VybmFtZSI6InRhbmdvLWJldGEyIiwicGFzc3dvcmQiOi
JhRGstcmVOLW1UQi1IejciLCJlbWFpbCI6InRhbmdvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoiZEdGdVoyOHRZbVYw
WVRJNllVUnJMWEpsVGkxdFZFSXRTSG8zIn19
kind: Secret
metadata:
  name: jfrog
  namespace: ${input.GIT_BRANCH_NAME}
type: kubernetes.io/dockercfg
---
apiVersion: v1
kind: Service
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  ports:
    - port: 80
  selector:
    app: codestream
    tier: frontend
  type: LoadBalancer
---
apiVersion: extensions/v1
kind: Deployment
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  selector:
    matchLabels:
      app: codestream
```

```
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: codestream
        tier: frontend
    spec:
      containers:
      - name: codestream
        image: cas.jfrog.io/codestream:${input.GIT_BRANCH_NAME}-${Dev.PublishApp.output.jobId}
        ports:
        - containerPort: 80
          name: codestream
      imagePullSecrets:
      - name: jfrog
```

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 Code Stream à Jenkins, reportez-vous à la section Intégration de Code Stream à Jenkins.

Déploiement d'une application dans 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 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 Code Stream représente votre modèle de déploiement Bleu-vert en deux étapes. Une étape est consacrée au développement, l'autre étape à la production.

L'espace de travail de pipeline Code Stream prend en charge Docker et Kubernetes pour les tâches d'intégration continue et les tâches personnalisées.

Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.

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 réplicas 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 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 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 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 Code Stream avant d'utiliser le modèle de pipeline intelligent.

Procédure

1 Cliquez sur **Pipelines > Nouveau pipeline > Modèle intelligent > Modèle CI/CD**.

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 Code Stream avant d'utiliser le modèle de pipeline intelligent.

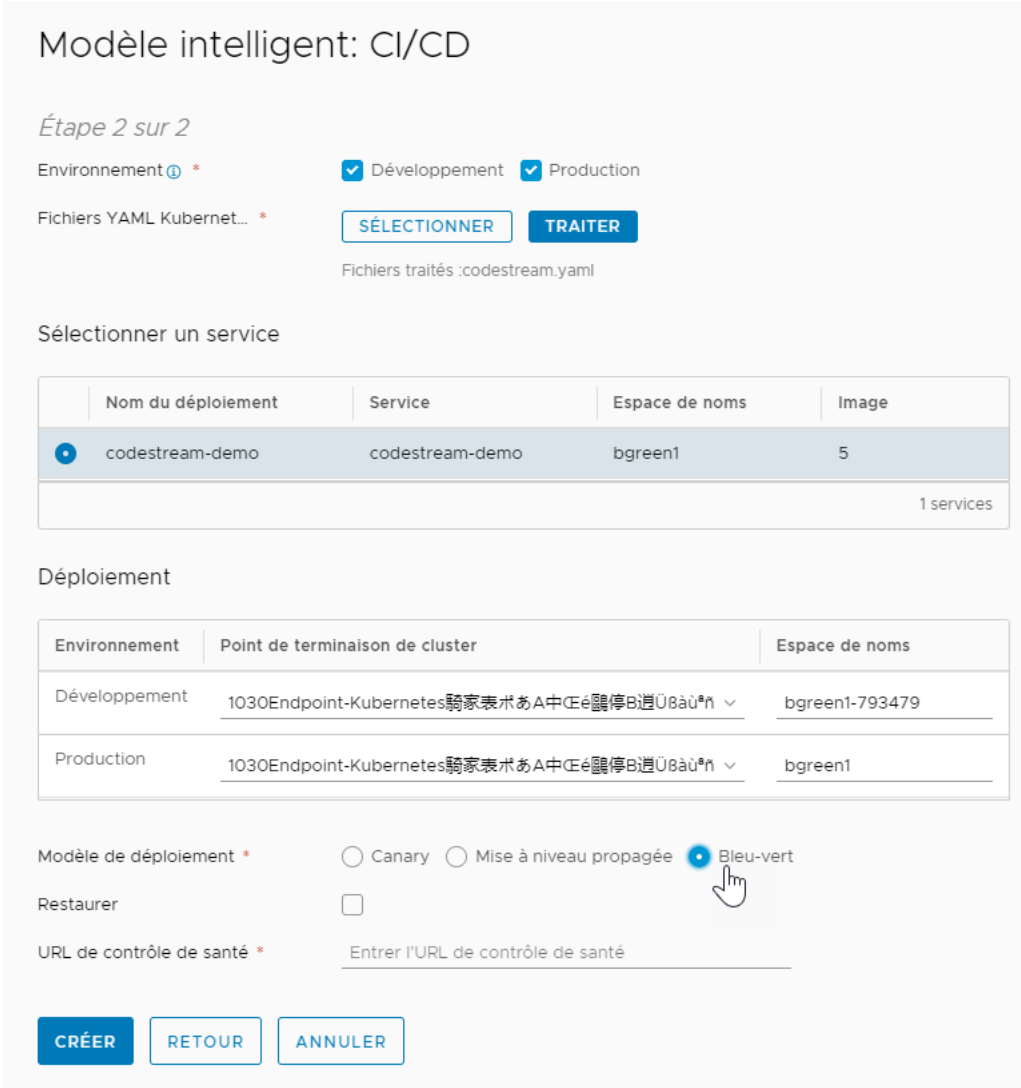
3 Renseignez la partie CD du modèle de pipeline intelligent.

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 Dev et Prod.

d Pour le modèle de déploiement Production, sélectionnez **Bleu-vert** , puis cliquez sur **Créer**.



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 qui créent l'espace de noms, le service et le déploiement. Si vous devez télécharger une image à partir d'un référentiel privé, le fichier YAML doit inclure une section avec le secret de configuration Docker. Reportez-vous à la partie CD de la section Planification d'une build native CICD dans Code Stream avant d'utiliser le modèle de pipeline intelligent.

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 Code Stream, reportez-vous à la section Chapitre 5 Didacticiels d'utilisation de Code Stream.

Pour restaurer un déploiement, reportez-vous à la section Restauration d'un déploiement dans Code Stream.

Pour obtenir des références supplémentaires, reportez-vous à la section Plus de ressources pour les administrateurs et les développeurs Code Stream.

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

En tant qu'administrateur ou développeur DevOps, vous pouvez créer des scripts personnalisés qui étendent la capacité de Code Stream.

Avec votre script, vous pouvez intégrer 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 l'intégration de vos outils de génération, de test et de déploiement à Code Stream. Par exemple, votre script peut utiliser l'espace de travail de votre pipeline pour prendre en charge les tâches d'intégration continue qui génèrent et testent votre application, ainsi que les tâches de livraison continue qui la déploient. Il peut envoyer un message à Slack lorsqu'un pipeline se termine, et bien plus encore.

L'espace de travail de pipeline Code Stream prend en charge Docker et Kubernetes pour les tâches d'intégration continue et les tâches personnalisées.

Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.

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 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.	La tâche personnalisée : ■ s'exécute sur le même conteneur que les autres tâches CI de votre pipeline. ■ inclut les variables d'entrée et de sortie que votre script remplit avant l'exécution de la tâche personnalisée par le pipeline. ■ prend en charge plusieurs types de données et différents types de métadonnées que vous définissez comme entrées et sorties dans votre script.
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 Code Stream.
Une fois votre pipeline exécuté, observation des exécutions.	Vérifiez que le pipeline a fourni les résultats attendus.

Lorsque vous utilisez une tâche personnalisée qui appelle une version d'intégration personnalisée, vous pouvez inclure des variables d'environnement personnalisées sous forme de paires nom-valeur dans l'onglet **Espace de travail** du pipeline. Lorsque l'image de générateur crée le conteneur d'espace de travail qui exécute la tâche CI et déploie votre image, Code Stream transmet les variables d'environnement à ce conteneur.

Par exemple, lorsque votre instance de Code Stream nécessite un proxy Web et que vous utilisez un hôte Docker pour créer un conteneur pour une intégration personnalisée, Code Stream exécute le pipeline et transmet les variables de configuration de proxy Web à ce conteneur.

Tableau 5-5. Exemple de paires nom-valeur de variable d'environnement

Nom	Valeur
HTTPS_PROXY	http://10.0.0.255:1234
https_proxy	http://10.0.0.255:1234
NO_PROXY	10.0.0.32, *.dept.vsphere.local
no_proxy	10.0.0.32, *.dept.vsphere.local
HTTP_PROXY	http://10.0.0.254:1234

Tableau 5-5. Exemple de paires nom-valeur de variable d'environnement (suite)

Nom	Valeur
http_proxy	http://10.0.0.254:1234
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Les paires nom-valeur s'affichent dans l'interface utilisateur de la manière suivante :

Cet exemple crée une intégration personnalisée qui connecte 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 suivants : 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.

   a Cliquez sur **Intégrations personnalisées > Nouveau** et entrez un nom pertinent.

   b Sélectionnez l'environnement d'exécution préféré.

   c 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
runtime	Environnement d'exécution des tâches où Code Stream exécute le code, qui peut être l'une de ces chaînes insensibles à la casse : ■ nodejs ■ python2 ■ python3 ■ shell Si rien n'est indiqué, shell est la valeur par défaut supposée.
code	Logique métier personnalisée à exécuter dans le cadre de la tâche personnalisée.
inputProperties	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.
outputProperties	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
`type`	Types d'entrée à rendre : ■ **`text`** ■ **`textarea`** ■ **`number`** ■ **`checkbox`** ■ **`password`** ■ **`select`**	Oui
`name`	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
`title`	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, **`name`** est utilisé par défaut.	Non
`required`	Détermine si un utilisateur doit entrer la propriété d'entrée lorsqu'il configure la tâche personnalisée. Définissez sur true ou false. Lorsque la valeur est true, 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
`placeHolder`	Texte par défaut pour la zone d'entrée de la propriété d'entrée lorsqu'aucune valeur n'est présente. Correspond à l'attribut html placeholder. Uniquement pris en charge pour certains types de propriétés d'entrée.	Non

Touches d'entrée YAML de tâche personnalisée	Description	Requis
`defaultValue`	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
`bindable`	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 **$** en regard du titre. Uniquement pris en charge pour certains types de propriétés d'entrée.	Non
`labelMessage`	Chaîne qui agit comme une info-bulle d'aide pour les utilisateurs. Ajoute une icône d'info-bulle **i** en regard du titre d'entrée.	Non
`enum`	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. Lorsqu'un utilisateur sélectionne une option et l'enregistre pour la tâche personnalisée, la valeur de **inputProperty** correspond à cette valeur et figure dans la modélisation de tâche personnalisée. Par exemple, la valeur 2015. ■ 2015 ■ 2016 ■ 2017 ■ 2018 ■ 2019 ■ 2020	Non
`options`	Prend un tableau d'objets en utilisant **optionKey** et **optionValue**. ■ **optionKey**. Valeur propagée à la section code de la tâche. ■ **optionValue**. Chaîne qui affiche l'option dans l'interface utilisateur. Uniquement pris en charge pour certains types de propriétés d'entrée. Options : **optionKey** : key1. Lorsque cette option est sélectionnée et enregistrée pour la tâche personnalisée, la valeur de inputProperty correspond à **key1** dans la section code. **optionValue** : « Label for 1 ». Valeur d'affichage de **key1** dans l'interface utilisateur et ne figure nulle part ailleurs pour la tâche personnalisée. **optionKey** : key2 **optionValue** : « Label for 2 » **optionKey** : key3 **optionValue** : « Label for 3 »	Non

Touches d'entrée YAML de tâche personnalisée	Description	Requis
minimum	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
maximum	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 5-6. 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
■ Chaîne ■ Texte ■ Liste : sous forme de liste de n'importe quel type ■ Carte : sous la forme map[chaîne]any ■ Sécurisé : rendu sous forme de zone de texte pour mot de passe, chiffrée lorsque vous enregistrez la tâche personnalisée ■ Nombre ■ Booléen : s'affiche sous forme de zones de texte ■ URL : identique au type Chaîne, avec validation supplémentaire ■ Sélection, case d'option	■ type : Une chaîne | Texte… ■ valeur par défaut : valeur par défaut ■ options : liste ou carte d'options à utiliser avec la sélection ou la case d'option ■ min. : valeur ou taille minimale ■ max. : valeur ou taille maximale ■ titre : nom détaillé de la zone de texte ■ espace réservé : espace réservé dans l'interface utilisateur ■ description : devient une info-bulle

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 Code Stream prend en charge pour chaque **outputProperty**.

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

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 pointeur 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 **Version** pour pouvoir sélectionner le script dans votre tâche personnalisée.

d Pour créer la version, cliquez sur **Créer**.

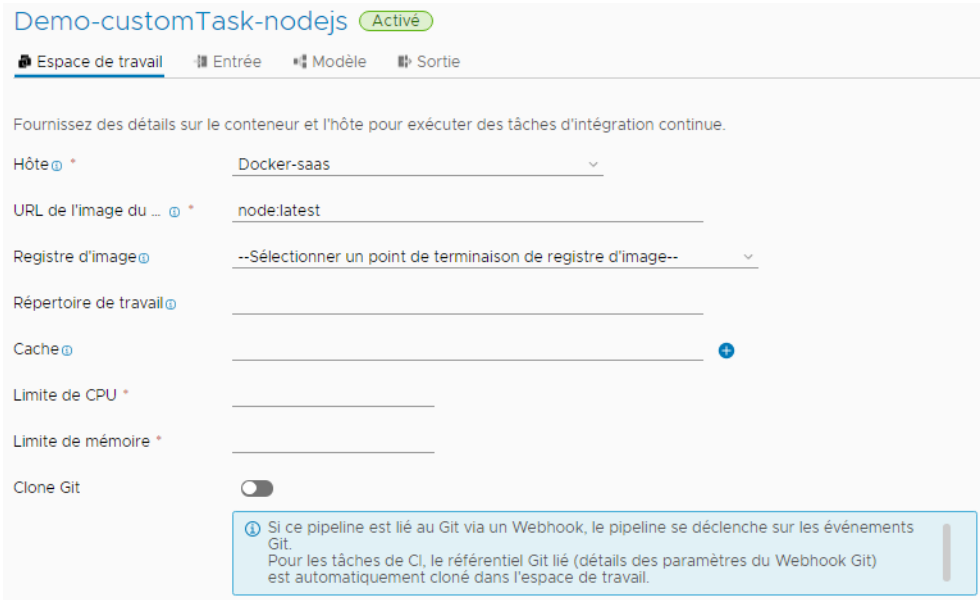
8 Pour enregistrer le script, cliquez sur **Enregistrer**.

9 Dans votre pipeline, configurez l'espace de travail.

Cet exemple utilise un espace de travail Docker.

a Cliquez sur l'onglet **Espace de travail**.

b Sélectionnez l'hôte Docker et l'URL de l'image du générateur.



10 Ajoutez une tâche personnalisée à votre pipeline et configurez-la.

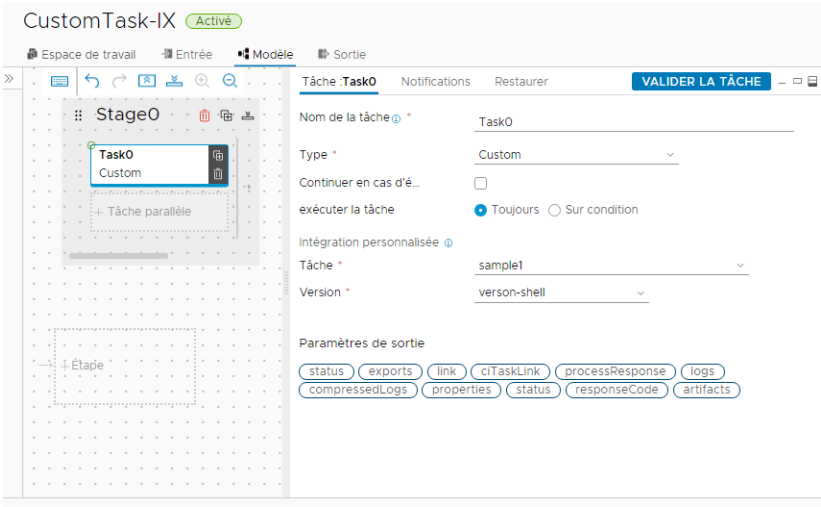
a Cliquez sur l'onglet **Modèle**.

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.

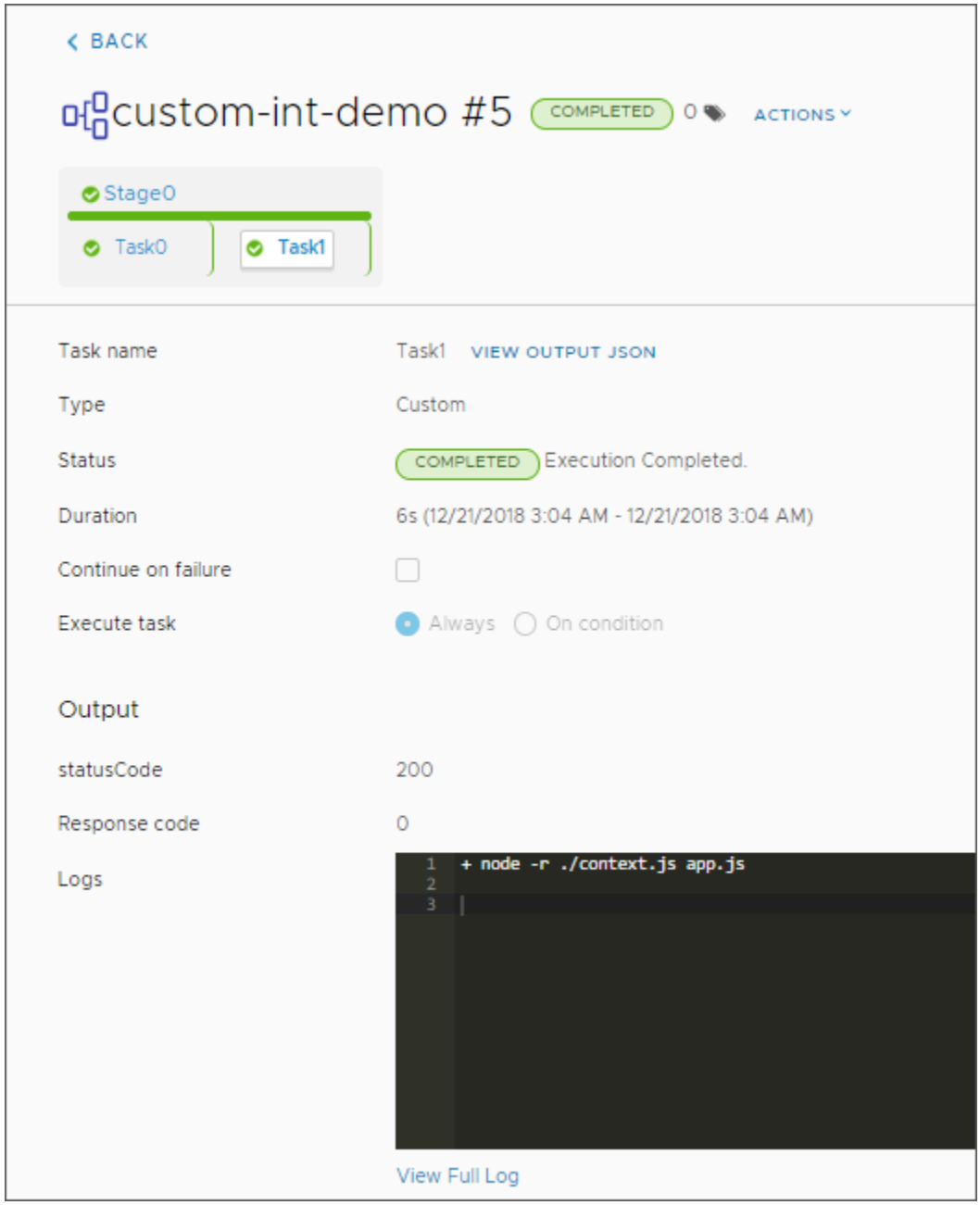
a Cliquez sur **Exécuter**.

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 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 Code Stream dans l'automatisation du cycle de vie de votre version logicielle.

Utilisation des propriétés de ressource d'une tâche de modèle de cloud dans ma tâche suivante

Lorsque vous utilisez une tâche de modèle de cloud dans Code Stream, une question fréquente concerne la manière d'utiliser la sortie de cette tâche dans une tâche ultérieure de votre pipeline. Pour utiliser la sortie d'une tâche de modèle de cloud, telle qu'une machine de cloud, vous devez savoir comment trouver les propriétés de ressource dans les détails du déploiement de la tâche de modèle de cloud et l'adresse IP de la machine de cloud.

Par exemple, les détails de déploiement d'un modèle VMware Cloud incluent la ressource de machine de cloud et son adresse IP. Dans votre pipeline, vous pouvez utiliser la machine de cloud et l'adresse IP en tant que variable pour lier une tâche de modèle de cloud à une tâche REST.

La méthode que vous utilisez pour trouver l'adresse IP de la machine de cloud n'est pas classique, car le déploiement du modèle VMware Cloud doit se terminer avant que les détails du déploiement ne soient disponibles. Ensuite, vous pouvez utiliser les ressources du déploiement de modèle VMware Cloud pour lier vos tâches de pipeline.

- Les propriétés de ressource qui s'affichent dans une tâche de modèle de cloud dans votre pipeline sont définies dans le modèle VMware Cloud de Cloud Assembly.
- Vous ne savez pas nécessairement à quel moment un déploiement de ce modèle de cloud est terminé.
- Une tâche de modèle de cloud dans Code Stream peut uniquement afficher les propriétés de sortie du modèle VMware Cloud une fois le déploiement terminé.

Cet exemple peut être particulièrement utile si vous déployez une application et que vous appelez plusieurs API. Par exemple, si vous utilisez une tâche de modèle de cloud qui appelle un modèle VMware Cloud, qui déploie une application WordPress avec REST API, l'adresse IP de la machine déployée figure dans les détails du déploiement et vous pouvez utiliser l'API pour la tester.

La tâche de modèle de cloud vous permet d'utiliser une liaison de variable en affichant les détails de remplissage automatique du type. Il s'agit du mode de liaison de la variable.

Cet exemple montre comment :

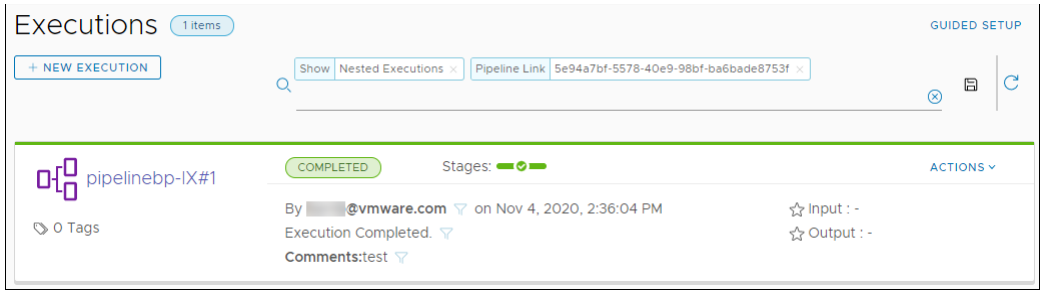
- Rechercher les détails du déploiement et les propriétés de ressource de votre tâche de modèle de cloud dans un pipeline qui s'est exécuté correctement.
- Rechercher l'adresse IP de la machine de cloud dans la section des ressources des détails du déploiement.
- Ajouter une tâche REST après la tâche de modèle de cloud dans votre pipeline.
- Lier la tâche du modèle de cloud à la tâche REST en utilisant l'adresse IP de la machine de cloud dans l'URL de la tâche REST.
- Exécutez votre pipeline et observez le travail de liaison de la tâche de modèle de cloud à la tâche REST.

Conditions préalables

- Vérifiez que vous disposez d'un modèle VMware Cloud opérationnel faisant l'objet d'une gestion des versions.
- Vérifiez que le déploiement du modèle VMware Cloud a réussi dans Cloud Assembly.
- Vérifiez que vous disposez d'un pipeline qui inclut une tâche de modèle de cloud qui utilise ce modèle VMware Cloud.
- Vérifiez que votre pipeline s'est exécuté correctement.

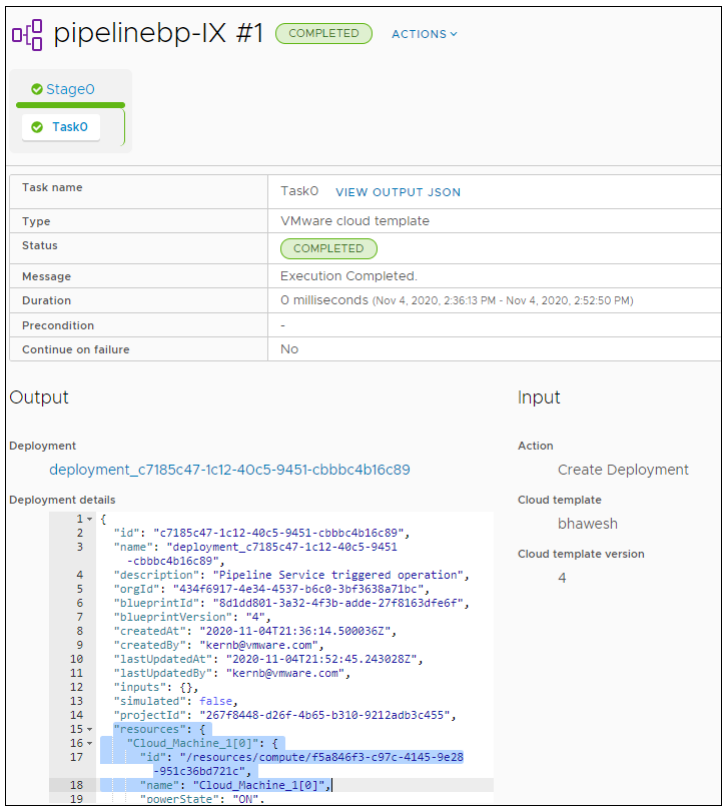
Procédure

1. Dans votre pipeline, localisez l'adresse IP de la machine de cloud dans la section de ressources des détails de déploiement de la tâche de modèle de cloud.
   a. Cliquez sur **Actions > Afficher les exécutions**.
   b. Dans un pipeline exécuté correctement, cliquez sur le lien vers l'exécution du pipeline.



c Sous le nom du pipeline, cliquez sur le lien vers la **Tâche**.

d Dans la zone Sortie, recherchez les détails du déploiement.

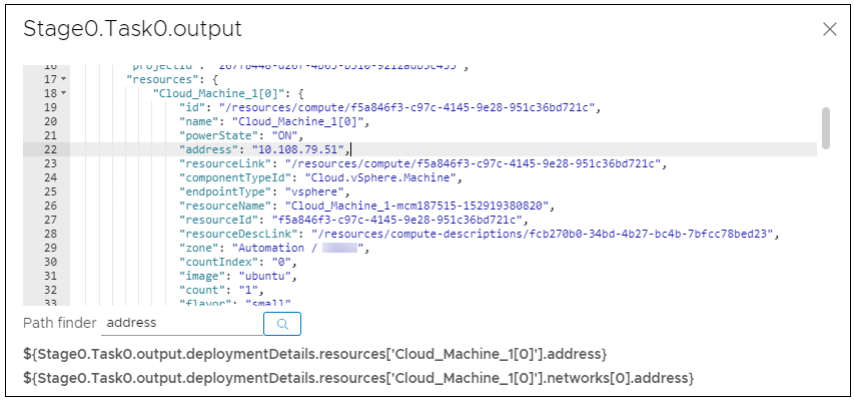


e Dans la section des ressources des détails du déploiement, recherchez le nom de la machine de cloud.

Vous inclurez la syntaxe du nom de la machine de cloud dans l'URL de votre tâche REST.

f Pour trouver l'expression de liaison pour la propriété de sortie de la tâche de modèle de cloud, cliquez sur **AFFICHER LE JSON DE SORTIE**, recherchez la propriété d'adresse et localisez l'adresse IP de la machine de cloud.

L'expression de liaison figure sous la propriété et l'icône de recherche dans la sortie JSON.



La propriété de ressource d'adresse affiche l'adresse IP de la machine de cloud. Par exemple :

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

2 Revenez au modèle de pipeline et entrez l'URL dans la tâche REST.

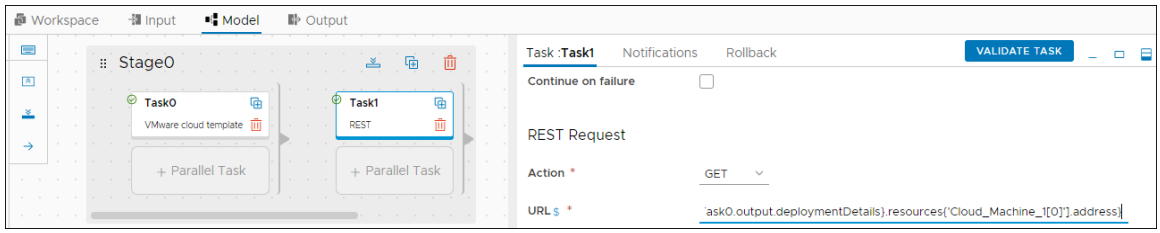
a Cliquez sur **Actions > Afficher le pipeline**.

b Cliquez sur la tâche REST.

c Dans la zone d'URL Demande REST, entrez **$**, sélectionnez **Étape**, **Tâche**, **sortie**, **deploymentDetails** et entrez **`ressources`**.

La possibilité de saisir le texte avec la fonction de saisie automatique est disponible jusqu'au moment où vous devez entrer **`ressources`**.

d Entrez le reste des ressources de machine de cloud à partir des détails du déploiement sous la forme : **`{'Cloud_Machine_1[0]'].address}`**



Pour l'entrée de la machine de cloud, vous devez utiliser la notation à crochets, comme indiqué.

Le format d'URL complet est : `$ {Stage0.Task0.output.deploymentDetails.resources{'Cloud_Machine_1[0]'].address}`

3 Exécutez le pipeline et regardez la tâche REST utiliser la machine de cloud et l'adresse IP de la sortie de votre tâche de modèle de cloud comme URL à tester.

Résultats

Félicitations ! Vous avez trouvé le nom et l'adresse IP de la machine de cloud dans les détails de déploiement et la sortie JSON d'un modèle de cloud, puis vous les avez utilisés pour lier votre sortie de tâche de modèle de cloud à l'entrée d'URL de la tâche REST dans le pipeline.

Étape suivante

Continuez à explorer l'utilisation de variables de liaison provenant de ressources dans la tâche de modèle de cloud avec d'autres tâches dans le pipeline.

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

Code Stream fournit un plug-in REST qui vous permet d'intégrer Code Stream à d'autres applications qui utilisent une REST API 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 Code Stream et une autre application.

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

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

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

Tableau 5-7. Préparation d'un pipeline pour communiquer sur REST API

Actions	Résultat obtenu
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 REST API 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 REST API 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**.

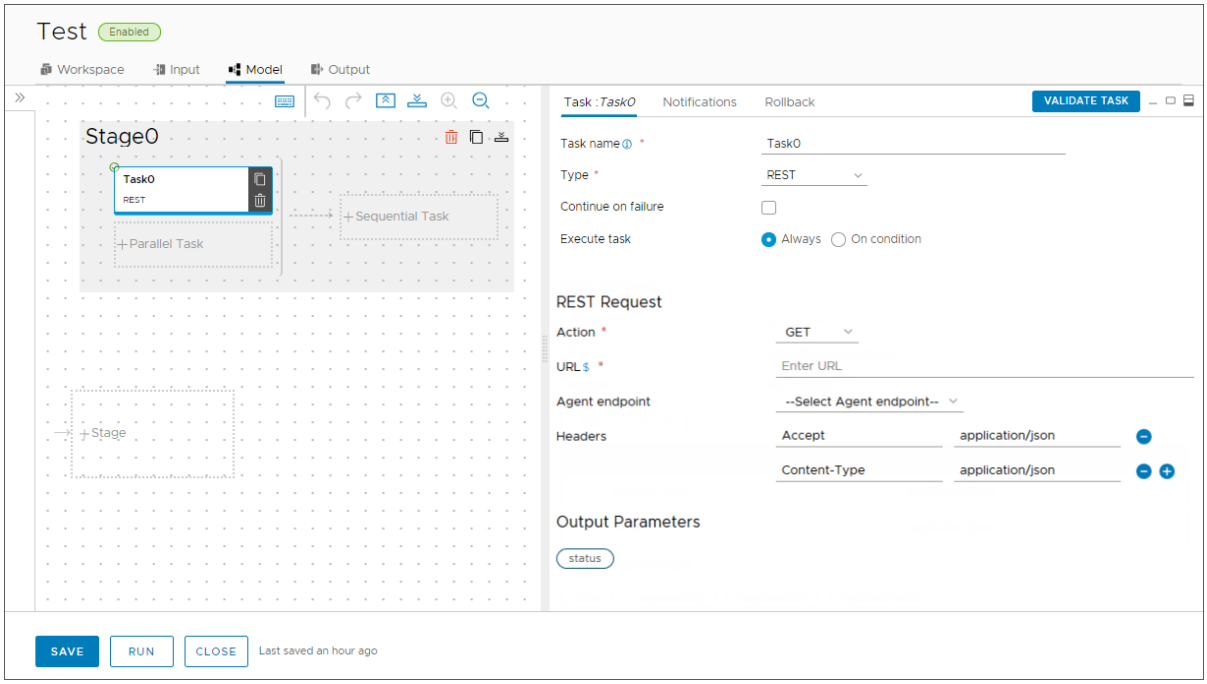
   c Dans la zone Demande REST, sélectionnez **GET**.

   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 REST API. Par exemple, `https://www.google.com`.

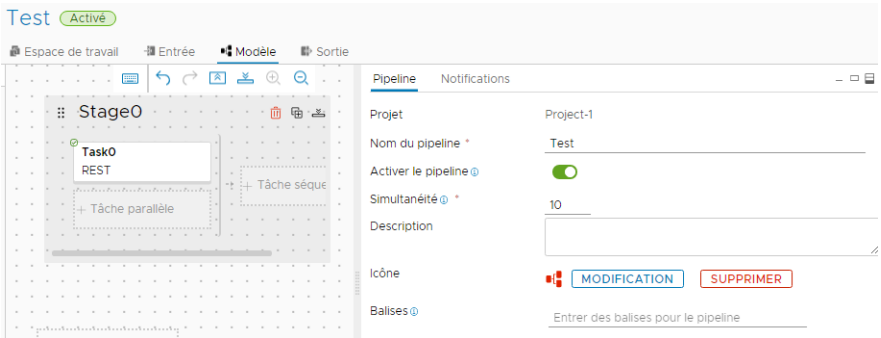
Pour qu'une tâche REST importe des données d'une autre application, vous pouvez inclure la variable de charge utile. Par exemple, pour une action d'importation, vous pouvez entrer `${Stage0.export.responseBody}`. Si la taille des données de réponse dépasse 5 Mo, la tâche REST peut échouer.

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



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

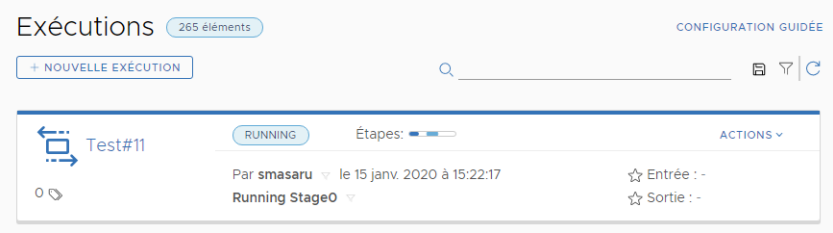
5 Dans l'onglet Pipeline, cliquez sur **Activer le pipeline**.



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

7 Cliquez sur **Exécuter**.

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



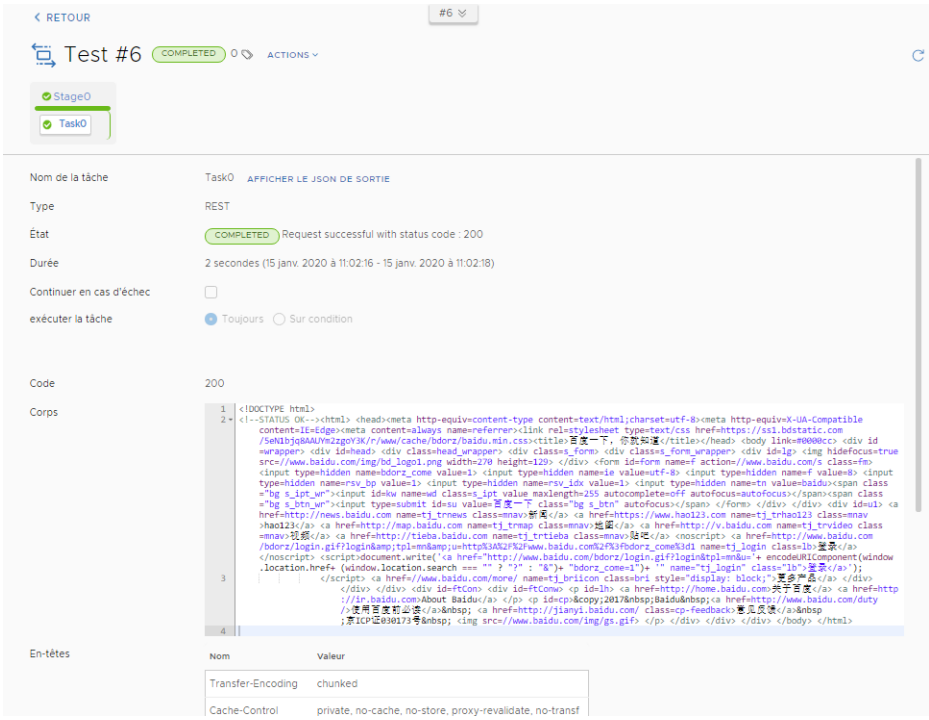
9 Pour vérifier que la tâche REST renvoie les informations attendues, examinez l'exécution du pipeline et les résultats de la tâche.

a 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.

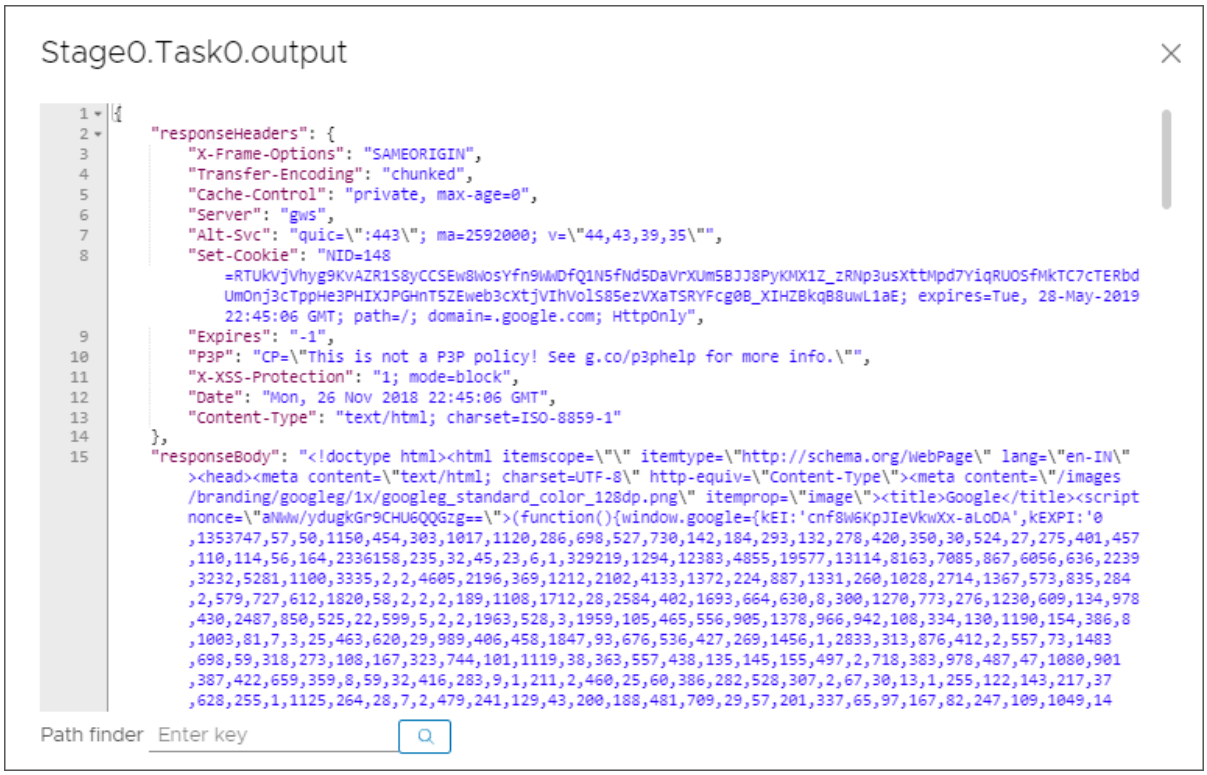
b Cliquez sur la tâche REST dans le pipeline.

c Dans l'exécution du pipeline, cliquez sur la tâche, examinez-en les détails et vérifiez que le tâche 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**.



Résultats

Félicitations ! Vous avez configuré une tâche REST qui a appelé une REST API et échangé des informations entre 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 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.

Comment utiliser le pipeline en tant que code dans Code Stream

En tant qu'administrateur ou développeur DevOps, vous pouvez créer un pipeline dans Code Stream à l'aide du code YAML, au lieu d'utiliser l'interface utilisateur. Lorsque vous créez des pipelines en tant que code, vous pouvez utiliser n'importe quel éditeur et insérer des commentaires dans le code de pipeline.

Dans votre code de pipeline, vous pouvez vous reporter à des configurations externes telles que des variables d'environnement et des informations d'identification de sécurité. Lorsque vous mettez à jour les variables que vous utilisez dans votre code de pipeline, vous pouvez les mettre à jour sans devoir mettre à jour le code de pipeline.

Vous pouvez utiliser le code YAML du pipeline comme modèle pour cloner et créer d'autres pipelines, et partager les modèles avec d'autres utilisateurs.

Vous pouvez stocker vos modèles de code de pipeline dans un référentiel de contrôle de code source, qui en gère les versions et assure le suivi des mises à jour. En utilisant un système de contrôle de source, vous pouvez facilement sauvegarder votre code de pipeline et le restaurer si nécessaire.

Conditions préalables

- Vérifiez que vous disposez d'un éditeur de code.
- Si vous prévoyez de stocker votre code de pipeline dans un référentiel de contrôle de source, vérifiez que vous pouvez accéder à une instance opérationnelle.

Procédure

1. Dans votre éditeur de code, créez un fichier.
2. Copiez et collez l'exemple de code de pipeline et mettez-le à jour pour refléter les besoins spécifiques de votre pipeline.
3. Pour inclure un point de terminaison dans votre code de pipeline, copiez et collez l'exemple de code de point de terminaison, puis mettez-le à jour pour refléter votre point de terminaison.

   Lors de l'utilisation d'un point de terminaison d'API Kubernetes dans l'espace de travail de pipeline, Code Stream crée les ressources Kubernetes nécessaires, telles que ConfigMap, Secret et Pod, pour exécuter la tâche d'intégration continue (CI) ou la tâche personnalisée. Code Stream communique avec le conteneur à l'aide du port de nœud.

   L'espace de travail de pipeline Code Stream prend en charge Docker et Kubernetes pour les tâches d'intégration continue et les tâches personnalisées.

   Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.
4. Enregistrez le code.
5. Pour stocker et gérer les versions de votre code de pipeline, archivez le code dans votre référentiel de contrôle de code source.
6. Lorsque vous créez un pipeline d'intégration et de livraison continues, vous devez importer le fichier YAML Kubernetes.

   Pour importer le fichier YAML Kubernetes, sélectionnez-le dans la zone Livraison continue du modèle de pipeline intelligent, puis cliquez sur **Traiter**. Vous pouvez également utiliser l'API.

Résultats

En utilisant les exemples de code, vous avez créé le code YAML qui représente votre pipeline et vos points de terminaison.

Exemple : Exemple de code YAML pour un pipeline et des points de terminaison.

Cet exemple de code YAML inclut des sections qui représentent l'espace de travail de la build native, des étapes, des tâches, des notifications, etc. de Code Stream.

Pour obtenir des exemples de code pour les plug-ins pris en charge, consultez Chapitre 6 Connexion de Code Stream à des points de terminaison

```
---
kind: PIPELINE
name: myPipelineName
tags:
  - tag1
  - tag2


# Ready for execution
enabled: false

#Max number of concurrent executions
concurrency: 10

#Input Properties
input:
  input1: '30'
  input2: 'Hello'

#Output Properties
output:
  BuildNo: '${Dev.task1.buildNo}'
  Image: '${Dev.task1.image}'

#Workspace Definition
ciWorkspace:
  image: docker:maven-latest
  path: /var/tmp
  endpoint: my-k8s
  cache:
    - ~/.m2

# Starred Properties
starred:
  input: input1
  output: output1

# Stages in order of execution
stageOrder:
  - Dev
  - QA
  - Prod

# Task Definition Section
stages:
```

```
  Dev:
    taskOrder:
      - Task1, Task6
      - Task2 Long, Task Long Long
      - Task5
    tasks:
      Task1:
        type: jenkins
        ignoreFailure: false
        preCondition: ''
        endpoints:
          jenkinsServer: myJenkins
        input:
          job: Add Two Numbers
          parameters:
            number1: 10
            number2: 20
      Task2:
        type: blah
        # repeats like Task1 above
  QA:
    taskOrder:
      - TaskA
      - TaskB
    tasks:
      TaskA:
        type: ssh
        ignoreFailure: false
        preCondition: ''
        input:
          host: x.y.z.w
          username: abcd
          password: ${var.mypassword}
          script: >
            echo "Hello, remote server"
      TaskB:
        type: blah
        # repeats like TaskA above

# Notificatons Section
notifications:
  email:
    - stage: Dev #optional ; if not found - use pipeline scope
      task: Task1 #optional; if not found use stage scope
      event: SUCCESS
      endpoint: default
      to:
         - user@yourcompany.com
         - abc@yourcompany.com
      subject: 'Pipeline ${name} has completed successfully'
      body: 'Pipeline ${name} has completed successfully'

  jira:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskA #optional; if not found use stage scope
```

```
      event: FAILURE
      endpoint: myJiraServer
      issuetype: Bug
      project: Test
      assignee: abc
      summary: 'Pipeline ${name} has failed'
      description: |-
        Pipeline ${name} has failed
        Reason - ${resultsText}
  webhook:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskB #optional; if not found use stage scope
      event: FAILURE
      agent: my-remote-agent
      url: 'http://www.abc.com'
      headers: #requestHeaders: '{"build_no":"123","header2":"456"}'
          Content-Type: application/json
          Accept: application/json
      payload: |-
        Pipeline ${name} has failed
        Reason - ${resultsJson}
---
```

Ce code YAML représente un exemple de point de terminaison Jenkins.

```
---
name: My-Jenkins
tags:
- My-Jenkins
- Jenkins
kind: ENDPOINT
properties:
  offline: true
  pollInterval: 15.0
  retryWaitSeconds: 60.0
  retryCount: 5.0
  url: http://urlname.yourcompany.com:8080
description: Jenkins test server
type: your.jenkins:JenkinsServer
isLocked: false
---
```

Ce code YAML représente un exemple de point de terminaison Kubernetes.

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

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

Étape suivante

Exécutez votre pipeline et apportez les modifications nécessaires. Reportez-vous à la section Comment exécuter un pipeline et afficher les résultats.

Connexion de Code Stream à des points de terminaison

6

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

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

Pour intégrer Code Stream à Jira, vous n'avez pas besoin d'un plug-in externe, car Code Stream inclut la capacité de création de ticket Jira 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 Code Stream

Un point de terminaison est une instance d'une application DevOps qui se connecte à Code Stream et met 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 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 Code Stream.

Pour connecter Code Stream à un point de terminaison, procédez comme suit.

1 Ajouter une tâche dans votre pipeline

2 Configurez la tâche pour qu'elle communique avec le point de terminaison.

3 Vérifiez que Code Stream peut se connecter au point de terminaison en cliquant sur **Valider**.

4 Ensuite, lorsque vous exécutez le pipeline, la tâche se connecte au point de terminaison afin qu'elle puisse exécuter la tâche.

Pour plus d'informations sur les types de tâches qui utilisent ces points de terminaison, reportez-vous à la section Types de tâches disponibles dans Code Stream.

Tableau 6-1. Points de terminaison pris en charge par 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 de Docker Hub, vous devez vous assurer que l'image intègre `cURL` ou `wget` avant d'exécuter le pipeline. Lorsque le pipeline s'exécute, Code Stream télécharge un fichier binaire qui utilise `cURL` ou `wget` 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.*	

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

Point de terminaison	Opérations possibles	Versions prises en charge	Configuration requise
Kubernetes	Automatise les étapes de déploiement, de mise à l'échelle et de gestion des applications en conteneur.	Toutes les versions prises en charge pour Cloud Assembly 8.4 et versions ultérieures 1.18 pour Cloud Assembly 8,3 et versions antérieures	Lors de l'utilisation d'un point de terminaison d'API Kubernetes dans l'espace de travail de pipeline, Code Stream crée les ressources Kubernetes nécessaires, telles que ConfigMap, Secret et Pod, pour exécuter la tâche d'intégration continue (CI) ou la tâche personnalisée. Code Stream communique avec le conteneur à l'aide du port de nœud. Pour plus d'informations sur la configuration de l'espace de travail, consultez Configuration de l'espace de travail de pipeline.
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	
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 Code Stream à Jenkins

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 Code Stream. Ensuite, vous créez un pipeline et y ajoutez une tâche Jenkins.

Lorsque vous utilisez la tâche Jenkins et un point de terminaison Jenkins dans Code Stream, vous pouvez créer un pipeline qui prend en charge les tâches multi-branches dans Jenkins. La tâche multi-branche inclut des tâches individuelles dans chaque branche d'un référentiel Git. Lorsque vous créez des pipelines dans Code Stream qui prennent en charge les tâches multi-branches :

- La tâche Jenkins peut exécuter des tâches Jenkins qui résident dans plusieurs dossiers sur le serveur Jenkins.
- Vous pouvez remplacer le chemin du dossier dans la configuration des tâches Jenkins afin qu'elle utilise un chemin de dossier différent, ce qui remplace le chemin par défaut défini dans le point de terminaison Jenkins dans Code Stream.
- Les pipelines à multi-branches dans Code Stream détectent les fichiers de tâche Jenkins de type `.groovy` dans un référentiel Git ou un référentiel GitHub, et commencent à créer des tâches pour chaque branche qu'ils détectent dans le référentiel.
- Vous pouvez remplacer le chemin d'accès par défaut défini dans le point de terminaison Jenkins par un chemin fourni dans la configuration de la tâche Jenkins, et exécuter une tâche et un pipeline associés à n'importe quelle branche d'une tâche Jenkins principale.

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 Code Stream. Si vous n'êtes pas membre, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans 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 sélectionnez **Jenkins** comme 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 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	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 : ■ `folder_1` peut inclure `job_1` ■ `folder_1` peut inclure `folder_2`, qui peut inclure `job_2` Lorsque vous créez un point de terminaison pour `folder_1`, le chemin du dossier est `job/folder_1`, et le point de terminaison répertorie uniquement `job_1`. Pour obtenir la liste des travaux dans le dossier enfant nommé `folder_2`, vous devez créer un autre point de terminaison qui utilise le chemin d'accès au dossier, comme `/job/folder_1/job/folder_2/`.
Chemin d'accès au dossier pour les tâches Jenkins multi-branches	Pour prendre en charge les tâches Jenkins multi-branches, dans la tâche Jenkins, vous entrez le chemin d'accès complet qui inclut l'URL du serveur Jenkins et le chemin d'accès complet à la tâche. Lorsque vous incluez un chemin de dossier dans la tâche Jenkins, ce chemin remplace le chemin qui s'affiche dans le point de terminaison Jenkins. Avec le chemin d'accès au dossier personnalisé dans la tâche Jenkins, Code Stream affiche uniquement les tâches présentes dans ce dossier. ■ Par exemple : `https://server.yourcompany.com/job/project` ■ Si le pipeline doit également déclencher la tâche Jenkins principale, utilisez : `https://server.yourcompany.com/job/project/job/main`
URL	URL hôte du serveur Jenkins. Entrez l'URL sous la forme `protocol://host:port`. Par exemple : `http://192.10.121.13:8080`
Intervalle d'interrogation	Durée de l'intervalle d'interrogation du serveur Jenkins par Code Stream, à la recherche de mises à jour.

Tableau 6-2. Informations restantes pour le point de terminaison Jenkins (suite)

Entrée du point de terminaison	Description
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 Code Stream. S'il ne se connecte pas, corrigez les erreurs éventuelles, puis cliquez sur **Enregistrer**.

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.

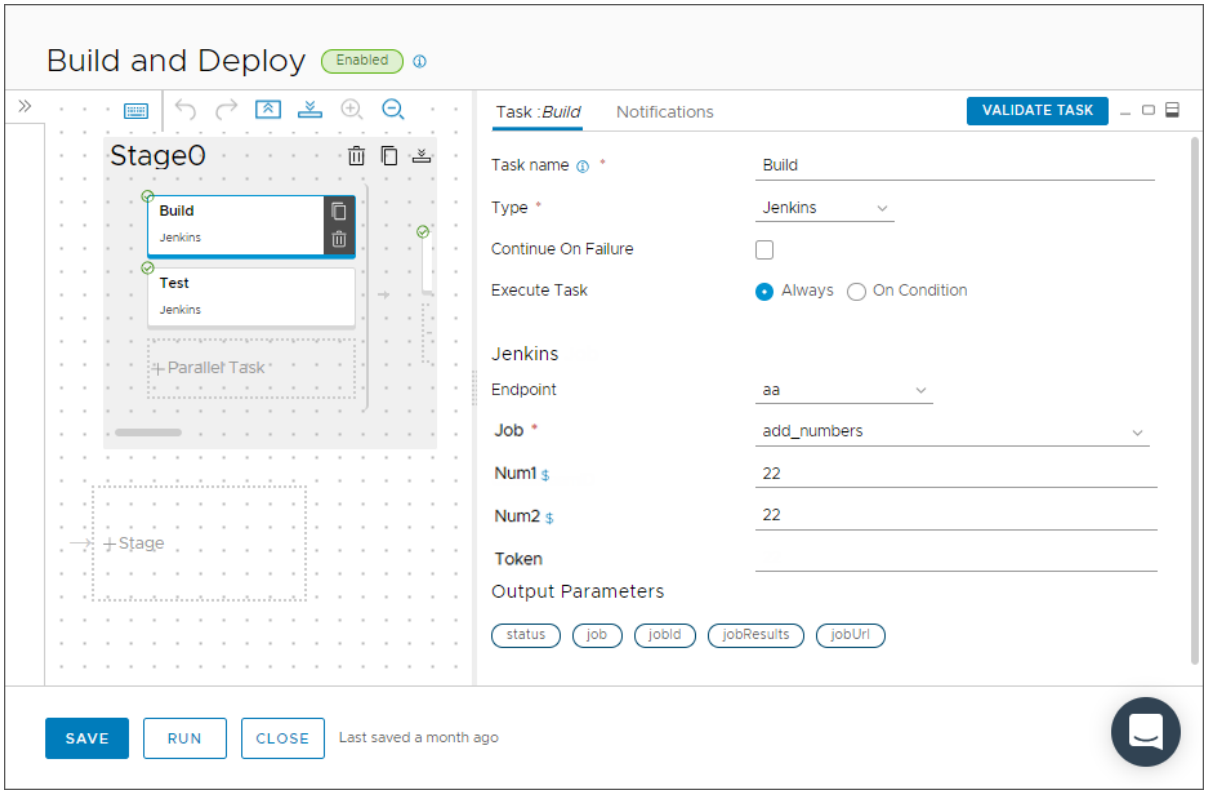
d Sélectionnez le type de tâche **Jenkins**.

e Sélectionnez le point de terminaison Jenkins que vous avez créé.

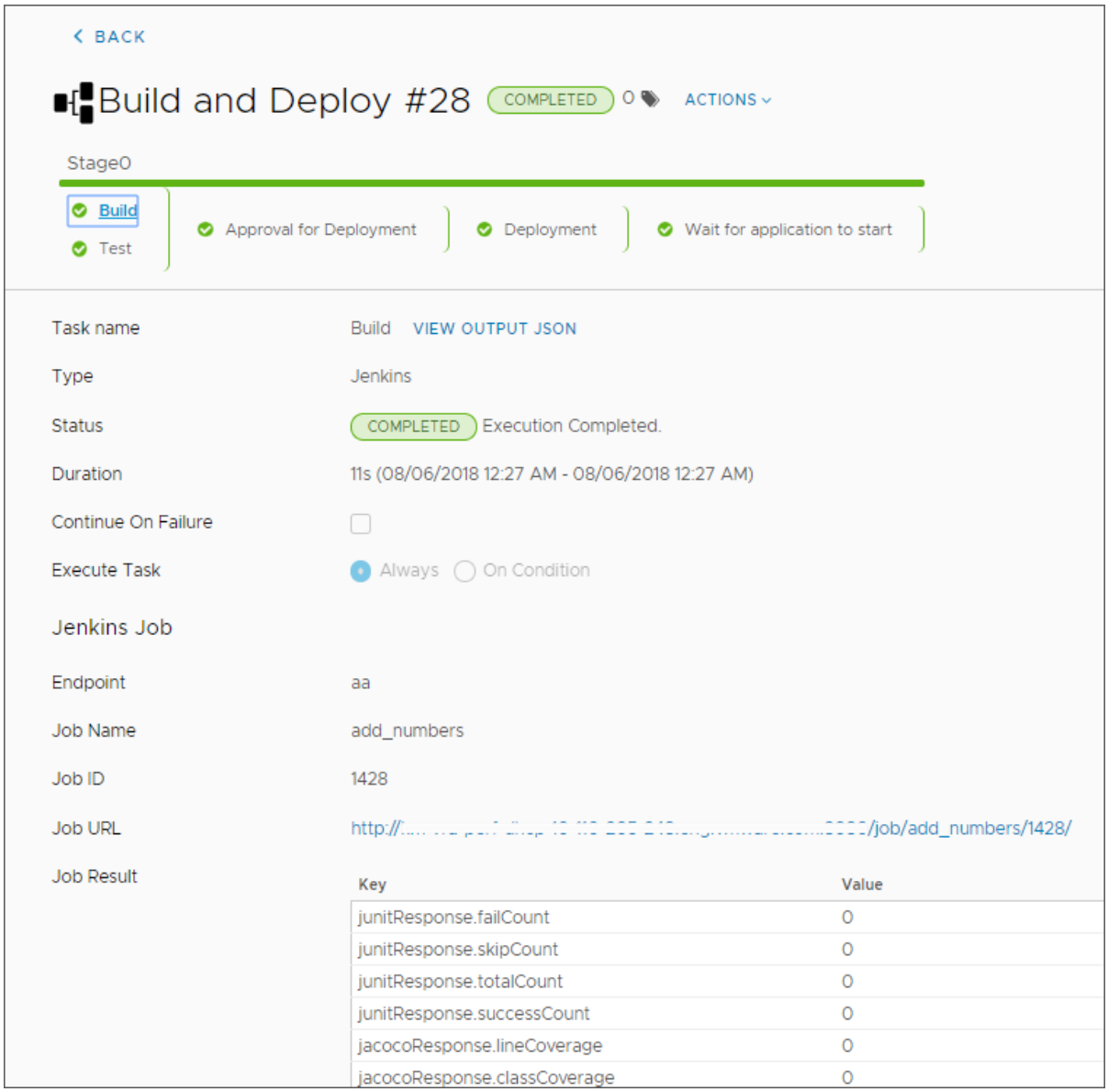
f Dans le menu déroulant, sélectionnez, sur le serveur Jenkins, un travail que votre pipeline exécutera.

g Saisissez les paramètres correspondant au travail.

h Entrez le jeton d'authentification du travail Jenkins.

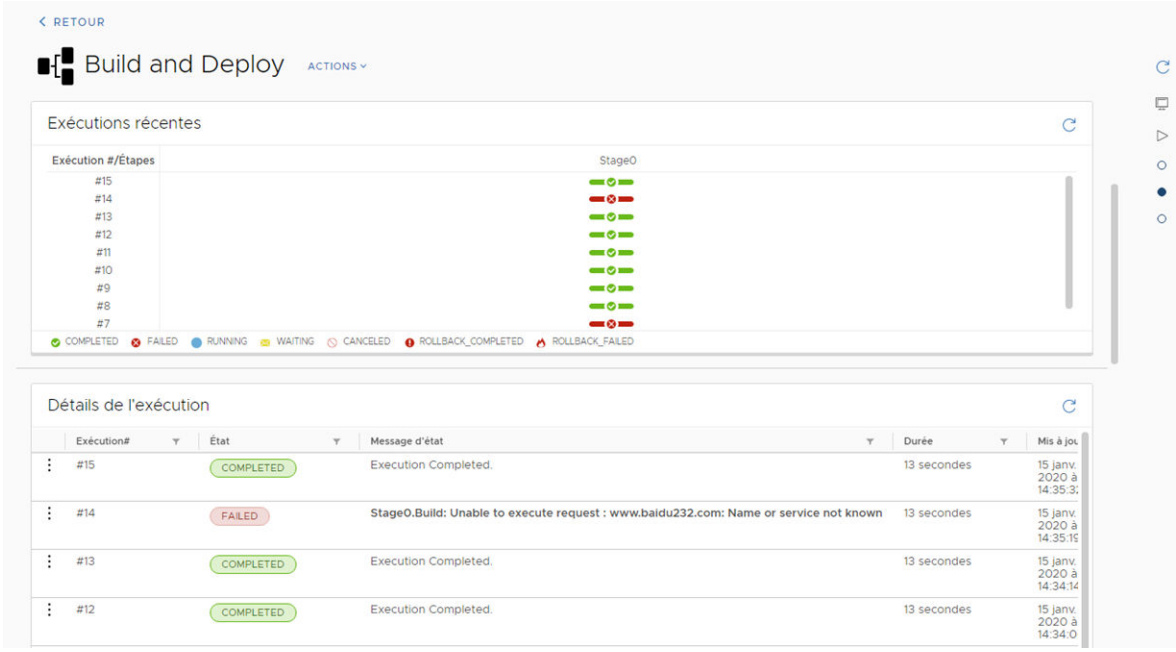

Build and Deploy
Enabled
Stage0
Build
Jenkins
Test
Jenkins
Parallel Task
Stage
Task : Build
Notifications
VALIDATE TASK
Task name *
Build
Type *
Jenkins
Continue On Failure
Execute Task
Always
On Condition
Jenkins
Endpoint
aa
Job *
add_numbers
Num1
22
Num2
22
Token
Output Parameters
status
job
jobId
jobResults
jobUrl
SAVE
RUN
CLOSE
Last saved a month ago

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



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 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 Code Stream à des points de terminaison.

Comment intégrer Code Stream à Git

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. Code Stream se connecte au point de terminaison Git via un webhook.

Pour définir un point de terminaison Git dans 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 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 Code Stream. Si ce n'est pas le cas, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans 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 Code Stream
- Chapitre 2 Configuration de 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 :

Pour GitHub, entrez : **`https://api.github.com/vmware-example/repo-example`**

Pour BitBucket, entrez : **`https://api.bitbucket.org/{user}/{repo name}`** or **`http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`**

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 jeton privé qui correspond au nom d'utilisateur.

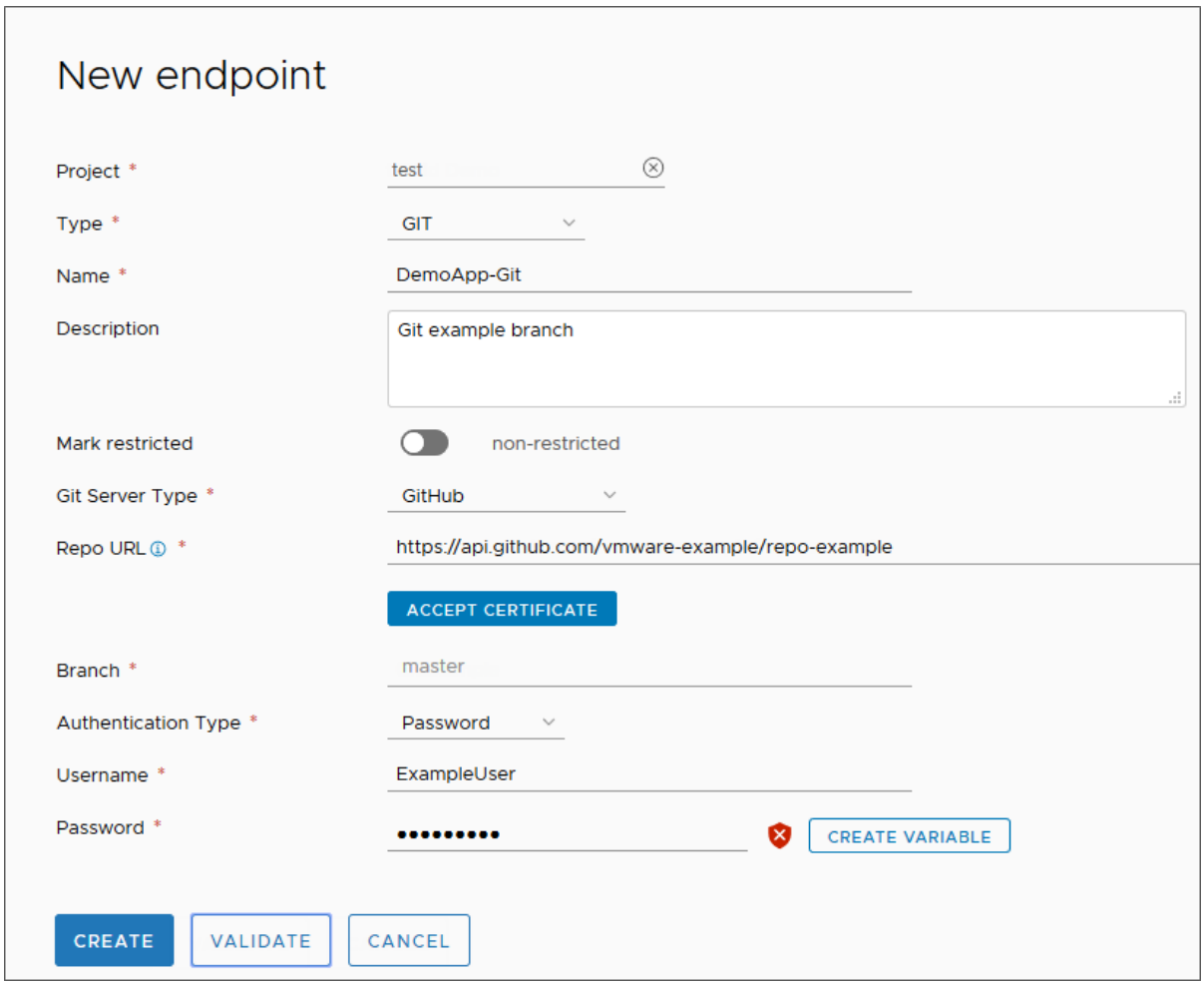
- Mot de passe. Pour créer un Webhook ultérieurement, vous devez entrer le jeton privé pour le mot de passe. Les Webhooks pour Git ne prennent pas en charge les points de terminaison créés à l'aide de l'authentification de base.

  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. Vous pouvez également créer une variable pour le jeton privé.

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

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



Étape suivante

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

Intégration de Code Stream à Gerrit

Code Stream vous permet de déclencher un pipeline lorsqu'un code fait l'objet d'une révision dans votre projet Gerrit. Le déclencheur de la définition Gerrit inclut le projet Gerrit et les pipelines à exécuter pour différents types d'événement.

Le déclencheur pour Gerrit utilise un écouteur Gerrit sur le serveur que vous allez surveiller. Pour définir un point de terminaison Gerrit dans 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.

Si vous utilisez un serveur Gerrit comme point de terminaison Code Stream dans une instance de vRealize Automation sur laquelle FIPS est activé, vous devez vérifier que votre fichier de configuration Gerrit inclut les clés d'authentification de message correctes. Si le fichier de configuration du serveur Gerrit n'inclut pas les clés d'authentification de message correctes, le serveur ne peut pas démarrer correctement et affiche le message suivant : `PrivateKey/PassPhrase is incorrect`

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 Code Stream. Si vous n'êtes pas membre, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans 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 pour le type de point de terminaison sélectionnez **Gerrit**. 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.

   Pour utiliser le port par défaut vous pouvez fournir un numéro de port avec l'URL ou laisser la valeur vide.

   e Indiquez un nom d'utilisateur et un mot de passe pour le serveur Gerrit.

   Si le mot de passe doit être chiffré, cliquez sur **Créer une variable** et sélectionnez le type :

   - Secret. Le mot de passe est résolu lorsqu'un utilisateur ayant un rôle exécute le pipeline.
   - Restreint. Le mot de passe est résolu lorsqu'un utilisateur ayant le rôle d'administrateur exécute le pipeline.

   Pour la valeur , entrez le mot de passe qui doit être sécurisé, 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 qui réside dans le répertoire `.ssh`.

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

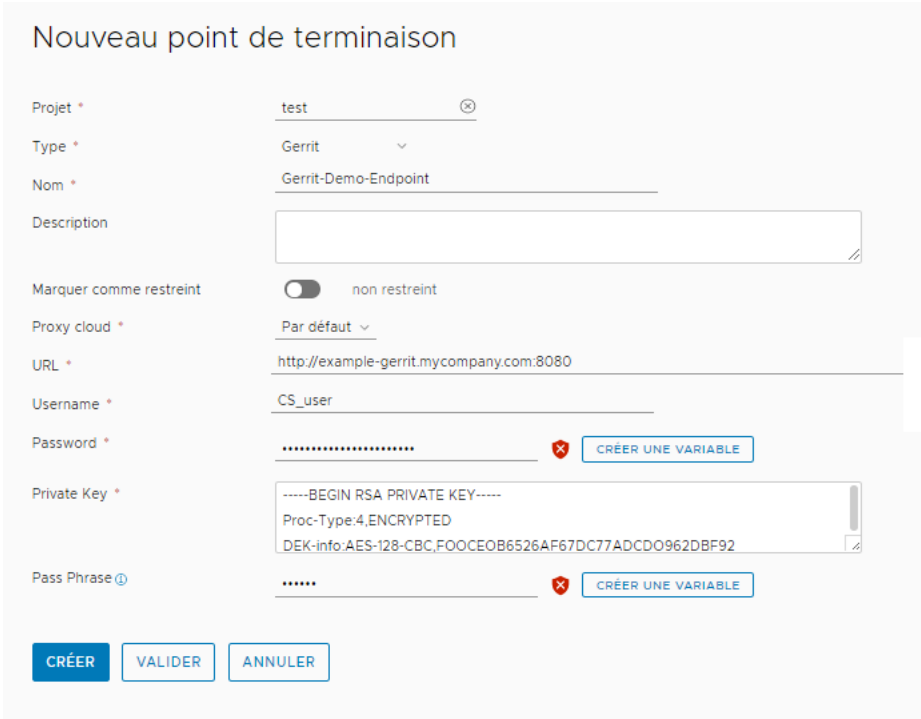
Pour chiffrer la phrase secrète, cliquez sur **Créer la variable** et sélectionnez le type :

- Secret. Le mot de passe est résolu lorsqu'un utilisateur ayant un rôle exécute le pipeline.
- Restreint. Le mot de passe est résolu lorsqu'un utilisateur ayant le rôle d'administrateur exécute le pipeline.

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

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

S'il ne se connecte pas, corrigez les erreurs éventuelles, puis cliquez de nouveau sur **Valider**.



3 Cliquez sur **Créer**.

4 Vérifiez que FIPS est activé dans l'environnement vRealize Automation ou que la tâche Jenkins crée un environnement dans lequel FIPS est activé à l'aide de l'URL Jenkins.

   a Pour exécuter la commande à partir de la ligne de commande, connectez-vous à votre dispositif vRealize Automation 8.x via SSH et connectez-vous en tant qu'utilisateur racine. Par exemple, connectez-vous à l'URL de votre nom de domaine complet, telle que `https://cava-1-234-567.yourcompanyFQDN.com` sur le port 22, 5480 ou 443.

   b Pour vérifier la présence de FIPS sur vRealize Automation, exécutez la commande `vracli security fips`.

   c Vérifiez que la commande renvoie `FIPS mode: strict`.

5 Si le serveur Gerrit est un point de terminaison dans une instance de vRealize Automation sur laquelle FIPS est activé, assurez-vous que votre fichier de configuration Gerrit inclut les clés d'authentification de message correctes (MAC).

   a Ouvrez Gerrit et créez une paire de clés SSH.

   b Localisez le fichier de configuration du serveur Gerrit dans `'$site_path'/etc/Gerrit.config`.

   c Vérifiez que le fichier de configuration du serveur Gerrit inclut une ou plusieurs clés de code d'authentification de message (MAC), à l'exception de `hmac-MD5`.

   Note En mode FIPS, `hmac-MD5` n'est pas un algorithme MAC pris en charge. Pour vous assurer que le serveur Gerrit démarre correctement, le fichier de configuration du serveur Gerrit doit exclure cet algorithme. Si le serveur Gerrit ne démarre pas correctement, il affiche le message suivant : `PrivateKey/PassPhrase is incorrect`

   Les noms de clé MAC (Message Authentication Code) pris en charge qui commencent par un signe plus (+) sont activés. Les noms de clé MAC qui commencent par un trait d'union (-) sont supprimés de la liste des code MAC par défaut. Par défaut, ces MAC pris en charge sont disponibles dans Code Stream pour le serveur Gerrit :

   - `hmac-md5-96`
   - `hmac-sha1`
   - `hmac-sha1-96`
   - `hmac-sha2-256`
   - `hmac-sha2-512`

Étape suivante

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

Intégration de Code Stream à vRealize Orchestrator

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.

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 Code Stream. Si ce n'est pas le cas, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans Code Stream.
- Dans 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 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 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 Code Stream et l'ajout d'une tâche `vRO` à votre pipeline, le workflow balisé s'affiche dans la liste des workflows.

2 Dans 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://vro-appliance.yourdomain.local:8281`**

N'utilisez pas ce format : https://vro-appliance.yourdomain.local:8281/vco/api

L'URL d'une instance de vRealize Orchestrator intégrée au dispositif vRealize Automation est le nom de domaine complet du dispositif sans port. Par exemple : **`https://vra-appliance.yourdomain.local/vco`**

Pour les dispositifs vRealize Orchestrator externes à partir de vRealize Automation 8.x, le nom de domaine complet du dispositif est **`https://dispositif-vro.votredomaine.local`**

Pour les dispositifs vRealize Orchestrator externes inclus avec vRealize Automation 7.x, le nom de domaine complet du dispositif est **`https://dispositif-vro.votredomaine.local:8281/vco`**

Si un problème se produit lorsque vous ajoutez le point de terminaison, vous devrez peut-être importer une configuration YAML avec une empreinte digitale de certificat SHA-256 avec les deux-points supprimés. Par exemple, **`B0:01:A2:72...`** devient **`B001A272...`**. L'exemple de code YAML ressemble à ce qui suit :

```
```
---
project: Demo
kind: ENDPOINT
name: external-vro
description: ''
type: vro
properties:
  url: https://yourVROhost.yourdomain.local
  username: yourusername
  password: yourpassword
  fingerprint: <your_fingerprint>
```
```

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.

Si vous utilisez un utilisateur non local pour l'authentification, vous devez omettre la partie domaine du nom d'utilisateur. Par exemple, pour vous authentifier avec **`svc_vro@votredomaine.local`** vous devez entrer **`svc_vro`** dans la zone de texte **Nom d'utilisateur**.

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

a Ajoutez une tâche `vRO` à votre étape de pipeline.

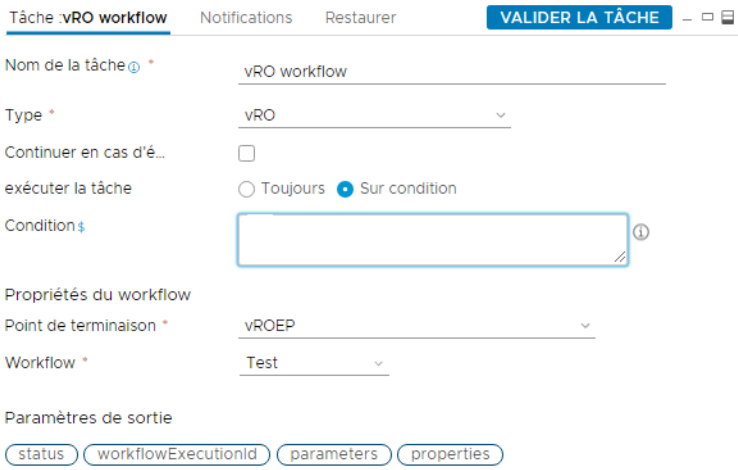
b Entrez un nom pertinent.

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...
Sur condition	Exécute la tâche de pipeline uniquement si la condition définie est évaluée comme étant true. Si la condition est false, la tâche est ignorée. La tâche `vRO` vous permet d'inclure une expression booléenne qui utilise les opérandes et les opérateurs suivants. ■ Variables de pipeline comme `${pipeline.variableName}`. Utilisez uniquement des accolades lors de la saisie de variables. ■ Variables de sortie de tâche comme `${Stage1.task1.machines[0].value.hostIp[0]}`. ■ Variables de liaison de pipeline par défaut comme `${releasePipelineName}`. ■ Valeurs booléennes insensibles à la casse comme `true`, `false`, `'true'`, `'false'`. ■ Valeurs entières ou décimales sans guillemets. ■ Valeurs de chaîne utilisées avec des guillemets simples ou doubles, comme `"test"` ou `'test'`. ■ Types de chaînes et de valeurs numériques, comme `== Equals` et `!= Not Equals`. ■ Opérateurs relationnels comme >, >=, < et <=. ■ Logique booléenne comme `&&` et `||`. ■ Opérateurs arithmétiques comme +, –, * et /. ■ Expressions imbriquées à l'aide de parenthèses. ■ Les chaînes qui incluent la valeur littérale `ABCD` sont évaluées comme étant false et la tâche est ignorée. ■ Les opérateurs unaires ne sont pas pris en charge. Un exemple de condition peut être `${Stage1.task1.output} == "Passed" || ${pipeline.variableName} == 39`
Toujours	Si vous sélectionnez **Toujours**, le pipeline exécute la tâche sans conditions.

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 Code Stream et ajouté une tâche vRO dans votre pipeline 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 Code Stream 7

Vous pouvez demander à Code Stream de déclencher un pipeline lorsque certains événements se produisent.

Par exemple :

- le déclencheur Docker peut exécuter un pipeline lorsque nouvel artefact est créé ou mis à jour.
- Le déclencheur pour Git peut déclencher un pipeline lorsque les développeurs mettent à jour le code.
- Le déclencheur pour Gerrit peut déclencher un pipeline lorsque les développeurs examinent le code.

Ce chapitre contient les rubriques suivantes :

Utilisation du déclencheur Docker dans Code Stream pour exécuter un pipeline de livraison continue

En tant qu'administrateur ou développeur de Code Stream, vous pouvez utiliser le déclencheur Docker dans 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 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 Code Stream.

Tableau 7-1. Utilisation du déclencheur Docker

Actions...	Plus d'informations sur cette action...
Créez un point de terminaison de registre Docker.	Pour que 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. Le point de terminaison du registre Docker inclut l'URL vers le référentiel Docker Hub.
Ajoutez des paramètres d'entrée au pipeline qui injecte automatiquement des paramètres Docker lors de son exécution.	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. 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.
Créez un Webhook Docker.	Lorsque vous créez le Webhook Docker dans Code Stream, cette action déclenche également la création d'un Webhook correspondant dans Docker Hub. Le Webhook Docker dans Code Stream se connecte à Docker Hub via l'URL que vous incluez dans le Webhook. Les Webhooks communiquent entre eux et déclenchent le pipeline lorsqu'un artefact est créé ou mis à jour dans Docker Hub. Si vous mettez à jour ou supprimez le Webhook Docker dans Code Stream, le Webhook de Docker Hub est également mis à jour ou supprimé.
Ajoutez et configurez une tâche Kubernetes dans votre pipeline.	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.
Incluez une définition YAML locale dans la tâche.	La définition YAML que vous appliquez à la tâche de déploiement inclut l'image de conteneur Docker. Si vous devez télécharger une image à partir d'un référentiel privé, le fichier YAML doit inclure une section avec le secret de configuration Docker. Reportez-vous à la partie CD de la section Planification d'une build native CICD dans Code Stream avant d'utiliser le modèle de pipeline intelligent.

Lorsqu'un artefact est créé ou mis à jour dans le référentiel Docker Hub, le Webhook de Docker Hub en informe le Webhook dans Code Stream, qui déclenche le pipeline. Les actions suivantes se produisent :

1 Docker Hub envoie une demande POST à l'URL du Webhook.

2 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 Code Stream déclenche son Webhook Docker, qui exécute un pipeline de livraison continue qui déploie l'artefact sur votre hôte Docker.

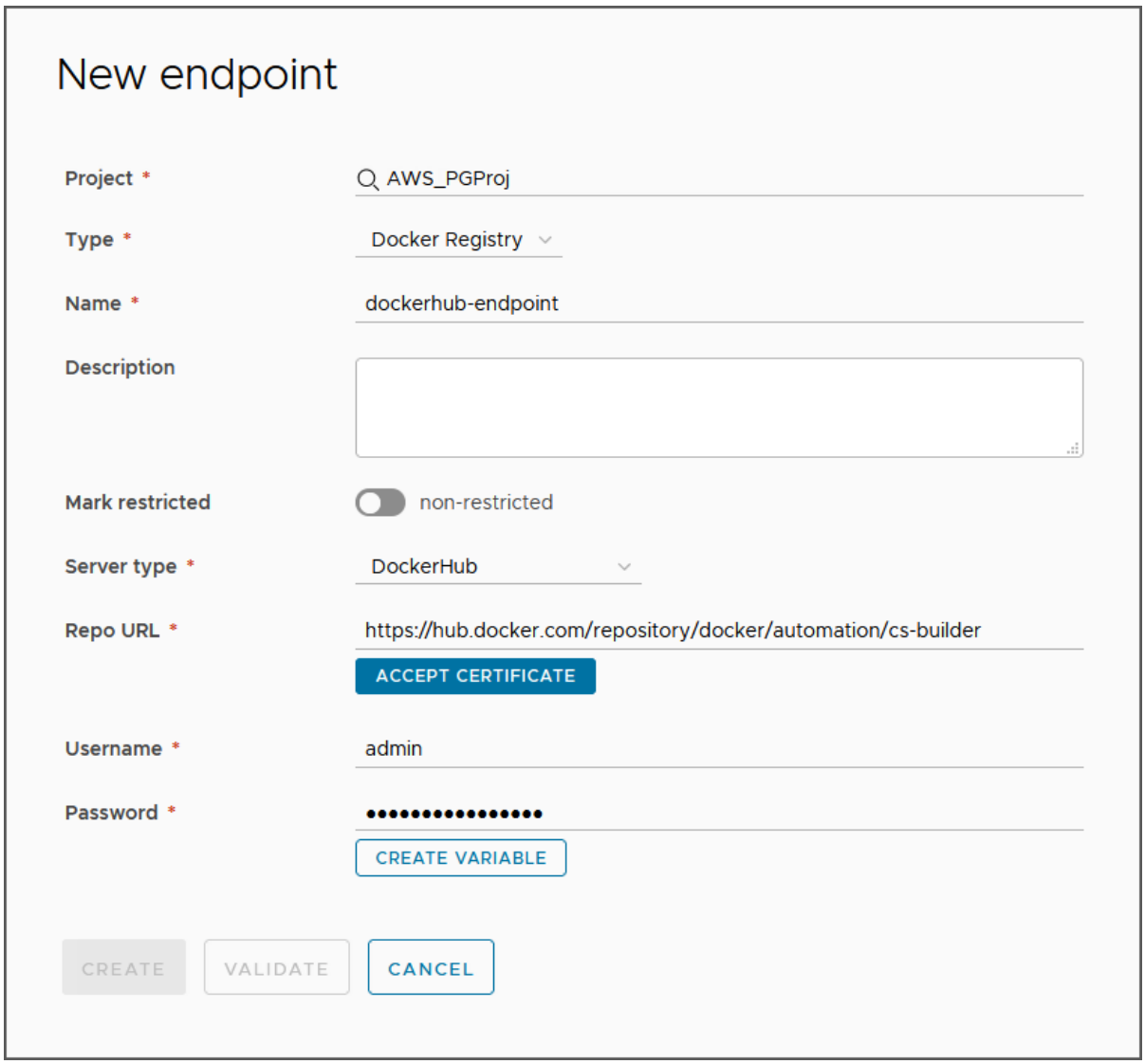
Dans cet exemple, vous créez un point de terminaison Docker et un Webhook Docker dans Code Stream qui déploie votre application sur votre cluster Kubernetes de développement. Les étapes incluent l'exemple de code de la charge utile que Docker publie sur l'URL du Webhook, le code d'API qu'il utilise et le code d'authentification avec le jeton sécurisé.

Conditions préalables

- Vérifiez qu'un pipeline de livraison continue (CD) existe dans votre instance de 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 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 Code Stream. Si ce n'est pas le cas, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans 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 Commencez à saisir le nom du projet existant.
   d Sélectionnez le type de **registre Docker**.
   e Entrez un nom pertinent.
   f Sélectionnez le type de serveur **DockerHub**.
   g Entrez l'URL du référentiel Docker Hub.
   h Entrez le nom et le mot de passe utilisés pour accéder au référentiel.


New endpoint
Project *
AWS_PGProj
Type *
Docker Registry
Name *
dockerhub-endpoint
Description
Mark restricted
non-restricted
Server type *
DockerHub
Repo URL *
https://hub.docker.com/repository/docker/automation/cs-builder
ACCEPT CERTIFICATE
Username *
admin
Password *
CREATE VARIABLE
CREATE
VALIDATE
CANCEL

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 CICD dans 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 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.

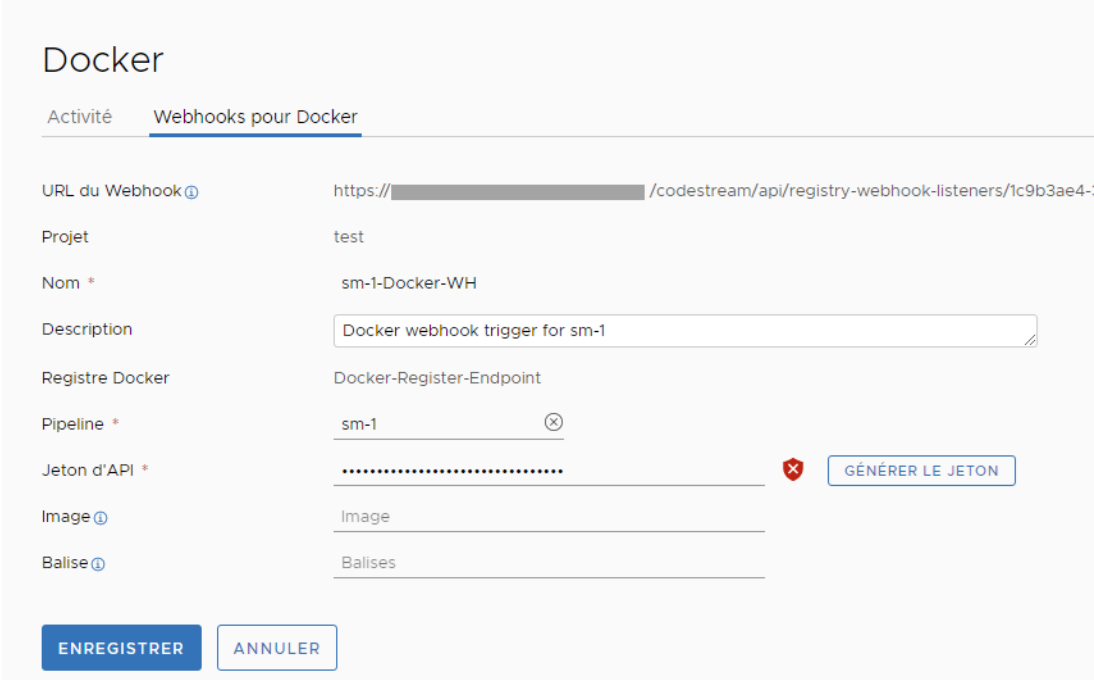
- Pour conserver le jeton sous forme de variable pour une utilisation ultérieure, cliquez sur **Créer une variable**, entrez un nom pour la variable et cliquez sur **Enregistrer**.
- Pour conserver le jeton sous forme de valeur de texte pour une utilisation ultérieure, cliquez sur **Copier** et collez 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.



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#
\u0009\u0009docker run -t -i -rm -e http_proxy http://192.168.1.2:3142/
debian bash\n#\nFROM\u0009\u0009ubuntu\n\n\nVOLUME\u0009\u0009[\/var/cache/apt-cacher-
ng\]\nRUN\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n\nEXPOSE
\u0009\u00093142\nCMD\u0009\u0009chmod 777 /var/cache/apt-cacher-ng ; /etc/init.d/apt-
cacher-ng start ; tail -f /var/log/apt-cacher-ng/*\n",
"full_description": "Docker Hub based automated build from a GitHub repo",
"is_official": false,
"is_private": true,
"is_trusted": true,
"name": "testhook",
"namespace": "svendowideit",
"owner": "svendowideit",
"repo_name": "svendowideit/testhook",
"repo_url": "https://registry.hub.docker.com/u/svendowideit/testhook/",
"star_count": 0,
"status": "Active"
}
}
```

L'API qui crée le Webhook dans Docker Hub utilise ce formulaire : `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",
```

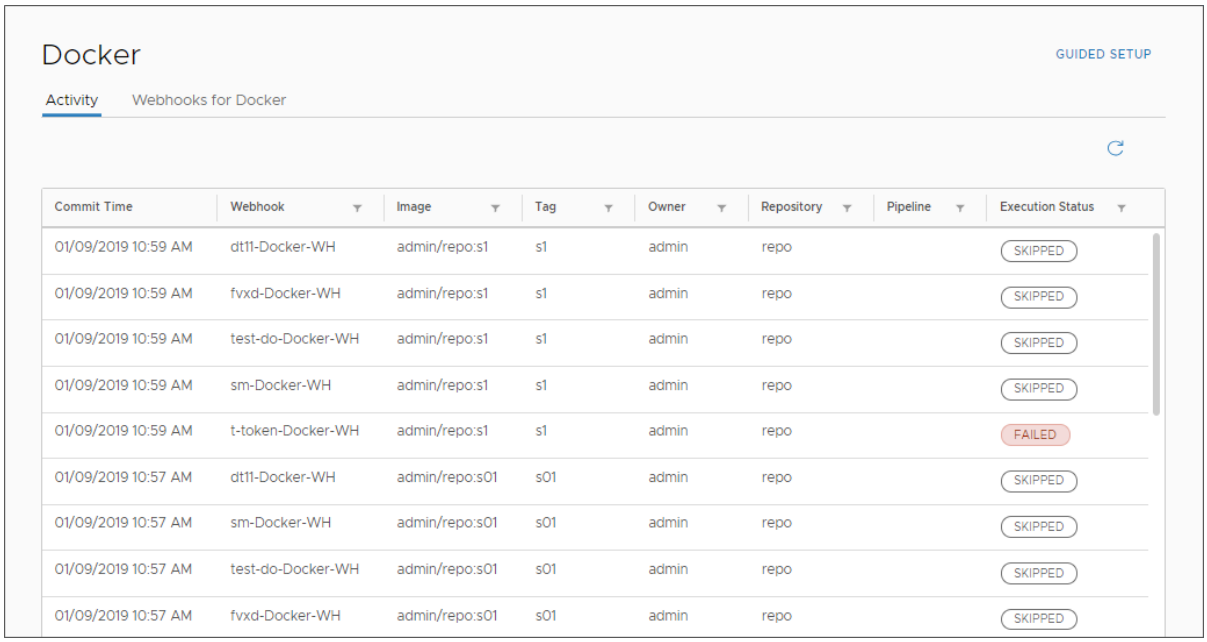
```
"webhooks": [
{
"name": "demo_webhook",
"hook_url": "http://www.google.com"
}
]
}
```

Pour recevoir des événements du serveur Docker Hub, le schéma d'authentification correspondant au Webhook Docker que vous créez dans 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`.

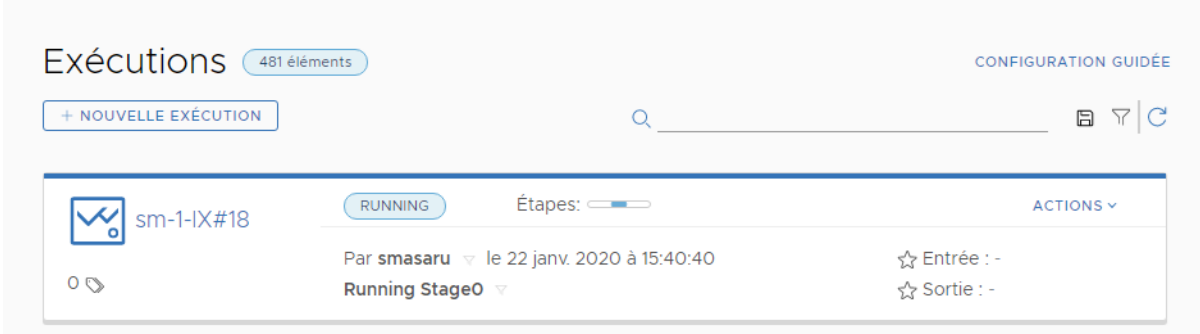
Code Stream peut vérifier toutes les demandes du serveur Docker Hub en utilisant le jeton sécurisé configuré. Par exemple : `hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""`

5 Créez un artefact Docker dans votre référentiel Docker Hub. Sinon, mettez à jour un artefact existant.

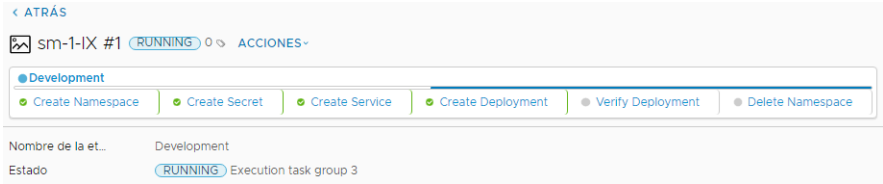
6 Pour vous assurer que le déclencheur est intervenu et afficher l'activité sur le Webhook Docker, cliquez sur **Déclencheurs > Docker > Activité**.



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 Code Stream pour exécuter un pipeline

En tant qu'administrateur ou développeur de Code Stream, vous pouvez intégrer Code Stream au cycle de vie Git à l'aide du déclencheur Git. Lorsque vous modifiez le code dans GitHub, GitLab ou Bitbucket Enterprise, l'événement communique avec Code Stream via un Webhook et déclenche un pipeline. Le Webhook fonctionne avec les versions d'entreprise sur site de GitLab, GitHub et Bitbucket lorsque Cloud Assembly et la version d'entreprise sont accessibles sur le même réseau.

Lorsque vous ajoutez le Webhook pour Git dans 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, cette action met également à jour ou supprime le Webhook dans GitHub, GitLab ou Bitbucket.

Votre définition du Webhook doit inclure un point de terminaison Git sur la branche du référentiel que vous souhaitez surveiller. Pour créer le Webhook, Code Stream utilise le point de terminaison Git. 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.

Note Pour créer un Webhook, le point de terminaison Git doit utiliser un jeton privé pour l'authentification ; il ne peut pas utiliser un mot de passe.

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 Code Stream. Si ce n'est pas le cas, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans 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 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.

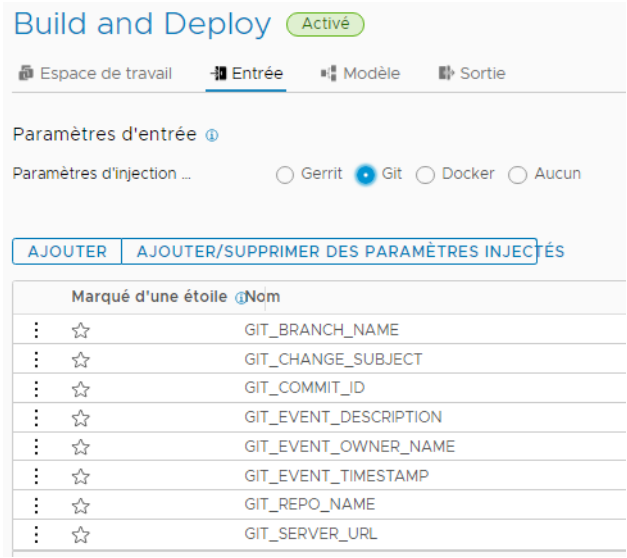
  a Connectez-vous à votre instance de GitLab en tant qu'administrateur.

  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.



  Pour plus d'informations sur les paramètres d'entrée, reportez-vous à la section Planification d'une build native CICD dans Code Stream avant d'ajouter des tâches manuellement.

Procédure

1 Dans 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 jeton privé dans le point de terminaison, le Webhook continuera à fonctionner.
- Si vous utilisez un référentiel BitBucket, l'URL du référentiel doit être dans l'un de ces formats : `https://api.bitbucket.org/{user}/{repo name}` ou `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`.

Note Si vous avez précédemment créé un Webhook à l'aide d'un point de terminaison Git qui utilise un mot de passe pour l'authentification de base, vous devez supprimer et redéfinir le Webhook avec un point de terminaison Git qui utilise un jeton privé pour l'authentification.

Reportez-vous à la section Comment intégrer Code Stream à Git.

d (Facultatif) Entrez la branche que vous souhaitez faire surveiller par le Webhook.

Si vous laissez la branche non spécifiée, le Webhook surveille la branche que vous avez configurée pour le point de terminaison Git.

e (Facultatif) Générez un jeton secret pour le Webhook.

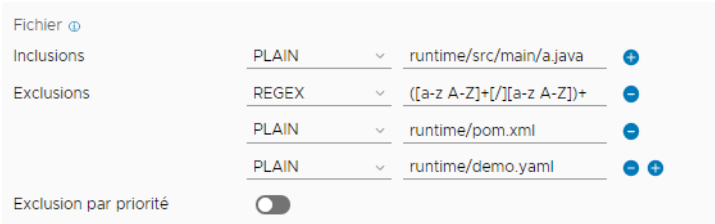
Si vous utilisez un jeton secret, Code Stream génère un jeton de chaîne aléatoire pour le Webhook. Ensuite, lorsque le Webhook reçoit les données d'événement Git, il les envoie avec le jeton secret. Code Stream utilise les informations pour déterminer si les appels proviennent de la source attendue, 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.

- Inclusions de fichiers. Si un des fichiers d'une validation correspond aux fichiers spécifiés dans les chemins d'inclusion ou regex, la validation déclenche les pipelines. Avec un regex spécifié, 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.
- Exclusions de fichiers. Lorsque tous les fichiers d'une validation correspondent aux fichiers spécifiés dans les chemins d'exclusion ou regex, les pipelines ne se déclenchent pas.
- Exclusions par priorité. Lorsque cette option est activée, l'exclusion par priorité garantit que les pipelines ne se déclenchent pas, 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 répondent à la fois aux inclusions et aux exclusions de fichiers, les pipelines ne se déclenchent pas.

Dans l'exemple suivant, les inclusions et les exclusions de fichiers sont des conditions de déclenchement.



- 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 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 inclut des paramètres d'entrée ajoutés personnalisés, la liste Paramètres d'entrée affiche les paramètres et les valeurs. Vous pouvez entrer des valeurs pour les paramètres d'entrée qui passent au pipeline avec l'événement déclencheur. Sinon, vous pouvez laisser les valeurs vides ou utiliser les valeurs par défaut, si elles sont définies.

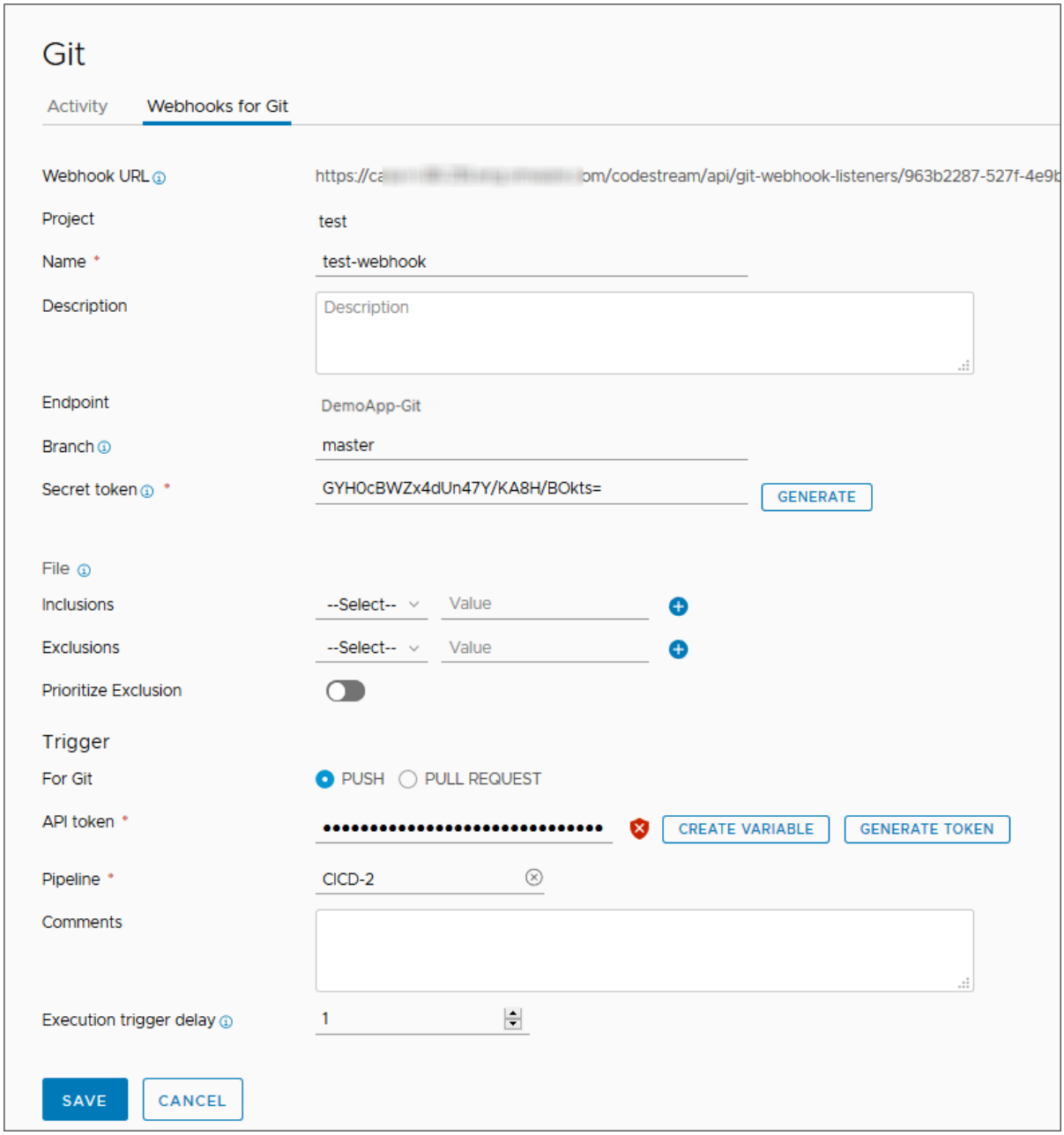
Pour plus d'informations sur les paramètres d'entrée injectés automatiquement pour les déclencheurs Git, reportez-vous aux conditions préalables.

j Cliquez sur **Créer**.

Le Webhook s'affiche sous la forme d'une nouvelle fiche.

3 Cliquez sur la fiche du Webhook.

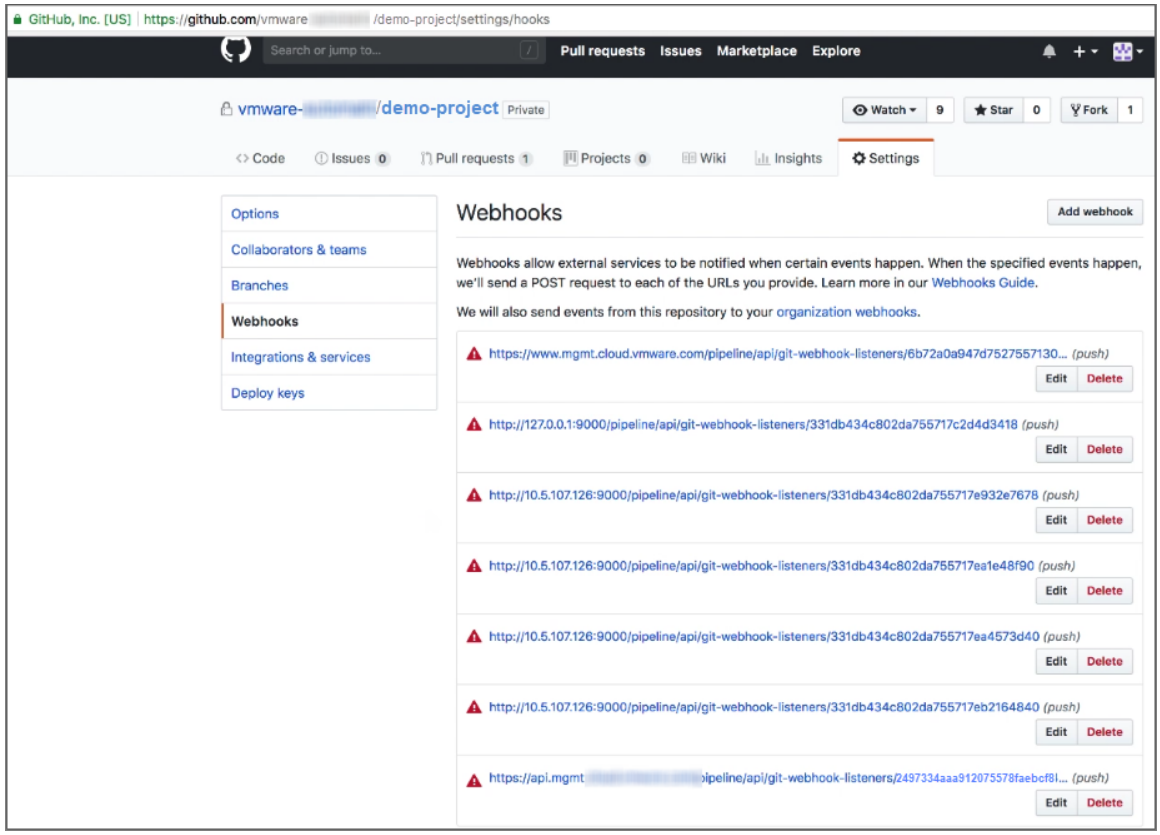
Lorsque le formulaire de données du Webhook s'affiche à nouveau, vous constatez qu'une URL Webhook a été ajoutée en haut du formulaire. Le Webhook Git se connecte au référentiel GitHub via l'URL Webhook.


Git
Activity
Webhooks for Git
Webhook URL
https://ca om/codestream/api/git-webhook-listeners/963b2287-527f-4e9b
Project
test
Name *
test-webhook
Description
Description
Endpoint
DemoApp-Git
Branch
master
Secret token *
GYH0cBWZx4dUn47Y/KA8H/BOkts=
GENERATE
File
Inclusions
--Select--
Value
Exclusions
--Select--
Value
Prioritize Exclusion
Trigger
For Git
PUSH
PULL REQUEST
API token *
CREATE VARIABLE
GENERATE TOKEN
Pipeline *
CICD-2
Comments
Execution trigger delay
1
SAVE
CANCEL

4 Dans une nouvelle fenêtre de navigateur, ouvrez le référentiel GitHub qui se connecte via le Webhook.

a Pour voir le Webhook que vous avez ajouté dans Code Stream, cliquez sur l'onglet **Paramètres** et sélectionnez **Webhooks**.

Au bas de la liste des Webhooks, la même URL Webhook est indiquée.



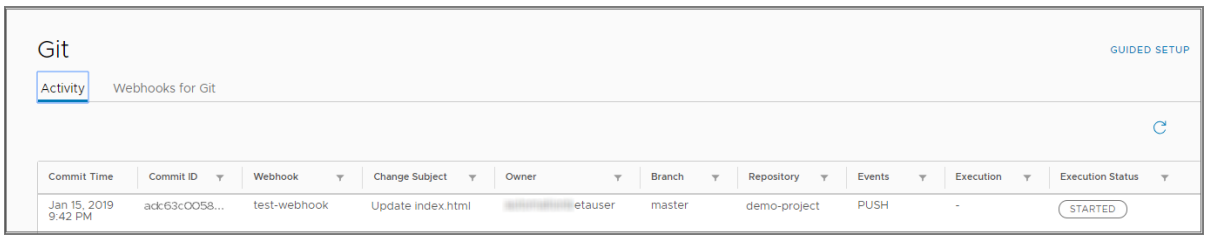
b Pour apporter une modification au code, cliquez sur l'onglet **Code** et sélectionnez un fichier sur la branche. Après avoir modifié le fichier, validez la modification.

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.

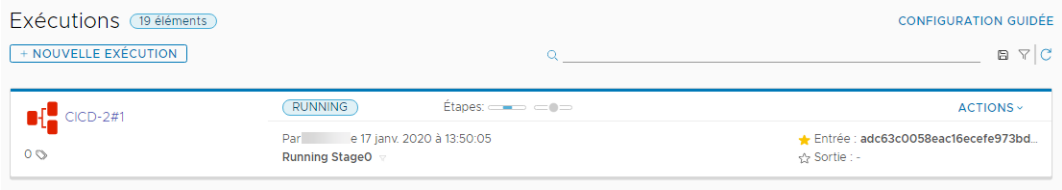
5 Revenez à Code Stream pour afficher l'activité sur le Webhook Git. Cliquez sur **Déclencheurs > Git > Activité**.

Sous État de l'exécution, vérifiez que l'exécution du pipeline a démarré.



6 Cliquez sur **Exécutions** et suivez la progression de votre pipeline au fil de son exécution.

Pour observer l'exécution du pipeline, vous pouvez cliquer sur Actualiser.



Résultats

Félicitations ! Vous avez correctement utilisé le déclencheur pour Git.

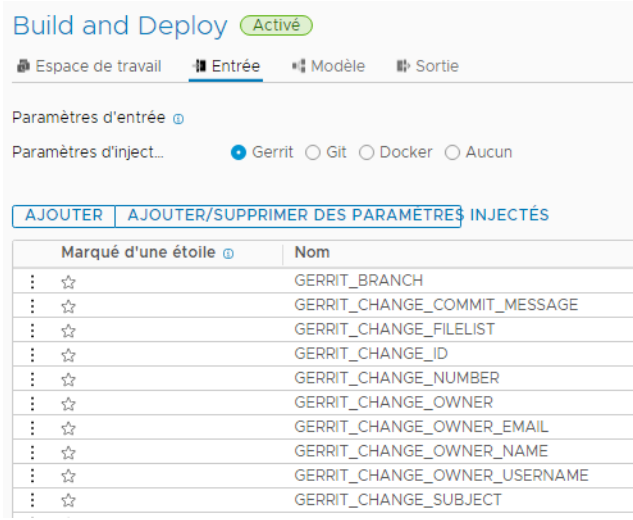
Utilisation du déclencheur Gerrit dans Code Stream pour exécuter un pipeline

En tant qu'administrateur ou développeur de Code Stream, vous pouvez intégrer Code Stream au cycle de vie de révision du code Gerrit à l'aide du déclencheur 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 Gerrit, vous sélectionnez un écouteur Gerrit et un projet Gerrit sur le serveur Gerrit et vous 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 Code Stream. Si ce n'est pas le cas, demandez à un administrateur de Code Stream de vous ajouter en tant que membre d'un projet. Reportez-vous à la section Ajout d'un projet dans Code Stream.
- Vérifier que vous disposez d'un point de terminaison Gerrit configuré dans Code Stream. Reportez-vous à la section Intégration de Code Stream à Gerrit.
- Vérifiez que vous connaissez la version de Gerrit.
- Pour que les pipelines se déclenchent, vérifiez que vous définissez les propriétés d'entrée du pipeline sur **Gerrit** ce qui permet au pipeline de recevoir les paramètres Gerrit en tant qu'entrées lors de l'exécution du pipeline.



Pour plus d'informations sur les paramètres d'entrée, reportez-vous à la section Planification d'une build native CICD dans Code Stream avant d'ajouter des tâches manuellement.

Procédure

1 Dans 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 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**.

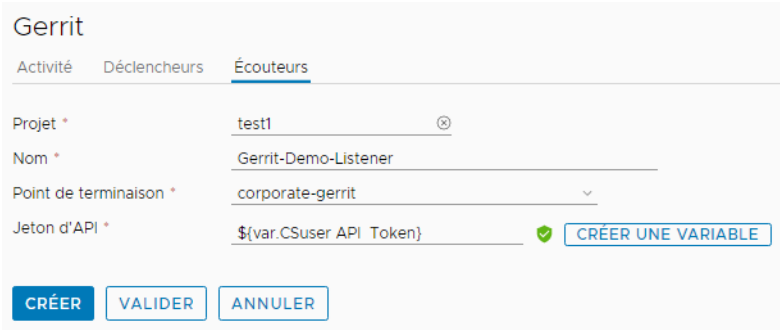
Le jeton généré est valide pendant six mois. Il est également appelé jeton d'actualisation.

- 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 en utilisant une liaison dollar. 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'électeur connecté au point de terminaison que vous souhaitez mettre à jour.

3 Cliquez sur l'onglet **Déclencheurs**, puis sur **Nouveau déclencheur**.

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.

En utilisant l'écouteur Gerrit, Code Stream fournit une liste de projets Gerrit disponibles sur le serveur.

7 Sélectionnez un projet sur le serveur Gerrit.

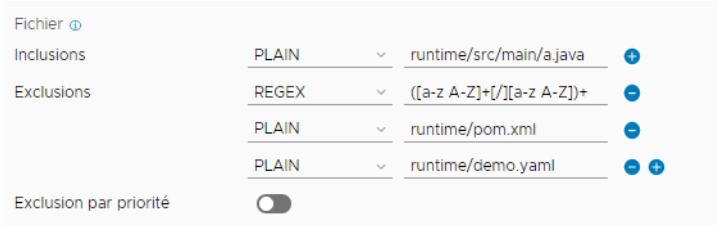
8 Entrez la branche dans le référentiel que l'écouteur Gerrit doit surveiller.

9 (Facultatif) Précisez des inclusions ou des exclusions de fichier comme conditions pour le déclencheur.

- Vous fournissez des inclusions de fichiers qui déclenchent les pipelines. 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, 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 qui empêchent 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.
- Lorsqu'il est activé, le paramètre **Exclusion par priorité** garantit que les pipelines ne se déclenchent pas, même si l'un des fichiers d'une validation correspond aux fichiers spécifiés dans les chemins d'exclusion ou regex. Le paramètre **Exclusion par priorité** est désactivé par défaut.

Si les conditions répondent à la fois à l'inclusion et à l'exclusion du fichier, les pipelines ne se déclenchent pas.

Dans l'exemple suivant, les inclusions de fichiers et les exclusions de fichiers sont des conditions pour le déclencheur.



- Pour les inclusions de fichiers, une validation qui comporte 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 qui comporte des modifications dans les deux fichiers uniquement ne déclenche pas les pipelines configurés dans la configuration des événements.

10 Cliquez sur **Nouvelle configuration**.

a 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**.

Note À partir de Gerrit 2.15, les modifications de brouillon et les ensembles de modifications de brouillon ne sont plus pris en charge. Par conséquent, si vous exécutez Gerrit 2.15 ou une version ultérieure, **Version préliminaire publiée** n'est pas un événement pris en charge.

b Sélectionnez le pipeline qui se déclenchera.

Si le pipeline inclut des paramètres d'entrée ajoutés personnalisés, la liste 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. Vous pouvez également 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 dans la configuration du déclencheur ne changent pas si les valeurs de paramètre dans le modèle de pipeline changent.

Pour plus d'informations sur les paramètres d'entrée injectés automatiquement pour les déclencheurs Gerrit, reportez-vous aux conditions préalables.

c 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.

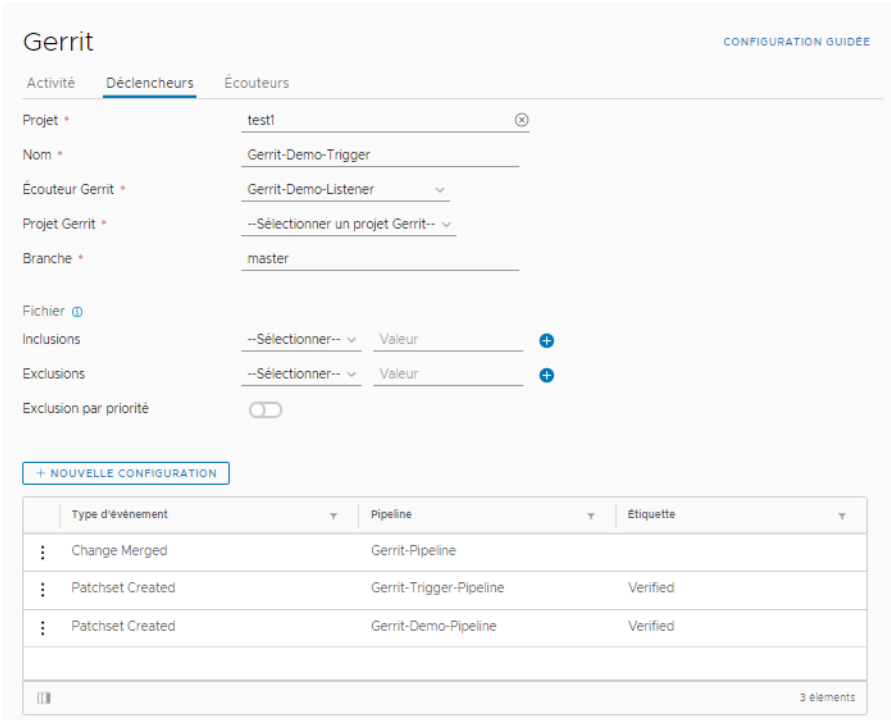
La configuration de l'événement Gerrit vous permet d'entrer des commentaires à l'aide d'une variable pour le commentaire de réussite ou le commentaire d'échec. Par exemple : `${var.success}` et `${var.failure}`.

d 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, le pipeline nommé **Pipeline-Gerrit** se déclenche.
- Si un événement **Jeu de correctifs créé** se produit dans le projet Gerrit, les pipelines nommés **Pipeline-Déclencheur-Gerrit** et **Pipeline-Démo-Gerrit** se déclenchent.



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**.

Après avoir activé le déclencheur, ce dernier peut utiliser l'écouteur Gerrit pour commencer à surveiller les événements qui se produisent sur la branche du projet Gerrit.

Pour créer un déclencheur ayant les mêmes conditions d'inclusion ou d'exclusion de fichiers, mais avec un référentiel différent de celui que vous avez inclus lorsque vous avez créé le déclencheur, sur la fiche du déclencheur, cliquez sur **Actions > Cloner**. Ensuite, sur le déclencheur cloné, cliquez sur **Ouvrir**, puis modifiez les paramètres.

Résultats

Félicitations ! Vous avez correctement configuré un déclencheur Gerrit avec deux événements sur trois pipelines différents.

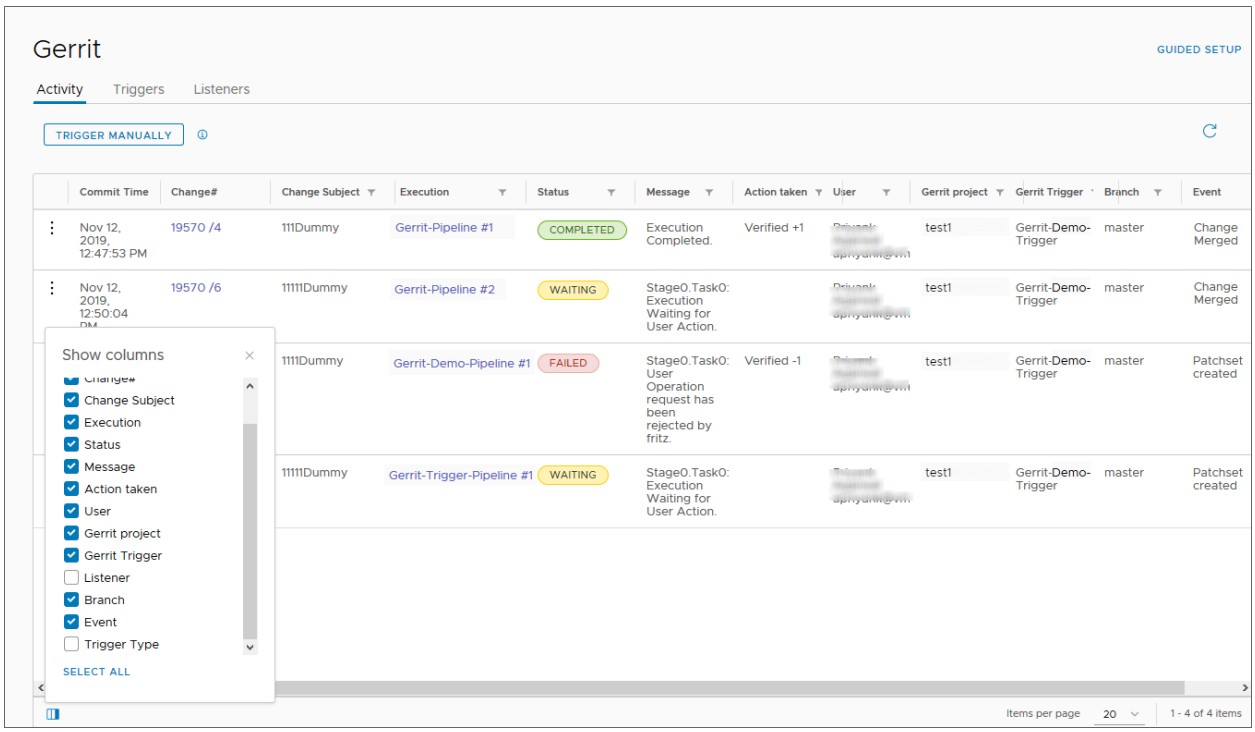
Étape suivante

Après avoir validé une modification de code dans le projet Gerrit, observez l'onglet **Activité** de l'événement Gerrit dans Code Stream. Vérifiez que la liste des activités inclut des entrées qui correspondent à chaque exécution de pipeline dans la configuration du déclencheur.

Lorsqu'un événement se produit, seuls les pipelines du déclencheur Gerrit qui se rapportent au type d'événement particulier peuvent s'exécuter. 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 sélectionner les colonnes qui s'affichent en cliquant sur l'icône de colonne qui s'affiche sous le tableau.

- Les colonnes **Modifier l'objet** et **Exécution** sont vides lorsque 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. Lorsque vous la sélectionnez, la colonne affiche l'écouteur Gerrit qui a reçu l'événement. Un même écouteur peut apparaître comme associé à plusieurs déclencheurs.
- La colonne **Type de déclencheur** est désactivée par défaut. Lorsque vous la sélectionnez, la colonne affiche le type de déclencheur AUTOMATIQUE ou MANUEL.
- Les autres colonnes incluent les éléments suivants : **Heure de validation**, **Numéro de modification**, **État**, **Message**, **Action effectuée**, **Utilisateur**, **Projet Gerrit**, **Branche** et **Événement**.



Pour contrôler l'activité correspondant à une exécution de pipeline 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 et accélérer le processus d'exécution.
- Utilisez **Afficher l'exécution**, qui ouvre la vue d'exécution du pipeline. 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 **Déclencher manuellement**, puis sélectionnez le déclencheur Gerrit, entrez l'ID de modification et cliquez sur **Exécuter**.

Surveillance des pipelines dans Code Stream

8

En tant qu'administrateur ou développeur de Code Stream, vous avez besoin de visualiser les performances de vos pipelines dans 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, vous utilisez des tableaux de bord 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.

- 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.

Ce chapitre contient les rubriques suivantes :

Présentation du tableau de bord de pipeline dans Code Stream

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. Code Stream crée le tableau de bord de pipeline lorsque vous créez un pipeline.

Le tableau de bord contient les widgets qui affichent les résultats de l'exécution du pipeline.

Widget Nombre d'états d'exécution du pipeline

Vous pouvez afficher le nombre total d'exécutions d'un pipeline sur une période de temps regroupés par état : Terminé, Échec ou Annulé. Pour voir comment l'état d'exécution du pipeline a changé sur des périodes plus ou moins longues, modifiez la durée sur l'affichage.

Widget Statistiques d'exécution du pipeline

Les statistiques d'exécution du pipeline incluent les durées moyennes avant la récupération, la livraison ou l'échec d'un pipeline au fil du temps.

Les états suivants s'appliquent à toutes les exécutions de pipeline :

- Terminé
- Échec
- Attente
- En cours d'exécution
- Annulé
- En file d'attente
- Non démarré
- Restauration
- Restauration terminée
- Échec de la restauration
- Suspendu(e)

Tableau 8-1. Mesure des durées moyennes

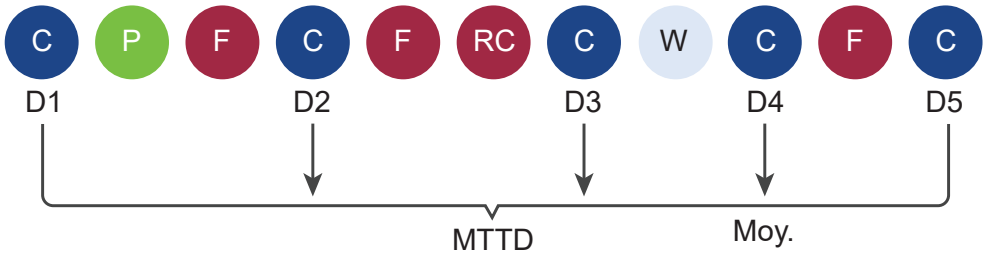
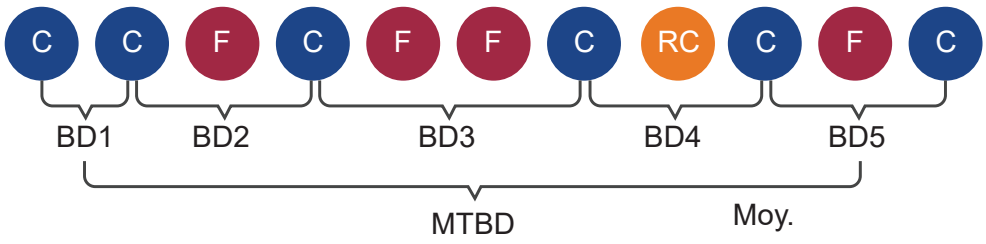
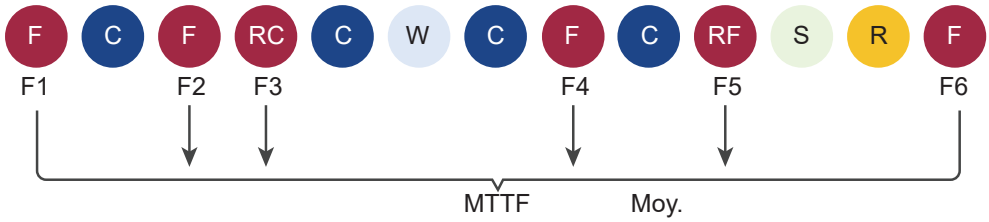
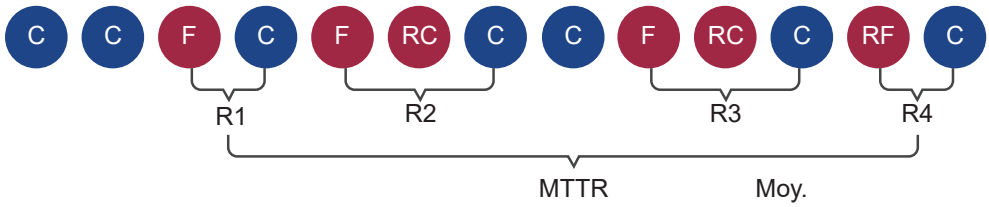
Éléments mesurés...	Signification
CI moyenne	Durée moyenne passée en phase d'intégration continue, mesurée par heure dans le type de tâche CI.
Durée moyenne avant livraison (MTTD)	Durée moyenne de toutes les exécutions à l'état TERMINÉ sur une période donnée. D1, D2, etc. correspondent à la durée de livraison de chaque exécution à l'état TERMINÉ. 

Tableau 8-1. Mesure des durées moyennes (suite)

Éléments mesurés...	Signification
Durée moyenne entre deux livraisons (MTBD)	Durée moyenne écoulée entre les livraisons réussies sur une période donnée. La durée écoulée entre deux opérations à l'état TERMINÉ consécutives est la durée entre deux livraisons réussies, telles que BD1, BD2, etc. MTBD indique la fréquence de mise à jour d'un environnement de production. 
Durée moyenne avant échec (MTTF)	Durée moyenne des exécutions qui se terminent par des états ÉCHEC, RESTAURATION TERMINÉE ou ÉCHEC DE LA RESTAURATION sur une période donnée. F1, F2, etc. correspondent à la durée prise par une exécution pour terminer à l'état ÉCHEC, RESTAURATION TERMINÉE ou ÉCHEC DE LA RESTAURATION. 
Durée moyenne avant récupération (MTTR)	Durée moyenne de récupération après un échec sur une période donnée. La durée moyenne de récupération après un échec est la durée écoulée entre une opération avec un état final ÉCHEC, RESTAURATION TERMINÉE ou ÉCHEC DE LA RESTAURATION et la prochaine opération réussie consécutive avec un état TERMINÉ. R1, R2, etc. correspondent à la durée de récupération après chaque exécution à l'état ÉCHEC ou ÉCHEC DE LA RESTAURATION. 

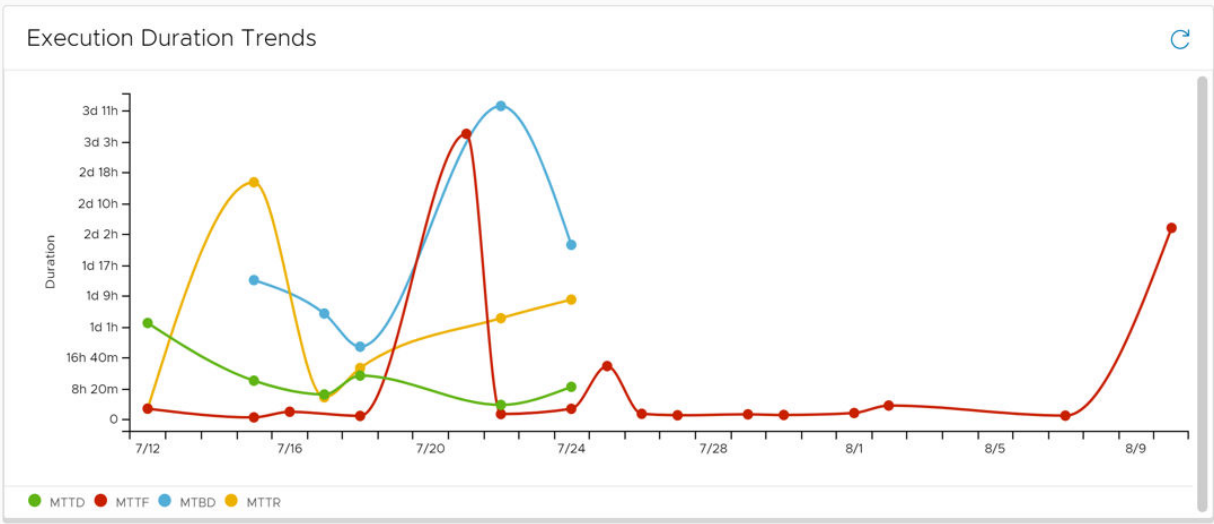
Widgets Principaux échecs d'étape et de tâche

Deux widgets affichent les principaux échecs d'étape et de tâche dans un pipeline. Chaque 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 exécutiez 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.

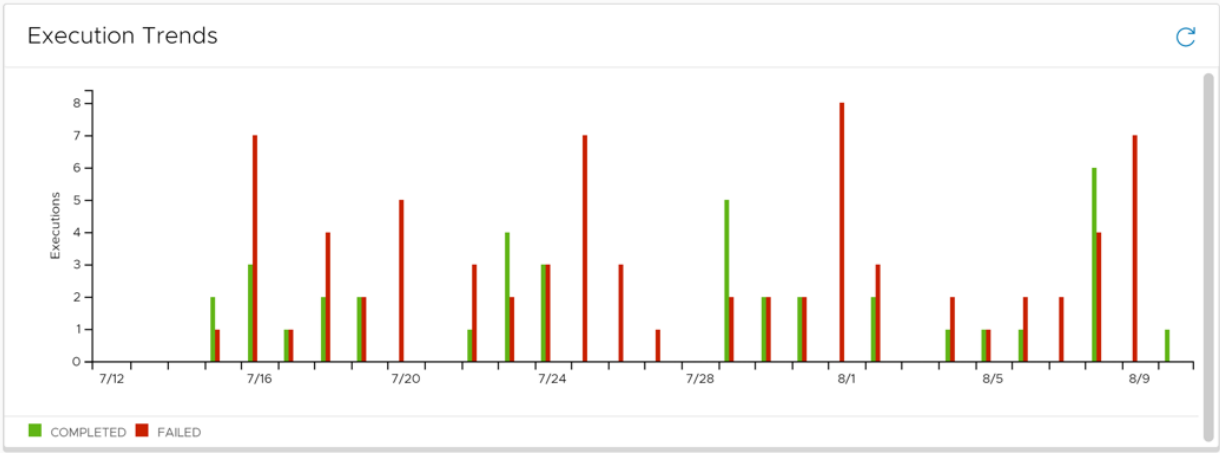
Widget Tendances de durée d'exécution du pipeline

Les tendances de durée d'exécution du pipeline affichent le MTTD, le MTTF, le MTBD et le MTTR sur une période donnée.



Widget Tendances d'exécution du pipeline

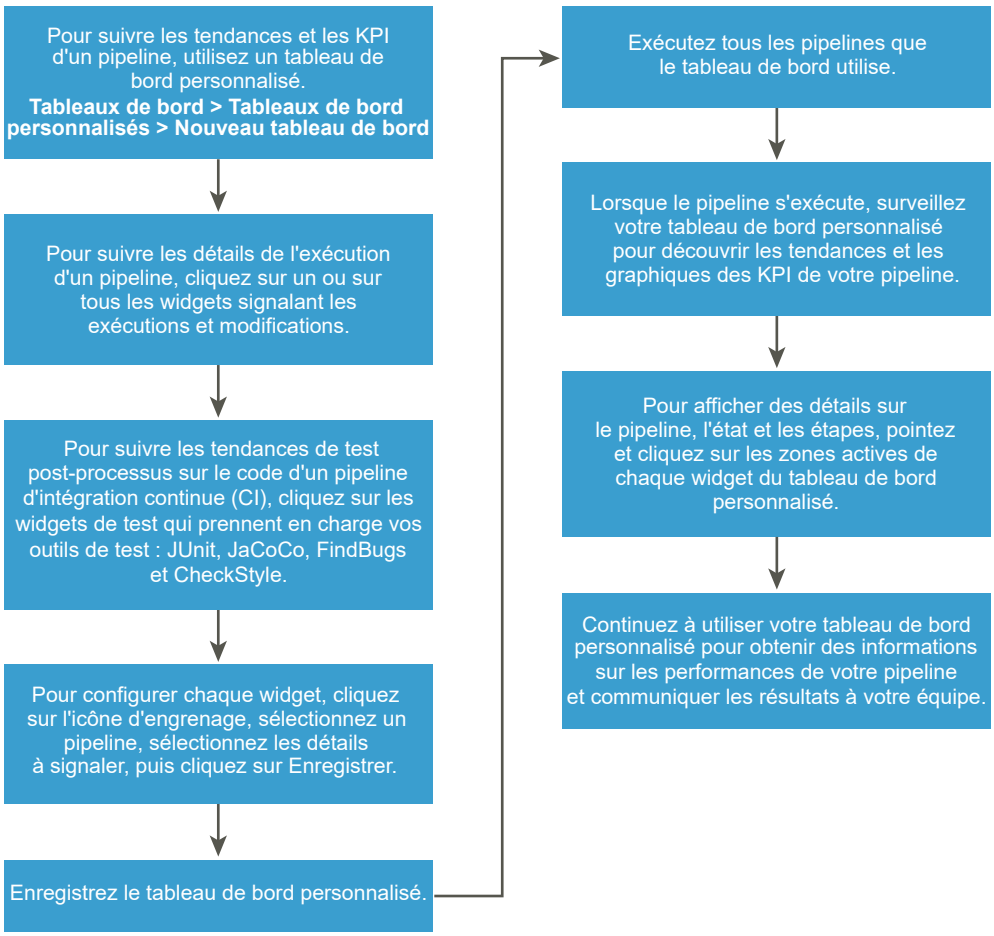
Les tendances d'exécution du pipeline affichent le nombre total d'exécutions quotidiennes d'un pipeline, regroupées par état sur une période donnée. À l'exception du jour actuel, la plupart des nombres d'agrégations quotidiennes indiquent les exécutions à l'état TERMINÉ et ÉCHEC uniquement.



Utilisation des tableaux de bord personnalisés pour suivre les indicateurs de performance clés d'un pipeline dans Code Stream

En tant qu'administrateur ou développeur Code Stream, vous créez un tableau de bord personnalisé pour afficher les résultats que vous souhaitez visualiser pour l'exécution d'un ou de plusieurs pipelines. 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.

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

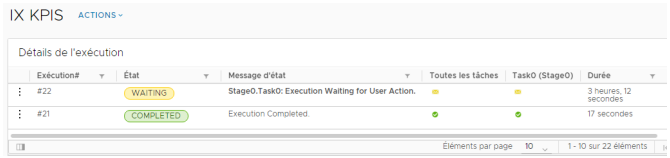
- Vérifiez qu'au moins un pipeline existe. Dans l'interface utilisateur, cliquez sur **Pipelines**.
- Pour les pipelines que vous souhaitez surveiller, vérifiez qu'ils ont bien été exécutés. Cliquez sur **Exécutions**.

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.



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 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 Code Stream d'en savoir plus sur 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 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 Code Stream
- Plus de ressources pour les administrateurs et les développeurs Code Stream

Présentation de la recherche dans 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 : `<key>:<value>`
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>`

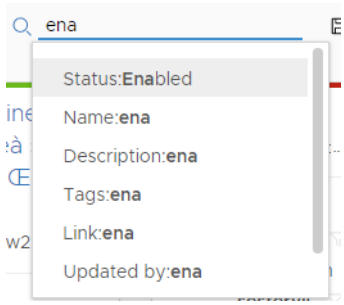
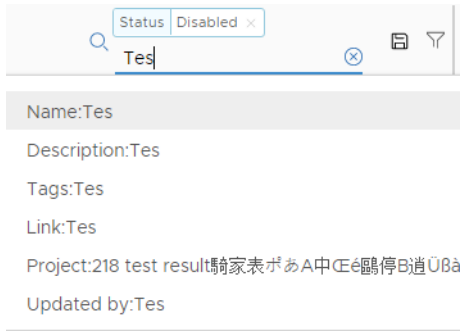
Exemple : **`output.BuildNo:29`**

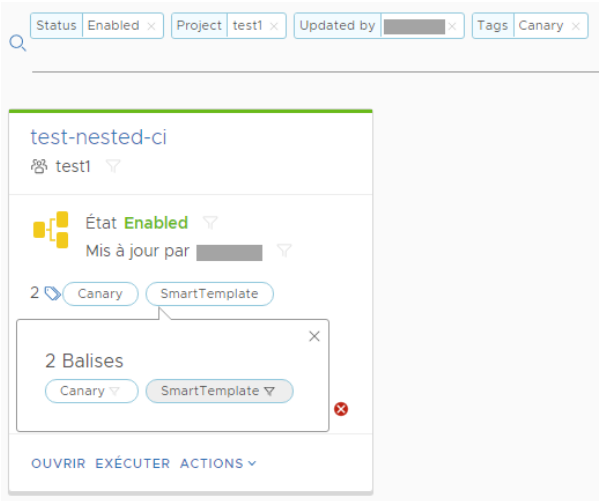
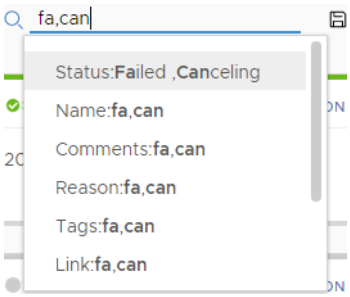
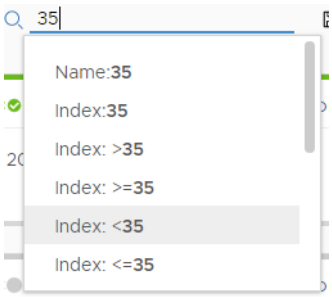
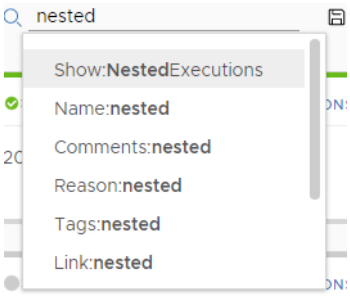
- Notation : `statusMessage:<value>`

  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, Code Stream connaît le contexte de la page et suggère des options de recherche.

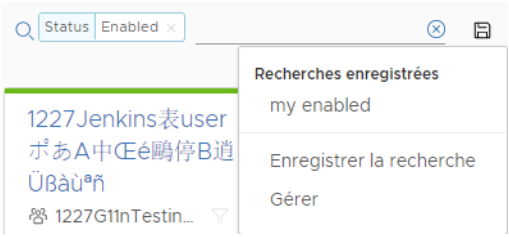
Méthodes de recherche	Modalités de saisie
Tapez une partie du paramètre de recherche. Par exemple, pour ajouter un filtre d'état répertoriant tous les pipelines activés, saisissez **`act`**.	
Pour réduire le nombre d'éléments trouvés, ajoutez un filtre. Par exemple, tapez **`Tes`** pour ajouter un filtre de nom. Le filtre fonctionne comme un `ET` avec le filtre **État:désactivé** existant pour afficher uniquement les pipelines désactivés dont le nom comporte **`Tes`**. Lorsque vous ajoutez un autre filtre, les options restantes s'affichent : **Nom**, **Description**, **Balises**, **Lien**, **Projet** et **Mis à jour par**.	

Méthodes de recherche	Modalités de saisie
Pour réduire le nombre d'éléments affichés, cliquez sur l'icône de filtre sur les propriétés d'un pipeline ou sur l'exécution d'un pipeline. ■ 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**.	
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.

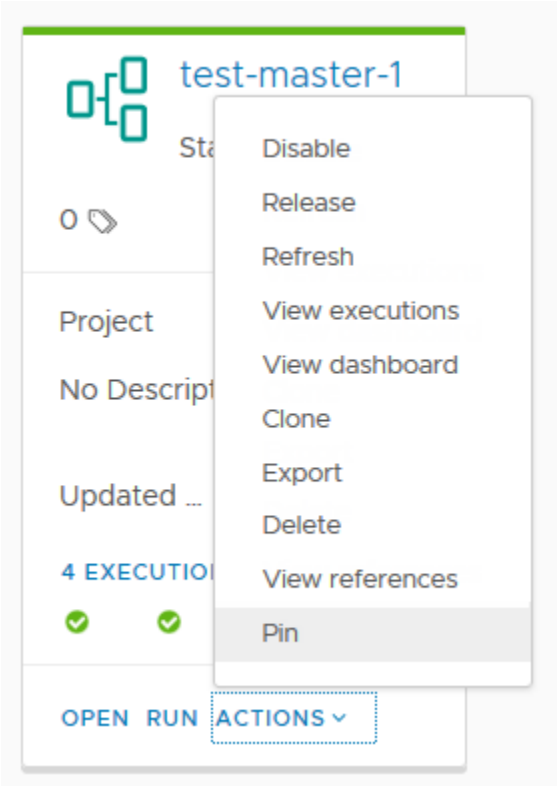
- Pour enregistrer une recherche, saisissez les paramètres de recherche et cliquez sur l'icône pour donner un nom à la recherche, comme **`Mes pipelines activés`**.
- Après avoir enregistré une recherche, cliquez sur l'icône pour accéder à la recherche. Vous pouvez également sélectionner **Gérer** pour 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 Code Stream

En tant qu'administrateur ou développeur de Code Stream, vous pouvez approfondir vos connaissances sur Code Stream.

Tableau 9-1. Plus de ressources pour les administrateurs

Pour en savoir plus sur…	Consultez ces ressources…
Les autres méthodes mises à la disposition des administrateurs pour utiliser Code Stream : ■ Configurer les pipelines pour automatiser les tests et la libération des applications natives de cloud. ■ Automatiser et tester le code source des développeurs, par le biais des tests, jusqu'à la production. ■ Configurer les pipelines pour que les développeurs puissent tester les modifications avant de les valider dans la branche principale. ■ Effectuer le suivi des mesures clés de pipeline.	Code Stream ■ Documentation de vRealize Automation ■ Site Web produit vRealize Automation Ateliers VMware ■ Utilisez la Communauté vRealize Automation. ■ Utilisez la Zone d'apprentissage VMware. ■ Consultez les Blogs VMware. ■ Inscrivez-vous aux Ateliers pratiques VMware.

Tableau 9-2. Plus de ressources pour les développeurs

Pour en savoir plus sur…	Consultez ces ressources…
Les autres méthodes mises à la disposition des développeurs pour utiliser Code Stream : ■ Utiliser les images de registre public et privé pour construire les environnements pour les nouvelles applications ou les nouveaux services. ■ Configurer les environnements de développement pour pouvoir créer des branches à partir du dernier build stable. ■ Mettre à jour les environnements de développement avec les modifications de code et les artefacts les plus récents. ■ Tester les modifications du code non validées par rapport aux derniers builds stables des autres services dépendants. ■ Recevoir une notification lorsqu'une modification validée dans un pipeline CICD principal rompt les autres services.	Code Stream ■ Documentation de vRealize Automation ■ Site Web produit vRealize Automation Ateliers VMware ■ Utilisez la Communauté vRealize Automation. ■ Utilisez la Zone d'apprentissage VMware. ■ Consultez les Blogs VMware. ■ Inscrivez-vous aux Ateliers pratiques VMware.