

Использование и настройка vRealize Automation Code Stream

14 декабря 2022 г.
vRealize Automation 8.7

Актуальная техническая документация доступна на веб-сайте VMware:

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

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

VMware Россия
Россия, 125284, г. Москва
ул. Беговая, д.3, стр.1
Бизнес-центр "NORDSTAR TOWER" 30й этаж
Телефон: +7 495 212 29 00
www.vmware.com/ru

© 2022 VMware, Inc. Все права защищены. [Информация об авторских правах и товарных знаках.](#)

Содержание

1	Что такое Code Stream, и как это работает	5
2	Настройка для моделирования процесса выпуска	10
	Добавление проекта	15
	Управление пользовательским доступом и утверждениями	16
	Что такое пользовательские операции и подтверждения	25
3	Создание и использование конвейеров	27
	Запуск конвейера и просмотр результатов	30
	Доступные типы задач	35
	Как использовать привязки переменных в конвейерах	40
	Использование привязок переменных в задаче «Условие» для запуска или остановки конвейера	50
	Какие переменные и выражения можно использовать для связывания задач конвейера	53
	Отправка уведомлений о работе конвейера	71
	Создание запроса Jira при неудачном выполнении задачи конвейера	74
	Как откатить развертывание	77
4	Планирование сборки, интеграции и предоставления кода встроенными средствами	84
	Настройка рабочей области конвейера	84
	Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки перед использованием смарт-шаблона конвейера	88
	Планирование собственной сборки по модели непрерывной интеграции перед использованием смарт-шаблона конвейера	95
	Планирование собственной сборки по модели непрерывного предоставления перед использованием смарт-шаблона конвейера	96
	Планирование собственной сборки по модели непрерывной интеграции и предоставления перед добавлением задач вручную	98
	Планирование отката	104
5	Учебники	107
	Настройка непрерывной интеграции кода из репозитория GitHub или GitLab в конвейер	108
	Как автоматизировать выпуск приложения, развернутого с использованием облачного шаблона YAML	112
	Автоматизация выпуска приложения в кластере Kubernetes	120
	Как развернуть приложение в среде Blue-Green	128
	Интеграция собственных средств сборки, тестирования и развертывания	133
	Как использовать свойства ресурса из задачи облачного шаблона в следующей задаче	144
	Как использовать интерфейс REST API для интеграции с другими приложениями	148
	Использование конвейера в виде кода	153

6 Подключение к конечным точкам 159

Что такое конечные точки 159

Интеграция с Jenkins 162

Интеграция с Git 168

Интеграция с Gerrit 170

Интеграция с решением vRealize Orchestrator 174

7 Запуск конвейеров 180

Как использовать триггер Docker для запуска конвейера непрерывного предоставления 180

Как использовать триггер Git для запуска конвейера 189

Как использовать триггер Gerrit для запуска конвейера 197

8 Мониторинг конвейеров 206

Элементы, отображаемые на панели управления конвейера 206

Отслеживание ключевых показателей эффективности с помощью настраиваемых панелей управления 210

9 Подробнее 212

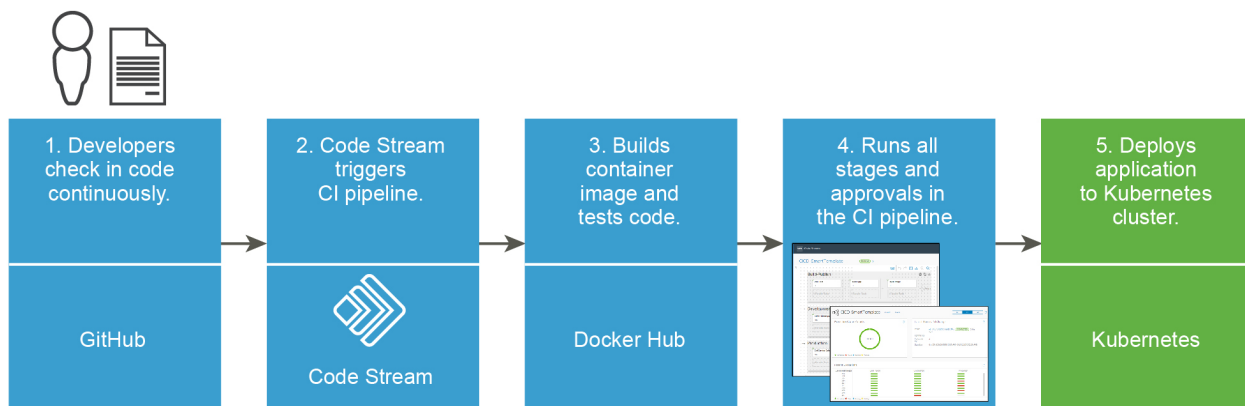
Что такое поиск 212

Дополнительные ресурсы для администраторов и разработчиков 218

Что такое Code Stream, и как это работает

1

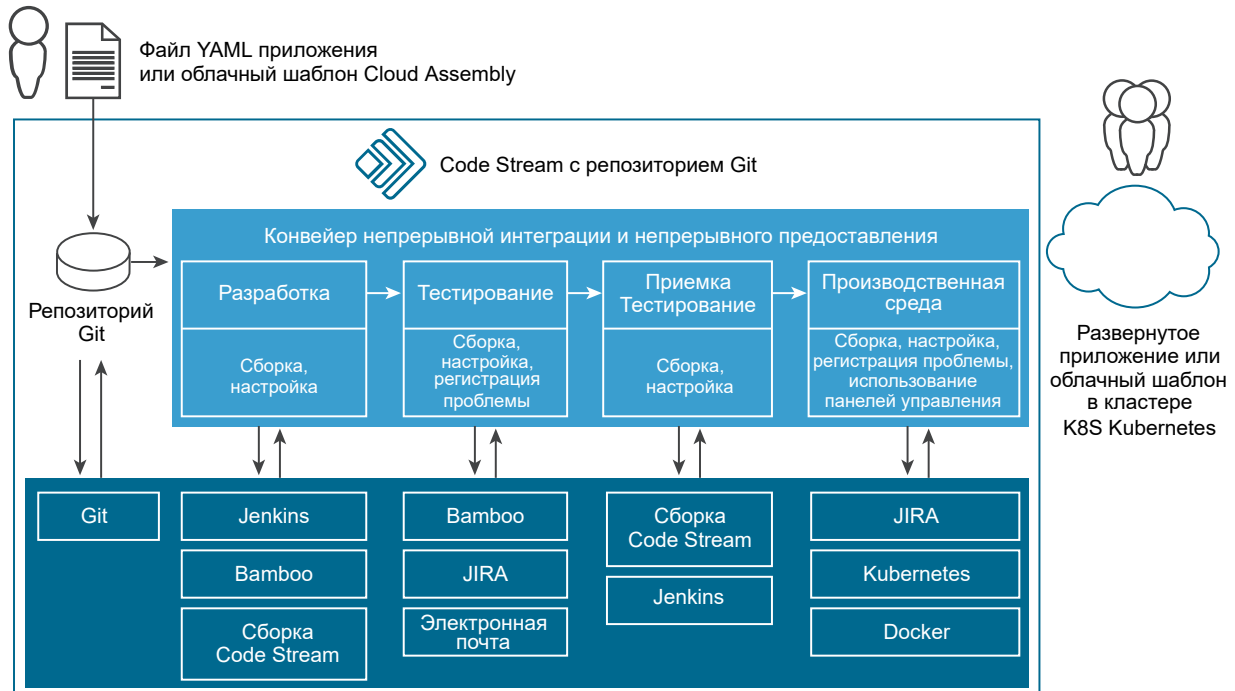
vRealize Automation Code Stream™ — это средство непрерывной интеграции и непрерывной доставки (CI/CD). Благодаря созданию конвейеров, моделирующих процесс выпуска программного обеспечения в жизненном цикле DevOps, это решение позволяет создавать инфраструктуру кода, которая обеспечивает быструю непрерывную доставку программного обеспечения.



При использовании Code Stream для доставки программного обеспечения необходимо интегрировать два наиболее важных этапа жизненного цикла DevOps: процесс выпуска и средства разработчика. После первоначальной установки, которая интегрирует Code Stream с существующими средствами разработки, конвейеры автоматизируют весь жизненный цикл DevOps.

Начиная с vRealize Automation 8.2, схемы элементов называются шаблонами VMware Cloud Templates.

Создается конвейер, который выполняет сборку, тестирование и выпуск программного обеспечения. В Code Stream этот конвейер используется для переноса ПО из репозитория исходного кода в среду тестирования, и затем — в производственную среду.



Дополнительные сведения о планировании конвейеров непрерывной интеграции и непрерывного предоставления см. в [Глава 4 Планирование сборки, интеграции и предоставления кода встроенными средствами в Code Stream](#).

Как администраторы Code Stream используют Code Stream

В обязанности администратора входит создание конечных точек и обеспечение доступа к рабочим экземплярам для разработчиков. Можно создавать, запускать конвейеры, управлять ими и т. д. У вас есть роль **Administrator**, как описано в разделе [Управление пользовательским доступом и утверждениями в службе Code Stream](#).

Таблица 1-1. Как администраторы Code Stream поддерживают разработчиков

Для поддержки разработчиков...	Действия...
Предоставление сред и управление ими.	<p>Создание сред для разработчиков для тестирования и развертывания кода.</p> <ul style="list-style-type: none"> ■ Отслеживание состояния и отправка уведомлений по электронной почте. ■ Поддержка эффективной работы разработчиков путем обеспечения непрерывного функционирования их рабочей среды. <p>Дополнительные сведения см. в разделе Дополнительные ресурсы для администраторов и разработчиков Code Stream.</p> <p>См. также Глава 5 Учебники по использованию Code Stream.</p>
Предоставьте конечные точки.	<p>Убедитесь, что у разработчиков есть рабочие экземпляры конечных точек, которые могут подключаться к их конвейерам.</p>
Обеспечение интеграции с другими службами.	<p>Убедитесь, что интеграция с другими службами работает.</p> <p>Дополнительные сведения см. в разделе документации по VMware Cloud Services.</p>

Таблица 1-1. Как администраторы Code Stream поддерживают разработчиков (продолжение)

Для поддержки разработчиков...	Действия...
Создание конвейеров	<p>Создайте конвейеры, которые моделируют процессы выпуска.</p> <p>Дополнительные сведения см. в разделе Глава 3 Создание и использование конвейеров в Code Stream.</p>
Запустите конвейеры.	<p>Убедитесь, что конвейеры выполняются при возникновении событий.</p> <ul style="list-style-type: none"> ■ Чтобы запустить автономный конвейер непрерывной доставки (CD) при создании или обновлении артефакта сборки, используйте триггер Docker. ■ Чтобы запустить конвейер, когда разработчик фиксирует изменения в своем коде, используйте триггер Git. ■ Чтобы запустить конвейер, когда разработчики проверяют код, объединяют его и выполняют другие операции, используйте триггер Gerrit. ■ Чтобы запустить автономный конвейер непрерывного предоставления (CD) при создании или обновлении артефакта сборки, используйте триггер Docker. <p>Дополнительные сведения см. в разделе Глава 7 Запуск конвейеров в Code Stream.</p>
Управляйте конвейерами и утверждениями.	<p>Отслеживайте работу конвейеров.</p> <ul style="list-style-type: none"> ■ Просматривайте сведения о состоянии конвейеров и пользователях, которые их запускали. ■ Просматривайте утверждения для выполнений конвейера, управляйте утверждениями для активных и неактивных выполнений конвейера. <p>Дополнительные сведения см. в разделе Что такое пользовательские операции и утверждения в Code Stream.</p> <p>См. также Отслеживание ключевых показателей эффективности конвейера в Code Stream с помощью настраиваемых панелей управления.</p>
Отслеживайте среды разработчиков.	<p>Создавайте настраиваемые панели управления, которые позволяют отслеживать состояние конвейера, тенденции, показатели и КПЭ. Используйте настраиваемые панели управления для отслеживания конвейеров, которые успешно проходят или не проходят проверки в средах разработчиков. Также можно выявить недостаточно используемые ресурсы, подготовить по ним отчеты и высвободить ресурсы.</p> <p>Здесь также можно отслеживать следующие показатели:</p> <ul style="list-style-type: none"> ■ Период времени, в течение которого конвейер работал до успешного выполнения. ■ Период ожидания утверждения конвейера и уведомления пользователя, который должен выполнить утверждение. ■ Наиболее нестабильные этапы и задачи. ■ Этапы и задачи с наибольшим временем выполнения. ■ Выпуски, над которыми работают группы разработчиков. ■ Приложения, которые были успешно развернуты и выпущены. <p>Дополнительные сведения см. в разделе Глава 8 Мониторинг конвейеров в Code Stream.</p>
Устранение проблем.	<p>Обнаружение и устранение сбоев конвейера в средах разработчика.</p> <ul style="list-style-type: none"> ■ Обнаружение и решение проблем в средах с непрерывной интеграцией и непрерывной доставкой (CICD). ■ Использование панелей управления конвейера и создание настраиваемых панелей управления для получения дополнительных сведений. См. раздел Глава 8 Мониторинг конвейеров в Code Stream. <p>См. также Глава 2 Настройка Code Stream для моделирования процесса выпуска.</p>

Code Stream является частью VMware Cloud Services.

- Для развертывания облачных шаблонов используйте Cloud Assembly.
- Для получения облачных шаблонов из каталога используйте Service Broker.

Сведения о других возможностях решения см. в [документации по VMware vRealize Automation](#).

Как разработчики используют Code Stream

Разработчик может использовать Code Stream для создания и запуска конвейеров, а также для мониторинга работы конвейеров с помощью панелей управления. У вас есть роль User, как описано в разделе [Управление пользовательским доступом и утверждениями в службе Code Stream](#).

После завершения процессов конвейера вам необходима следующая информация.

- Прошел ли код через все этапы конвейера? Чтобы получить ответ на этот вопрос, ознакомьтесь с результатами циклов выполнения конвейера.
- Что делать, если произошел сбой конвейера, и как определить, что его вызвало. Чтобы получить ответ на этот вопрос, ознакомьтесь с самыми частыми ошибками на панелях управления конвейером.

Таблица 1-2. Разработчики, использующие Code Stream

Для интеграции и выпуска кода	Действия
Сборка конвейеров.	Тестирование и развертывание кода. Обновление кода при сбое конвейера.
Подключение конвейера к конечным точкам.	Подключение задач в конвейере к конечным точкам, например к репозиторию GitHub.
Запустите конвейеры.	Добавьте задачу «утверждение пользовательской операции», чтобы другой пользователь мог подтвердить конвейер в определенных точках.
Просмотр данных на панелях управления	Просмотрите результаты на панели управления конвейера. Также можно увидеть тенденции, историю, ошибки и т. д.

Дополнительные сведения о том, как начать работу, см. в разделе [Начало работы с VMware Code Stream](#).

Дополнительные сведения см. в документации на панели «Поддержка продукта»

Если здесь нет необходимой информации, дополнительную помощь можно получить внутри продукта. 

- Щелкните и изучите справку по указателям и подсказки в пользовательском интерфейсе, чтобы своевременно получить нужную информацию.
- Откройте панель поддержки продукта и ознакомьтесь с разделами, доступными для активной страницы пользовательского интерфейса. Для получения ответов на вопросы можно также выполнить поиск на панели.

Дополнительные сведения о веб-перехватчиках

Можно создать несколько веб-перехватчиков для различных ветвей с помощью одной конечной точки Git, указав разные значения для имени ветви на странице конфигурации веб-перехватчика. Чтобы создать другой веб-перехватчик для другой ветви в том же репозитории Git, не нужно несколько раз клонировать конечную точку Git для нескольких ветвей. Просто укажите имя ветви в веб-перехватчике, чтобы использовать конечную точку Git еще раз. Если ветвь веб-перехватчика Git совпадает с ветвью в конечной точке, не нужно указывать имя ветви на странице веб-перехватчика Git.

Настройка Code Stream для моделирования процесса выпуска

2

Чтобы смоделировать процесс выпуска, необходимо создать конвейер, который содержит этапы, задачи и утверждения, обычно применяемые для выпуска программного обеспечения. Затем Code Stream автоматизирует процесс, который выполняет сборку, тестирует, утверждает и развертывает код.

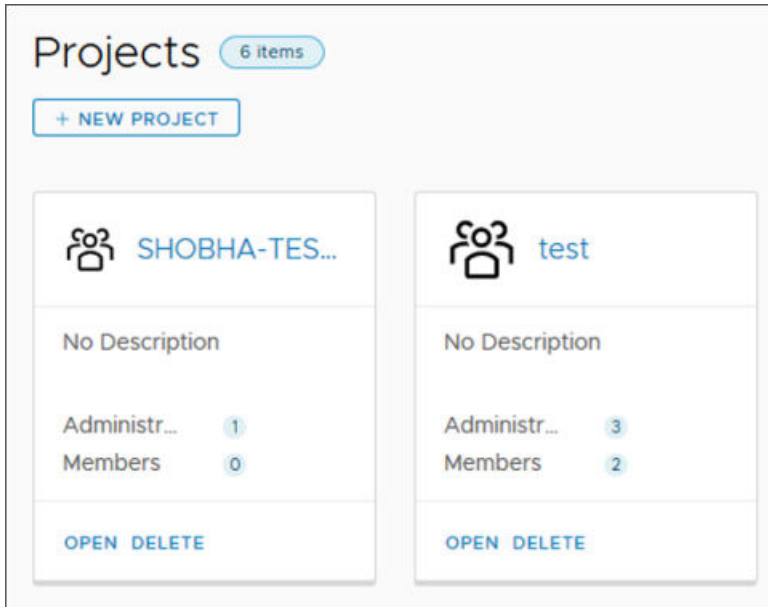
После того как все подготовлено, выполните моделирование процесса выпуска программного обеспечения в Code Stream следующим образом.

Необходимые условия

- Убедитесь, что конечные точки уже доступны. В Code Stream щелкните **Конечные точки**.
- Ознакомьтесь со встроенными способами сборки и развертывания кода. См. раздел [Глава 4 Планирование сборки, интеграции и предоставления кода встроенными средствами в Code Stream](#).
- Определите, следует ли для некоторых ресурсов, которые будут использоваться в конвейере, установить пометку «запрещенный». См. раздел [Управление пользовательским доступом и утверждениями в службе Code Stream](#).
- Если вам присвоена роль пользователя или обозревателя, а не администратора, выясните, кто является администратором экземпляра Code Stream.

Процедура

1. Проверьте проекты, доступные в Code Stream, и выберите подходящий.
 - Если в списке нет проектов, попросите администратора Code Stream создать проект и сделать вас его участником. См. раздел [Добавление проекта в Code Stream](#).
 - Если вы не являетесь участником ни одного из указанных проектов, попросите администратора Code Stream добавить вас в проект в качестве участника.



2. Добавьте любые новые конечные точки, необходимые для конвейера.

Например, могут потребоваться Git, Jenkins, Code Stream Build, Kubernetes и Jira.

3. Создайте переменные, чтобы было можно повторно использовать значения в задачах конвейера.

Используйте переменные с ограниченным доступом, чтобы ограничить доступ к ресурсам, используемым в конвейерах, таким как компьютер узла. Можно запретить запуск конвейера до его явного утверждения другим пользователем.

Администраторы могут создавать секретные переменные и переменные с ограниченным доступом. Пользователи могут создавать секретные переменные.

Можно повторно использовать переменную в рамках нескольких конвейеров столько раз, сколько необходимо. Например, переменной, которая определяет компьютер узла, может быть `HostIPAddress`. Чтобы использовать переменную в задаче конвейера, необходимо ввести `${var.HostIPAddress}`.

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

4. Если вы администратор, пометьте все конечные точки и переменные, которые являются крайне важными для вашего бизнеса, как запрещенные ресурсы.

Если пользователь, который не является администратором, пытается запустить конвейер, в котором есть ресурс с ограниченным доступом, конвейер останавливается на задаче, в которой используется такой ресурс. После этого администратор должен возобновить работу конвейера.

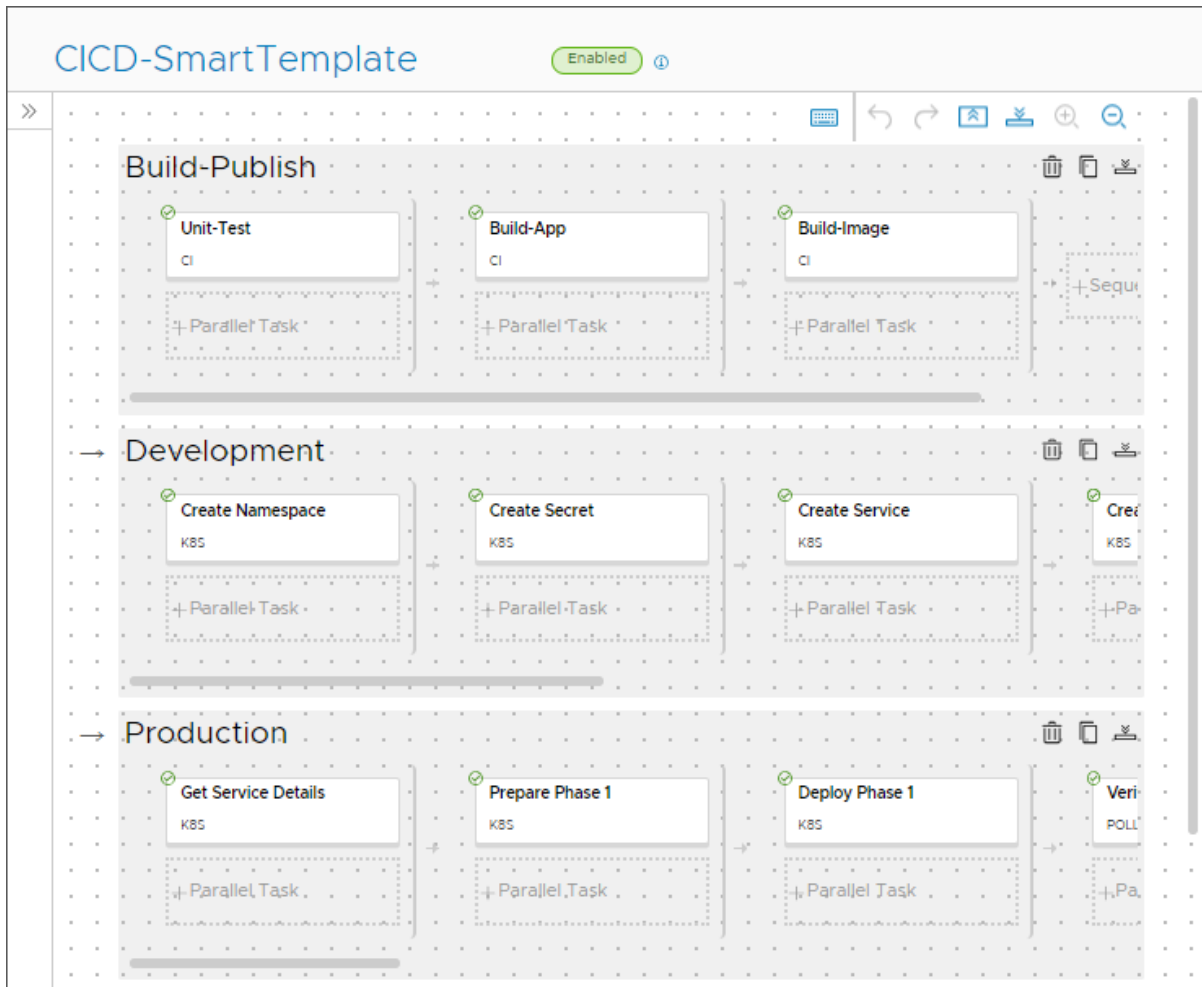
5. Планирование стратегии сборки для встроенного конвейера CICD, CI или CD.

Перед созданием конвейера, который непрерывно интегрирует (CI) и непрерывно развертывает (CD) код, спланируйте стратегию сборки. План сборки позволяет определить требования службы Code Stream для выполнения сборки, интеграции, тестирования и развертывания кода встроенными средствами.

Создание сборки встроенными средствами Code Stream	Результаты этой стратегии сборки
Используйте один из смарт-шаблонов конвейера.	<ul style="list-style-type: none"> ■ Выполняется сборка всех этапов и задач. ■ Клонировается исходный репозиторий. ■ Выполняется сборка и тестирование кода. ■ Код размещается в контейнерах для развертывания. ■ Формируются этапы, содержащие задачи конвейера, на основе выбранных параметров.
Добавьте этапы и задачи вручную.	Добавьте этапы и задачи, а также введите значения необходимых параметров.

6. Создайте конвейер с помощью смарт-шаблона конвейера или путем добавления этапов и задач в конвейер вручную.

После этого все ресурсы будут помечены как запрещенные. Добавьте утверждения, если это необходимо. Примените обычные, запрещенные или секретные переменные. Добавьте любые привязки между задачами.















7. Проверьте, включите и запустите конвейер.

8. Просмотрите циклы выполнения конвейера.

Executions 280 items

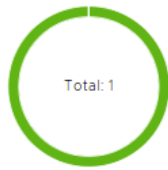
[NEW EXECUTION](#) Search Filter Refresh

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

9. Для отслеживания состояния и ключевых показателей эффективности используйте панели управления конвейера и создавайте настраиваемые панели управления.

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


Execution Status Counts Refresh



Total: 1

● Completed ● Failed ● Running ● Waiting

Latest Successful Change Refresh































When  CICD-SmartTemplate #46 **COMPLETED** a day ago

Comments -

Executed by d

Duration 6m 37s (09/06/2018 10:21 AM - 09/06/2018 10:29 AM)

Recent Executions Refresh

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

● Completed ● Failed ● Running ● Waiting

Результаты

Вы создали конвейер, который можно использовать в выбранном проекте.

Также можно экспортировать файл YAML конвейера для его импорта и повторного использования в других проектах.

Следующие шаги

Ознакомьтесь с примерами использования, которые могут быть применены в среде. См. раздел [Глава 5 Учебники по использованию Code Stream](#).

Добавление проекта в Code Stream

Вы создаете проект и добавляете в него администраторов и участников. Участники проекта могут использовать такие функции, как создание конвейера и добавление конечной точки. Чтобы создать, удалить или обновить проект для группы разработчиков, требуются права администратора Code Stream.

Проект должен быть создан до того, как будет создан конвейер. При создании конвейера необходимо выбрать проект, в котором будут объединяться все сведения о конвейере. Определения конечных точек и переменных также зависят от существующего проекта.

Необходимые условия

- Убедитесь, что вам назначена роль администратора Code Stream. См. раздел [Роли в службе Code Stream](#).

Если у вас нет роли администратора Code Stream, но вы являетесь администратором в Cloud Assembly, создавать, обновлять и удалять проекты можно через пользовательский интерфейс Cloud Assembly. См. раздел [Добавление проекта для группы разработчиков в Cloud Assembly](#).

- При добавлении групп Active Directory в проекты убедитесь, что эти группы настроены для организации. См. раздел [Включение групп Active Directory в vRealize Automation для проектов](#). Если группы не синхронизированы, при попытке добавления в проект они будут недоступны.

Процедура

1. Выберите **Проекты** и щелкните **Создать проект**.
2. Введите имя проекта.
3. Щелкните **Создать**.
4. Выберите карточку только что созданного проекта и нажмите кнопку **Открыть**.
5. Перейдите на вкладку **Пользователи**, добавьте пользователей и назначьте роли.
 - Администратор проекта может добавлять участников.
 - Участник проекта, у которого есть роль службы, может использовать службы.
 - Обзорщик проекта может просматривать проекты, но не может создавать, обновлять или удалять их.

Дополнительные сведения о ролях на уровне проекта см. в разделе [Управление пользовательским доступом и утверждениями в службе Code Stream](#).

6. Нажмите **Сохранить**.

Следующие шаги

Добавьте конечные точки и конвейеры, использующие проект. См. разделы [Глава 6 Подключение Code Stream к конечным точкам](#) и [Глава 3 Создание и использование конвейеров в Code Stream](#).

После создания конвейера имя проекта, в котором объединены все сведения о конвейере, отображается на карточках конвейера и карточках выполнения конвейера.

Управление пользовательским доступом и утверждениями в службе Code Stream

Служба Code Stream включает в себя несколько процедур, позволяющих убедиться, что все пользователи прошли проверку подлинности и дали согласие на работу с конвейерами, осуществляющими выпуск программных приложений.

Каждому участнику рабочей группы назначается роль, которая имеет определенные разрешения для работы с конвейерами, конечными точками и панелями управления, а также дает возможность ограничивать доступ к ресурсам.

Пользовательские операции и утверждения позволяют выявлять случаи, в которых работа конвейера должна быть приостановлена для получения утверждения. Роли пользователей определяют, могут ли они возобновить работу конвейера, а также запускать конвейеры, содержащие конечные точки и переменные с ограниченным доступом.

Используйте секретные переменные для скрытия и шифрования конфиденциальной информации.

Используйте переменную с ограниченным доступом для строк, паролей и URL-адресов, которые должны быть скрыты и зашифрованы, а также для ограничения возможностей использования элементов в циклах выполнения. Например, для пароля или URL-адреса. Секретные переменные и переменные с ограниченным доступом можно использовать в любом типе задач конвейера.

Роли в службе Code Stream

Роли, назначаемые пользователям в службе Code Stream, определяют, какие действия они могут выполнять и к каким областям имеют доступ. Например, роль может предоставлять право на создание, обновление и запуск конвейеров. Либо права роли могут сводиться исключительно к просмотру конвейеров.

Все, за исключением действий с ограниченным доступом: в этой роли разрешено выполнять действия по созданию, чтению, обновлению и удалению объектов, кроме переменных и конечных точек с ограниченным доступом.

Таблица 2-1. Разрешения для доступа на уровне служб и проектов в Code Stream

Уровни доступа	Роли Code Stream				
	Администратор Code Stream	Разработчик Code Stream	Исполнитель Code Stream	Обозреватель Code Stream	Пользователь Code Stream
Доступ на уровне службы Code Stream	Все действия	Все действия, кроме действий с ограниченным доступом	Действия при выполнении	Только для чтения	Нет
Доступ на уровне проекта: администратор проекта	Все действия	Все действия	Все действия	Все действия	Все действия
Доступ на уровне проекта: участник проекта	Все действия	Все действия, кроме действий с ограниченным доступом	Все действия, кроме действий с ограниченным доступом	Все действия, кроме действий с ограниченным доступом	Все действия, кроме действий с ограниченным доступом
Доступ на уровне проекта: обозреватель проекта	Все действия	Все действия, кроме действий с ограниченным доступом	Действия при выполнении	Только для чтения	Только для чтения

Пользователи с ролью администратора проекта могут выполнять все действия над проектами, в которых они являются администраторами.

Администратор проекта может создавать, читать, обновлять и удалять конвейеры, переменные, конечные точки, панели управления, триггеры и запускать конвейер, который включает в себя конечные точки или переменные с ограниченным доступом, если эти ресурсы принадлежат проекту, администратором которого является пользователь.

Пользователи, у которых есть роль «Обозреватель службы», могут видеть всю информацию, доступную администратору. Они не могут выполнять никаких действий, пока администратор не сделает их администраторами или участниками проекта. Если пользователь связан с проектом, у него есть разрешения, связанные с ролью. Обозреватель проекта не может расширять свои разрешения так же, как администратор или участник. Эта роль доступна только для чтения во всех проектах.

Если у вас есть разрешения на чтение в проекте, вы все равно можете видеть ресурсы с ограниченным доступом.

- Чтобы просмотреть конечные точки с ограниченным доступом, на карточке которых отображается значок блокировки, щелкните **Настройка > Конечные точки**.
- Чтобы просмотреть секретные переменные и переменные с ограниченным доступом, для которых в столбце **Тип** указаны значения RESTRICTED или SECRET, щелкните **Настройка > Переменные**.

Таблица 2-2. Возможности роли службы Code Stream

Контекст пользовательского интерфейса	Возможности	Роль администратора Code Stream	Роль разработчика Code Stream	Роль исполнителя Code Stream	Роль обозревателя Code Stream	Роль пользователя Code Stream
Конвейеры						
	Просмотр конвейеров	Да	Да	Да	Да	
	Создание конвейеров	Да	Да			
	Запуск конвейеров	Да	Да	Да		
	Запуск работы конвейеров, содержащих конечные точки и переменные с ограниченным доступом	Да				
	Обновление конвейеров	Да	Да			
	Удаление конвейеров	Да	Да			
Циклы выполнения конвейера						
	Просмотр циклов выполнения конвейера	Да	Да	Да	Да	
	Возобновление, приостановка и отмена циклов выполнения конвейера	Да	Да	Да		
	Возобновление работы конвейеров, приостановленных для утверждения использования ресурсов с ограниченным доступом	Да				
Настраиваемые интеграции						

Таблица 2-2. Возможности роли службы Code Stream (продолжение)

Контекст пользовательского интерфейса	Возможности	Роль администратора Code Stream	Роль разработчика Code Stream	Роль исполнителя Code Stream	Роль обозревателя Code Stream	Роль пользователя Code Stream
	Создание настраиваемых интеграций	Да	Да			
	Чтение настраиваемых интеграций	Да	Да	Да	Да	
	Обновление настраиваемых интеграций	Да	Да			
Конечные точки						
	Просмотр циклов выполнения	Да	Да	Да	Да	
	Создание циклов выполнения	Да	Да			
	Обновление циклов выполнения	Да	Да			
	Удаление циклов выполнения	Да	Да			
Пометить как ресурсы с ограниченным доступом						
	Пометка конечной точки или переменной как ресурса с ограниченным доступом	Да				
Панели управления						
	Просмотр панелей управления	Да	Да	Да	Да	
	Создание панелей управления	Да	Да			

Таблица 2-2. Возможности роли службы Code Stream (продолжение)

Контекст пользовательского интерфейса	Возможности	Роль администратора Code Stream	Роль разработчика Code Stream	Роль исполнителя Code Stream	Роль обозревателя Code Stream	Роль пользователя Code Stream
	Обновление панелей управления	Да	Да			
	Удаление панелей управления	Да	Да			

Настраиваемые роли и разрешения в Code Stream

Для пользователей, работающих с конвейерами, можно создавать настраиваемые роли в Cloud Assembly, расширяющие их права. При создании настраиваемой роли для конвейеров Code Stream необходимо выбрать одно или несколько разрешений для **конвейера**.

Для пользователей, которым будет назначена эта настраиваемая роль, выберите минимальное количество разрешений для **конвейера**.

Если пользователю, которому назначен проект и задана роль в этом проекте, назначается настраиваемая роль, которая включает в себя одно или несколько разрешений для **конвейера**, он может выполнять все действия, предусмотренные этими разрешениями. Например, он может создавать переменные с ограниченным доступом, управлять конвейерами с ограниченным доступом, создавать настраиваемые интеграции и управлять ими, а также многое другое.

Таблица 2-3. Разрешения для конвейера, которые можно назначать настраиваемым ролям

Разрешение для конвейера	Администратор Code Stream	Разработчик Code Stream	Исполнитель Code Stream	Обозреватель Code Stream	Пользователь Code Stream	Администратор проекта	Участник проекта	Наблюдатель проекта
Управление конвейерами	Да	Да				Да	Да	
Управление конвейерами с ограниченным доступом	Да					Да		
Управление настраиваемыми интеграциями	Да	Да						
Выполнение конвейеров	Да	Да	Да			Да	Да	
Выполнение конвейеров с ограниченным доступом	Да					Да		

Таблица 2-3. Разрешения для конвейера, которые можно назначать настраиваемым ролям (продолжение)

Разрешение для конвейера	Администратор Code Stream	Разработчик Code Stream	Исполнитель Code Stream	Обозреватель Code Stream	Пользователь Code Stream	Администратор проекта	Участник проекта	Наблюдатель проекта
Управление циклами выполнения	Да					Да		
Чтение. Это разрешение не отображается.	Да	Да	Да	Да		Да	Да	Да

Таблица 2-4. Как можно использовать разрешения для конвейера с настраиваемыми ролями

Разрешение	Возможные действия
Управление конвейерами	<ul style="list-style-type: none"> ■ Создание, обновление, удаление, клонирование конвейеров. ■ Выпуск конвейеров для VMware Service Broker и отмена выпуска. ■ Создание, обновление и удаление конечных точек. ■ Создание, обновление и удаление обычных и секретных переменных. ■ Создание, клонирование, обновление и удаление прослушивателя Gerrit. ■ Подключение и отключение прослушивателя Gerrit. ■ Создание, клонирование, обновление и удаление триггера Gerrit. ■ Создание, обновление и удаление веб-перехватчика Git. ■ Создание, обновление и удаление веб-перехватчика Docker. ■ Использование смарт-шаблонов конвейера для создания конвейеров. ■ Импорт конвейеров из YAML и экспорт их в YAML. ■ Создание, обновление или удаление настраиваемых панелей управления. ■ Чтение всех настраиваемых интеграций ■ Чтение всех конечных точек и переменных с ограниченным доступом; просмотр их значений не разрешен.
Управление конвейерами с ограниченным доступом	<ul style="list-style-type: none"> ■ Создание, обновление и удаление конечных точек. ■ Маркировка конечных точек как точек с ограниченным доступом, их обновление и удаление. ■ Создание, обновление и удаление обычных и секретных переменных. ■ Создание, обновление и удаление переменных с ограниченным доступом. ■ Все разрешения, которые можно использовать при управлении конвейерами.
Управление настраиваемыми интеграциями	<ul style="list-style-type: none"> ■ Создание и обновление настраиваемых интеграций. ■ Создание версии и выпуск настраиваемых интеграций. ■ Удаление и объявление устаревшими версий настраиваемой интеграции. ■ Удаление настраиваемых интеграций.
Выполнение конвейеров	<ul style="list-style-type: none"> ■ Запустите конвейеры. ■ Приостановка, возобновление и отмена выполнения конвейера. ■ Перезапуск цикла выполнения конвейера. ■ Возобновление, перезапуск и запуск вручную триггерного события Gerrit. ■ Утверждение операции пользователя, возможность пакетного утверждения операций пользователей.

Таблица 2-4. Как можно использовать разрешения для конвейера с настраиваемыми ролями (продолжение)

Разрешение	Возможные действия
Выполнение конвейеров с ограниченным доступом	<ul style="list-style-type: none"> ■ Запустите конвейеры. ■ Приостановка, возобновление, отмена и удаление циклов выполнения конвейера. ■ Перезапуск цикла выполнения конвейера. ■ Синхронизация активного цикла выполнения конвейера. ■ Принудительное удаление активного цикла выполнения конвейера. ■ Возобновление, перезапуск, удаление и запуск вручную триггерного события Gerrit. ■ Разрешение запрещенных элементов и продолжение выполнения конвейера. ■ Переключение контекста пользователя и продолжение выполнения конвейера после утверждения задачи «Операция пользователя». ■ Все разрешения, которые можно использовать при выполнении конвейеров.
Управление циклами выполнения	<ul style="list-style-type: none"> ■ Запустите конвейеры. ■ Приостановка, возобновление, отмена и удаление циклов выполнения конвейера. ■ Перезапуск цикла выполнения конвейера. ■ Возобновление, перезапуск, удаление и запуск вручную триггерного события Gerrit. ■ Все разрешения, которые можно использовать при выполнении конвейеров.

Настраиваемые роли могут включать в себя комбинации разрешений. Такие разрешения организуются в виде групп возможностей, которые позволяют пользователям управлять конвейерами, содержащими или не содержащими ресурсы с ограниченным доступом, или выполнять их. Эти разрешения представляют все возможности, которые может выполнять каждая роль в Code Stream.

Например, если создать настраиваемую роль и добавить разрешение с именем **Управление конвейерами с ограниченным доступом**, пользователи с ролью разработчика Code Stream смогут выполнять следующие действия.

- Создание, обновление и удаление конечных точек.
- Маркировка конечных точек как точек с ограниченным доступом, их обновление и удаление.
- Создание, обновление и удаление обычных и секретных переменных.
- Создание, обновление и удаление переменных с ограниченным доступом.

Таблица 2-5. Примеры комбинаций разрешений для конвейера в настраиваемых ролях

Количество разрешений, назначенных настраиваемой роли	Примеры комбинированных разрешений	Использование этой комбинации
Одно разрешение	Выполнение конвейеров	
Два разрешения	Управление конвейерами и Выполнение конвейеров	
Три разрешения	Управление конвейерами, Выполнение конвейеров и Выполнение конвейеров с ограниченным доступом	

Таблица 2-5. Примеры комбинаций разрешений для конвейера в настраиваемых ролях (продолжение)

Количество разрешений, назначенных настраиваемой роли	Примеры комбинированных разрешений	Использование этой комбинации
	Управление конвейерами, Управление настраиваемыми интеграциями и Выполнение конвейеров с ограниченным доступом	Эта комбинация может применяться к роли «Разработчик Code Stream», но только в проектах, в которых пользователь является участником.
	Управление конвейерами, Управление настраиваемыми интеграциями и Управление циклами выполнения	Это сочетание может применяться к роли «Администратор Code Stream», но только в проектах, в которых пользователь является участником.
	Управление конвейерами, Управление конвейерами с ограниченным доступом и Управление настраиваемыми интеграциями	Эта комбинация предоставляет пользователю все разрешения, а также позволяет создавать и удалять любые элементы в Code Stream.

Пользователям с ролью администратора

Администратор может создавать настраиваемые интеграции, конечные точки, переменные, триггеры, конвейеры и панели управления.

Проекты позволяют конвейерам получать доступ к ресурсам инфраструктуры. Администраторы создают проекты, чтобы пользователи могли объединять конвейеры, конечные точки и панели управления.

Затем пользователи выбирают проект в конвейерах. В каждом проекте присутствуют администратор и пользователи с назначенными им ролями.

Пользователь с ролью администратора может ограничивать доступ к конечным точкам и переменным, а также запускать конвейеры, в которых используются ресурсы с ограниченным доступом. Если пользователь, не являющийся администратором, запускает конвейер, содержащий конечную точку или переменную с ограниченным доступом, выполнение остановится на задаче, в которой используется такая переменная, после чего работу конвейера должен будет возобновить администратор.

Администратор может также отправлять запрос на публикацию конвейеров в vRealize Automation Service Broker.

Пользователям с ролью разработчика

Разработчики могут работать с конвейерами так же, как и администраторы, но они не могут работать с конечными точками и переменными, доступ к которым ограничен.

При запуске конвейера, в котором используются конечные точки или переменные с ограниченным доступом, он выполняется только до задачи, в которой используется ресурс с ограниченным доступом. После этого он останавливается, и администратор Code Stream или администратор проекта должен возобновить конвейер.

Роль пользователя

Можно пользоваться Code Stream, но без прав, доступных в рамках других ролей.

Роль обозревателя

Можно просматривать те же ресурсы, которые видит администратор, например конвейеры, конечные точки, циклы выполнения конвейера, панели управления, настраиваемые интеграции и триггеры, но их нельзя создавать, обновлять или удалять. Для выполнения действий обозревателю должна быть также предоставлена роль администратора проекта или участника проекта.

Пользователи с ролью обозревателя могут просматривать проекты. Они также могут просматривать конечные точки и переменные с ограниченным доступом, но подробная информация о них остается недоступной.

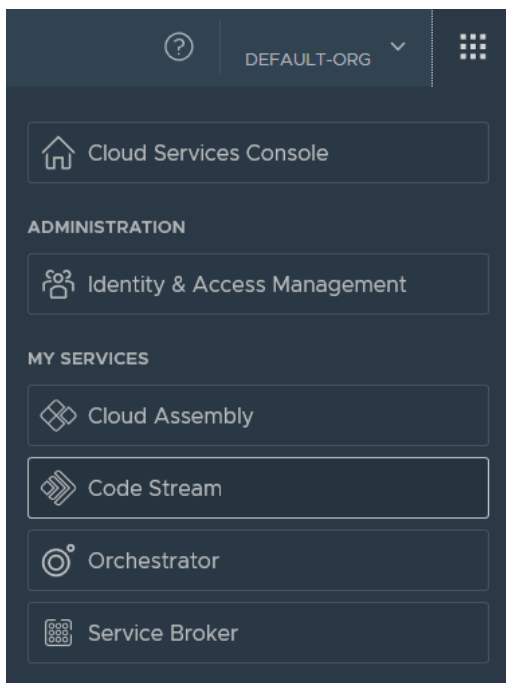
Роль исполнителя

Можно запускать конвейеры и выполнять действия в рамках задач, связанных с пользовательскими операциями. Также можно возобновлять, приостанавливать и отменять выполнение конвейеров, но нельзя изменять конвейеры.

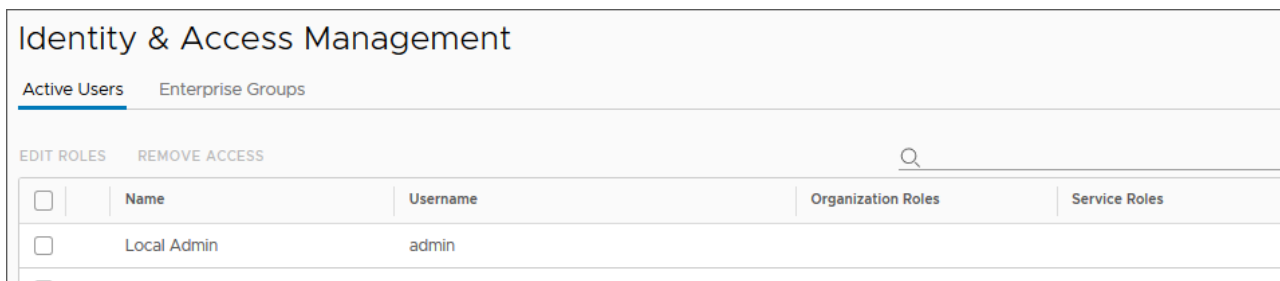
Назначение и обновление ролей

Чтобы назначать роли другим пользователям и обновлять их, требуются права администратора.

1. Для просмотра активных пользователей и их ролей в vRealize Automation нажмите расположенный справа вверху элемент с девятью точками.
2. Нажмите **Управление идентификацией и доступом**.



3. Чтобы просмотреть имена и роли пользователей, нажмите **Активные пользователи**.



4. Чтобы добавить или изменить роли пользователя, установите флажок рядом с его именем и щелкните **Изменить роли**.
5. При добавлении или изменении ролей пользователей можно также добавлять доступ к службам.
6. Чтобы сохранить изменения, нажмите **Сохранить**.

Что такое пользовательские операции и утверждения в Code Stream

В области «Пользовательские операции» отображаются запуски конвейеров, требующие утверждения. Назначенный утверждающий может либо одобрить, либо отклонить запуск конвейера.

При создании конвейера, возможно, понадобится добавить утверждение в конвейер в следующих случаях.

- Участнику рабочей группы нужно проверить код.
- Другому пользователю необходимо подтвердить артефакт сборки.
- Необходимо убедиться, что все тесты завершены.
- В задаче используется ресурс с отметкой администратора об ограниченном доступе, а для задачи требуется подтверждение.
- Конвейер выполняет публикацию программного обеспечения в производственной среде.

Чтобы определить, следует ли утвердить задачу конвейера, назначенный утверждающий должен иметь разрешение и соответствующий опыт.

При добавлении задачи пользовательской операции можно задать время ожидания в днях, часах или минутах. Например, пользователь должен одобрить конвейер в течение 30 минут. Если он этого не делает, работа конвейера завершается с ошибкой.

Если включена функция отправки сообщений электронной почты, задача «Пользовательская операция» отправляет уведомления только утверждающим с полными адресами электронной почты (если это имя, не соответствующее формату электронной почты, сообщение не отправляется).

После подтверждения задачи происходит следующее.

- Выполнение ожидающего конвейера может быть продолжено.
- При возобновлении работы конвейера будут отменены все ранее созданные ожидающие запросы подтверждения в этой задаче.

User Operations GUIDED SETUP

Active Items Inactive Items

✓ APPROVE × REJECT

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

1 Items per page 20 1 - 7 of 7 items

В области «Пользовательские операции» элементы, которые необходимо утвердить или отклонить, отображаются как активные или неактивные элементы. Каждый элемент сопоставляется с задачей пользовательской операции, выполняемой в конвейере.

- **Активные элементы** ожидают проверки задачи утверждающим для ее одобрения или отклонения. Если пользователь указан в списке утверждающих, он может развернуть строку пользовательской операции и нажать кнопку **Принять** или **Отклонить**.
- **Неактивные элементы** уже были подтверждены или отклонены. Если пользовательская операция отклонена пользователем или время ожидания подтверждения задачи истекло, ее уже нельзя будет подтвердить.

Номер индекса — это уникальная строка из шести буквенно-цифровых символов, которую можно использовать в качестве фильтра для поиска конкретного подтверждения.

Подтверждения конвейера также отображаются в области **Выполняемые элементы**.

- Конвейеры, ожидающие утверждения, имеют состояние «ожидание».
- Другие состояния включают в себя «поставленные в очередь», «выполненные» и «ошибка».
- Если конвейер находится в состоянии ожидания, назначенный утверждающий должен подтвердить задачу конвейера.

Создание и использование конвейеров в Code Stream

3

vRealize Automation Code Stream можно использовать для моделирования процесса сборки, тестирования и развертывания. Служба vRealize Automation Code Stream позволяет настроить инфраструктуру, поддерживающую цикл выпуска, и создавать конвейеры для моделирования действий по выпуску программного обеспечения. vRealize Automation Code Stream обеспечивает полный цикл выпуска ПО: написание кода, тестирование и развертывание в экземплярах производственной среды.

Каждый конвейер включает в себя этапы и задачи. Этапы представляют собой стадии процесса разработки, а задачи позволяют выполнять действия, обеспечивающие переход программного приложения с одного этапа на другой.

Конвейеры в vRealize Automation Code Stream

Конвейер — это модель непрерывной интеграции и непрерывного предоставления программного обеспечения в процессе его выпуска. Он переносит ПО из репозитория исходного кода в средства тестирования, и затем — в производственную среду. Конвейер состоит из нескольких последовательных этапов, а также из задач, представляющих собой действия в рамках цикла выпуска программного обеспечения. Конвейер обеспечивает переход программного приложения с одного этапа на другой.

Добавляя конечные точки, можно подключать задачи конвейера к источникам данных, репозиториям и системам уведомлений.

Создание конвейеров

Можно создать конвейер с пустого холста, используя смарт-шаблон конвейера или импортировав код YAML.

- Использование пустого холста. См. пример в разделе [Планирование собственной сборки по модели непрерывной интеграции и предоставления в Code Stream](#) перед добавлением задач вручную.
- Использование смарт-шаблона конвейера. См. пример в разделе [Глава 4 Планирование сборки, интеграции и предоставления кода встроенными средствами в Code Stream](#).
- Импорт кода YAML. Щелкните **Конвейеры > Импорт**. В диалоговом окне **Импорт** выберите файл YAML или введите код YAML и нажмите кнопку **Импортировать**.

При создании конвейера с помощью пустого холста требуется добавить этапы, задачи и подтверждения. Конвейер автоматизирует процесс сборки, тестирования, развертывания и выпуска приложения. Задачи, выполняемые в рамках каждого этапа, запускают действия, с помощью которых обеспечивается сборка, тестирование и выпуск кода в рамках соответствующего этапа.

Таблица 3-1. Примеры этапов конвейера и его использования

Пример этапа	Примеры использования
Разработка	<p>На этапе разработки можно подготовить компьютер, извлечь артефакт, добавить задачу сборки для создания узла Docker, который будет использоваться для непрерывной интеграции кода, и т. д.</p> <p>Например:</p> <ul style="list-style-type: none"> ■ Сведения о том, как спланировать и создать сборку с непрерывной интеграцией (CI), предоставляющую код с помощью встроенной функции сборки vRealize Automation Code Stream, см. в разделе Планирование собственной сборки по модели непрерывной интеграции в Code Stream перед использованием смарт-шаблона конвейера.
Тестирование	<p>На этапе тестирования можно добавить задачу Jenkins, чтобы выполнить тестирование программного приложения, а также включить средства тестирования для последующей обработки, такие как JUnit, JaCoCo и др.</p> <p>Например:</p> <ul style="list-style-type: none"> ■ Интегрируйте vRealize Automation Code Stream с Jenkins и запустите задание Jenkins в конвейере, который собирает и тестирует исходный код. См. раздел Интеграция Jenkins со службой Code Stream. ■ Создайте настраиваемые сценарии, расширяющие возможности vRealize Automation Code Stream для интеграции с вашими собственными инструментами сборки, тестирования и развертывания. См. раздел Интеграция собственных средств сборки, тестирования и развертывания со службой Code Stream. ■ Отслеживайте тенденции в последующей обработке конвейера непрерывной интеграции (CI). См. раздел Отслеживание ключевых показателей эффективности конвейера в Code Stream с помощью настраиваемых панелей управления.
Производственная среда	<p>На этапе переноса в производственную среду можно интегрировать облачный шаблон в Cloud Assembly для подготовки инфраструктуры, развертывания программного обеспечения в кластере Kubernetes и выполнения других задач.</p> <p>Например:</p> <ul style="list-style-type: none"> ■ Примеры этапов для разработки и переноса в производственную среду, с помощью которых можно развернуть программное приложение по модели Blue-Green, см. в разделе Как развернуть приложение Code Stream в среде Blue-Green. ■ Сведения об интеграции облачного шаблона в конвейер см. в разделе Как автоматизировать выпуск приложения, развернутого с использованием облачного шаблона YAML в Code Stream. Кроме того, можно добавить задачу по развертыванию для запуска сценария, выполняющего развертывание приложения. ■ Сведения о том, как автоматизировать развертывание программных приложений в кластере Kubernetes, см. в разделе Автоматизация выпуска приложения в кластере Kubernetes с помощью Code Stream. ■ Сведения о том, как интегрировать код в конвейер и развернуть образ сборки, см. в разделе Настройка непрерывной интеграции кода из репозитория GitHub или GitLab в конвейер Code Stream.

Конвейер можно экспортировать как файл YAML. Щелкните **Конвейеры**, выберите карточку конвейера, затем щелкните **Действия > Экспорт**.

Утверждение конвейеров

В определенных точках конвейера можно получить подтверждение от другого участника рабочей группы.

- Сведения о том, как настроить обязательное подтверждение конвейера путем включения в конвейер задачи «Пользовательская операция», см. в разделе [Запуск конвейера и просмотр результатов](#). Эта задача отправляет уведомление по электронной почте пользователю, который должен выполнить проверку. Прежде чем работа конвейера будет продолжена, проверяющий должен утвердить либо отклонить действие. Если срок действия задачи «Пользовательская операция» задан в днях, часах или минутах, пользователь должен одобрить конвейер до его истечения. В противном случае работа контейнера ожидаемо завершается сбоем.
- Службу vRealize Automation Code Stream можно настроить так, чтобы в случае сбоя задачи или этапа на любом этапе конвейера в ней создавался запрос Jira. См. раздел [Создание запроса Jira в Code Stream при неудачном выполнении задачи конвейера](#).

Запуск конвейеров

Конвейеры могут запускаться, когда разработчики вносят код в репозиторий или проверяют код, а также когда выявляется новый или обновленный артефакт сборки.

- Чтобы интегрировать vRealize Automation Code Stream с жизненным циклом Git и настроить запуск конвейера при обновлении кода разработчиками, используйте триггер Git. См. раздел [Как использовать триггер Git в Code Stream для запуска конвейера](#).
- Чтобы интегрировать vRealize Automation Code Stream с жизненным циклом проверки кода Gerrit и настроить запуск конвейера при проверках кода, используйте триггер Gerrit. См. раздел [Как использовать триггер Gerrit в Code Stream для запуска конвейера](#).
- Чтобы запустить конвейер при создании или обновлении артефакта сборки Docker, используйте триггер Docker. См. раздел [Как использовать триггер Docker в Code Stream для запуска конвейера непрерывного предоставления](#).

Дополнительные сведения о триггерах, поддерживаемых службой vRealize Automation Code Stream, см. в разделе [Глава 7 Запуск конвейеров в Code Stream](#).

В эту главу входят следующие разделы:

- [Запуск конвейера и просмотр результатов](#)
- [Типы задач, доступные в Code Stream](#)
- [Как использовать привязки переменных в конвейерах Code Stream](#)
- [Использование привязок переменных в задаче «Условие» для запуска или остановки конвейера в Code Stream](#)
- [Какие переменные и выражения можно использовать для связывания задач конвейера в Code Stream](#)
- [Отправка уведомлений о работе конвейера в службу Code Stream](#)
- [Создание запроса Jira в Code Stream при неудачном выполнении задачи конвейера](#)

- [Как откатить развертывание в Code Stream](#)

Запуск конвейера и просмотр результатов

Запустить конвейер можно из карточки конвейера, в режиме изменения конвейера и из цикла выполнения конвейера. Кроме того, можно использовать доступные триггеры для запуска конвейера службой Code Stream при возникновении определенных событий.

Если все этапы и задачи в конвейере являются допустимыми, конвейер готов к публикации и запуску напрямую или посредством триггера.

Чтобы запустить конвейер с помощью Code Stream, его можно включить и запустить либо в карточке конвейера, либо в самом конвейере. Затем можно просмотреть выполнение конвейера и убедиться, что он собрал, протестировал и развернул код.

В ходе выполнения конвейера можно удалить выполняемый элемент, если вы являетесь администратором или обычным пользователем.

- Администратор. Чтобы удалить текущее выполнение конвейера, щелкните **Выполняемые элементы**. Выберите выполняемый элемент, который нужно удалить, и щелкните **Действия > Удалить**.
- Обычный пользователь (без прав администратора). Чтобы удалить текущее выполнение конвейера, щелкните **Выполняемые элементы**, а затем нажмите **Alt Shift d**.

Если выполнение конвейера предположительно заблокировано, администратор может обновить выполнение на странице «Выполняемые элементы» или «Сведения о выполнении».

- Страница «Выполняемые элементы». Щелкните **Выполняемые элементы**. Выберите выполняемый элемент, который необходимо обновить, а затем щелкните **Действия > Синхронизация**.
- Страница «Сведения о выполнении». Щелкните **Выполняемые элементы**, затем ссылку на сведения о выполнении и выберите **Действия > Синхронизация**.

Для запуска конвейера при возникновении определенных событий используйте триггеры.

- Триггер Git может запускать конвейер, когда разработчики обновляют код.
- Триггер Gerrit может запускать конвейер при проверке кода.
- Триггер Docker может запускать конвейер при создании артефакта в реестре Docker.
- Использование команды curl или wget позволяет Jenkins запускать конвейер по окончании сборки Jenkins.

Дополнительные сведения об использовании триггеров см. в разделе [Глава 7 Запуск конвейеров в Code Stream](#).

В следующей процедуре показано, как запускать конвейер из его карточки, просматривать выполняемые элементы, сведения о выполнении и использовать действия. Здесь также показано, как опубликовать конвейер, чтобы его можно было добавить в vRealize Automation Service Broker.

Необходимые условия

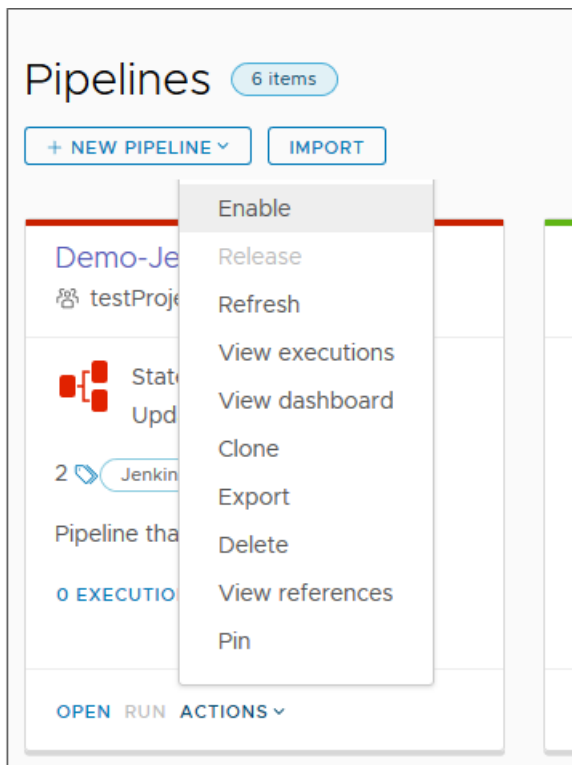
- Убедитесь в наличии нескольких конвейеров. Примеры см. в разделе [Глава 5 Учебники по использованию Code Stream](#).

Процедура

1. Включите конвейер.

Перед запуском или публикацией конвейере его необходимо включить.

- а) Нажмите **Конвейеры**.
- б) В карточке конвейера выберите пункт **Действия > Включить**.



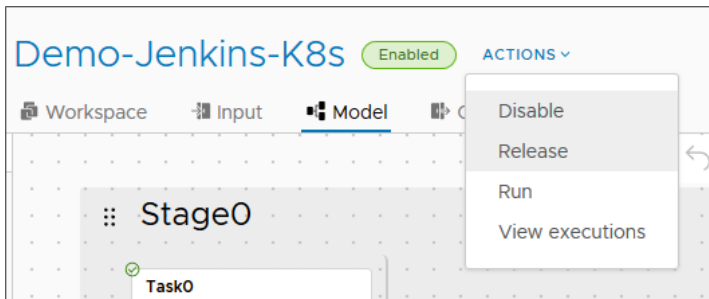
Включить конвейер также можно из самого конвейера. Если конвейер уже включен, кнопка **Выполнить** активна, а в меню **Действия** отображается опция **Отключить**.

2. (дополнительно) Опубликуйте конвейер.

Если требуется сделать конвейер доступным в качестве элемента каталога в vRealize Automation Service Broker, необходимо опубликовать его в Code Stream.

- а) Нажмите **Конвейеры**.
- б) В карточке конвейера выберите пункт **Действия > Опубликовать**.

Опубликовать конвейер также можно из самого конвейера.



После публикации конвейера откройте Service Broker, чтобы добавить конвейер в качестве элемента каталога и запустить его. См. раздел [Добавление конвейеров Code Stream в каталог Service Broker](#).

Примечание Если продолжительность работы конвейера составляет более 120 минут, укажите в качестве значения времени ожидания запроса приблизительное время выполнения. Чтобы задать или просмотреть время ожидания запроса для проекта, откройте Service Broker как администратор и выберите **Инфраструктура > Проекты**. Щелкните имя проекта, а затем **Предоставление**.

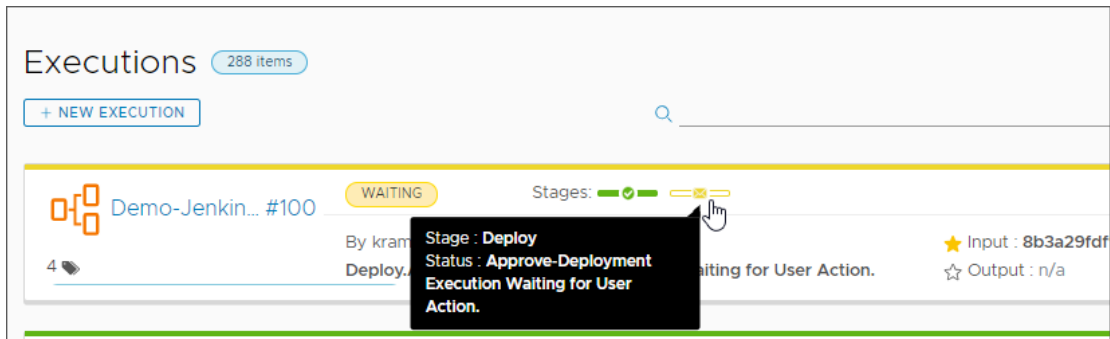
Если значение времени ожидания запроса не задано, выполнение, занимающее более 120 минут, будет отображаться как завершившееся сбоем с ошибкой запроса времени ожидания обратного вызова. Однако на выполнение конвейера это не влияет.

3. В карточке конвейера нажмите **Запустить**.
4. Для просмотра результатов текущего выполнения конвейера нажмите **Выполняемые элементы**.

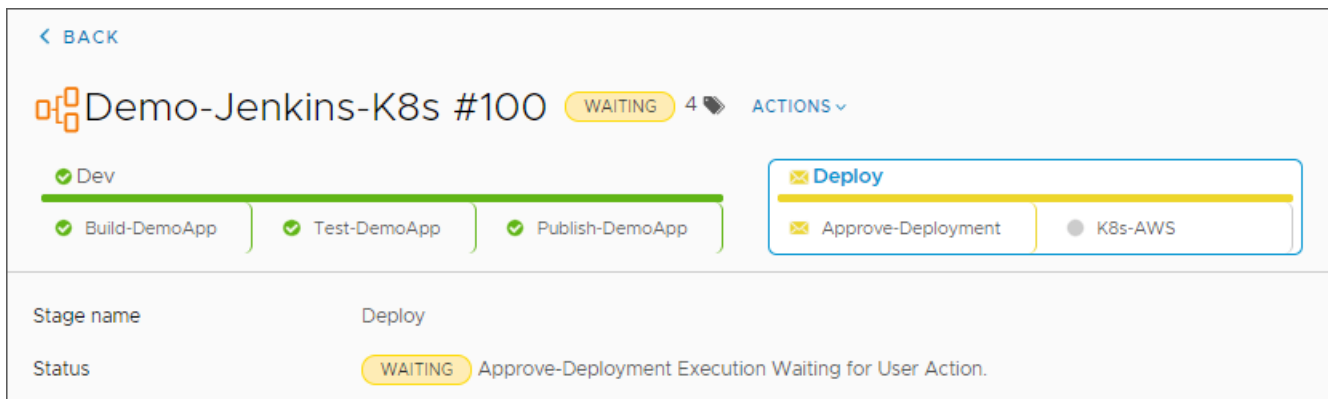
Конвейер последовательно выполняет каждый этап. При выполнении конвейера для каждого этапа отображается значок состояния. Если конвейер включает в себя задачу пользовательской операции, его выполнение может продолжиться только после утверждения этой задачи пользователем. Если используется задача пользовательской операции, конвейер прекращает работу и ожидает утверждения задачи соответствующим пользователем.

Например, задачу пользовательской операции можно использовать для подтверждения развертывания кода в производственной среде.

Если срок действия задачи «Пользовательская операция» задан в днях, часах или минутах, пользователь должен одобрить конвейер до его истечения. В противном случае работа контейнера ожидаемо завершается сбоем.



- Чтобы просмотреть этап конвейера, ожидающего подтверждения пользователем, щелкните значок состояния этапа.

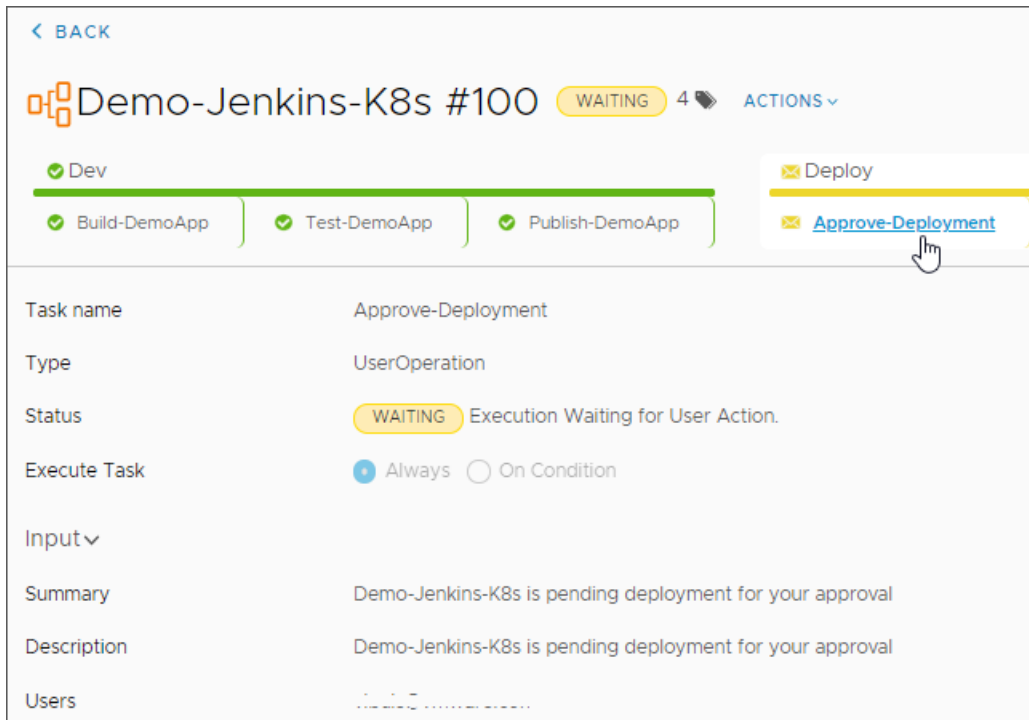


- Щелкните нужную задачу, чтобы просмотреть сведения о ней.

Как только нужный пользователь подтвердит задачу, пользователь с соответствующей ролью должен возобновить выполнение конвейера. Дополнительные сведения о необходимых ролях см. в разделе [Управление пользовательским доступом и утверждениями в службе Code Stream](#).

Если выполнение завершилось сбоем, необходимо выявить и исправить причину сбоя. Затем выберите выполняемый элемент и нажмите **Действия > Запустить повторно**.

Можно возобновить выполнение основного и вложенных циклов выполнения конвейера.



7. В среде выполнения конвейера можно щелкнуть **Действия**, чтобы просмотреть конвейер, и выбрать действие, например **Приостановить**, **Отмена** и т. д. Во время выполнения конвейера выполняемый процесс можно удалить или синхронизировать. Если вы являетесь обычным пользователем, а не администратором, можно удалить запущенный конвейер.
8. Чтобы легко перемещаться между выполняемыми элементами и просматривать сведения о задаче, щелкните **Выполняемые элементы** и выберите цикл выполнения конвейера. Затем перейдите на вкладку сверху и выберите цикл выполнения конвейера.



Результаты

Поздравляем! Конвейер запущен, выполнение конвейера проверено, задача подтверждения для продолжения выполнения конвейера просмотрена. Для возврата к модели конвейера и внесения необходимых изменений использовано меню **Действия** в представлении выполнения конвейера.

Следующие шаги

Дополнительные сведения об использовании Code Stream для автоматизации цикла выпуска программного обеспечения см. в разделе [Глава 5 Учебники по использованию Code Stream](#).

Типы задач, доступные в Code Stream

При настройке конвейера добавляются определенные типы задач, которые конвейер выполняет для требуемых действий. Каждый тип задач интегрируется с другим приложением и активирует конвейер для выполнения сборки, тестирования и предоставления приложений.

В Code Stream доступны все задачи, необходимые для работы конвейера: извлечение артефактов из репозитория для развертывания, запуск удаленного сценария, запрос подтверждения пользовательской операции от участника группы и т. д.

Code Stream поддерживает отмену выполнения конвейера для различных типов задач. Если нажать **Отмена** в ходе выполнения конвейера, задача, этап или весь конвейер перейдут в состояние отмены, и выполнение конвейера будет отменено.

Code Stream позволяет отменить выполнение конвейера для задачи, этапа или всего конвейера при использовании следующих задач.

- Jenkins
- SSH
- PowerShell
- Пользовательская операция
- Конвейер
- Облачный шаблон
- vRO
- POLL

Code Stream не распространяет поведение отмены на сторонние системы для следующих задач: непрерывная интеграция, настраиваемая интеграция и Kubernetes. Code Stream помечает задачу как отмененную и сразу же прекращает извлечение состояния, не ожидая завершения задачи. Задача может быть выполнена или завершиться сбоем в сторонней системе, но в Code Stream ее выполнение останавливается сразу после нажатия кнопки **Отмена**.

Прежде чем использовать задачу в конвейере, убедитесь в доступности соответствующей конечной точки.

Таблица 3-2. Получите утверждение или настройте точку принятия решения

Тип задачи	Функции	Примеры и сведения
Пользовательская операция	Задача «Пользовательская операция» включает обязательное утверждение, которое определяет, в какой момент работы конвейера он должен быть приостановлен для утверждения.	См. раздел Запуск конвейера и просмотр результатов и Управление пользовательским доступом и утверждениями в службе Code Stream .
Условие	Добавляет точку принятия решений, которая определяет, будет ли конвейер продолжать работать или остановится, на основе условных выражений. Если условие истинно, конвейер выполняет последующие задачи. Если условие ложно, конвейер останавливается.	См. раздел Использование привязок переменных в задаче «Условие» для запуска или остановки конвейера в Code Stream.

Таблица 3-3. Автоматизация непрерывной интеграции и развертывания

Тип задачи	Функции	Примеры и сведения
Облачный шаблон	Развертывает облачный шаблон автоматизации из GitHub и подготавливает приложение, а также автоматизирует непрерывную интеграцию и непрерывную доставку (CICD) этого облачного шаблона для развертывания.	<p>См. раздел Как автоматизировать выпуск приложения, развернутого с использованием облачного шаблона YAML в Code Stream.</p> <p>Параметры облачного шаблона отображаются, после того как пользователь выбрал элемент Создать или Обновить, а затем Облачный шаблон и Версия. В текстовые области ввода в задаче облачного шаблона можно добавить следующие элементы, которые содержат привязки переменных.</p> <ul style="list-style-type: none"> ■ Целое число ■ Строка перечисления ■ Логическое значение ■ Переменная массива <p>Если во входных данных используются привязки переменных, необходимо учитывать следующие исключения. Для перечислений значения выбираются из фиксированного набора. Логические значения вводятся в текстовую область ввода.</p> <p>Параметр облачного шаблона отображается в задаче облачного шаблона, если облачный шаблон в Cloud Assembly содержит входные переменные. Например, если в облачном шаблоне есть тип входных данных Integer, можно ввести непосредственно целочисленное значение или переменную с помощью привязки переменных.</p>
CI	<p>Задача CI обеспечивает непрерывную интеграцию кода пользователя в конвейер путем извлечения образа сборки Docker из конечной точки реестра и его развертывания в кластере Kubernetes.</p> <p>Задача CI отображает 100 строк журнала в качестве выходных данных и 500 строк при загрузке журналов.</p> <p>Для задач непрерывной интеграции требуются временные порты 32768–61000.</p>	См. раздел Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream перед использованием смарт-шаблона конвейера .
Настраиваемое	Задача «Настраиваемое» обеспечивает интеграцию Code Stream с пользовательскими инструментами сборки, тестирования и развертывания.	См. раздел Интеграция собственных средств сборки, тестирования и развертывания со службой Code Stream .

Таблица 3-3. Автоматизация непрерывной интеграции и развертывания (продолжение)

Тип задачи	Функции	Примеры и сведения
Kubernetes	Автоматизация развертывания приложений в кластерах Kubernetes на AWS.	См. раздел Автоматизация выпуска приложения в кластере Kubernetes с помощью Code Stream .
Конвейер	<p>Создает вложенный конвейер в главном конвейере. Если конвейер является вложенным, он работает как задача в первичном конвейере.</p> <p>На вкладке «Задача» главного конвейера можно легко перейти к вложенному конвейеру, если щелкнуть ссылку на него. Вложенный конвейер откроется на вкладке «Создать браузер».</p>	Чтобы найти вложенные конвейеры в разделе Выполняемые элементы , введите вложенный в области поиска.

Таблица 3-4. Интеграция приложений для разработки, тестирования и развертывания

Тип задачи...	Функции...	Примеры и сведения...
Bamboo	Взаимодействует с сервером непрерывной интеграции Bamboo (CI), который непрерывно создает, тестирует и интегрирует программное обеспечение при подготовке к развертыванию, а также запускает сборки кода, когда разработчики фиксируют изменения. Он предоставляет расположения артефактов, которые создает сборка Bamboo, чтобы задача могла выводить параметры для других задач, которые будут использоваться для сборки и развертывания.	Подключитесь к конечной точке сервера Bamboo и запустите план сборки Bamboo из своего конвейера.
Jenkins	Запускает задания Jenkins, которые создают и проверяют исходный код, запускают сценарии тестирования и могут использовать настраиваемые сценарии.	См. раздел Интеграция Jenkins со службой Code Stream .
TFS	Позволяет подключить конвейер к Team Foundation Server для управления проектами сборки и их вызова, в том числе настроенных заданий, которые обеспечивают сборку и тестирование кода.	Сведения о версиях Team Foundation Server, которые поддерживает Code Stream, см. в Что такое конечные точки в Code Stream .
vRO	<p>Расширяет возможности Code Stream за счет запуска предварительно определенных или настраиваемых рабочих процессов в vRealize Orchestrator.</p> <p>Code Stream поддерживает базовую проверку подлинности и проверку подлинности на основе маркеров для vRealize Orchestrator. Code Stream использует маркер API-интерфейса для проверки подлинности и проверки кластера vRealize Orchestrator. Благодаря проверке подлинности на основе маркеров Code Stream поддерживает конечные точки vRealize Orchestrator, которые используют облачный прокси-сервер, обеспечивающий расширяемость. В результате в Code Stream можно запускать рабочие процессы с использованием конечной точки vRealize Orchestrator, которая применяет облачный прокси-сервер, обеспечивающий расширяемость.</p>	См. раздел Интеграция службы Code Stream с решением vRealize Orchestrator .

Таблица 3-5. Интеграция других приложений с помощью API-интерфейса

Тип задачи...	Функции...	Примеры и сведения...
REST	Интегрирует Code Stream с другими приложениями, использующими интерфейс REST API, чтобы было можно непрерывно разрабатывать и доставлять приложения, которые должны взаимодействовать друг с другом.	См. раздел Как использовать интерфейс REST API для интеграции Code Stream с другими приложениями .
Опрос	<p>Вызывает интерфейс REST API и опрашивает его до тех пор, пока задача конвейера не станет соответствовать критериям выхода и не завершится.</p> <p>Максимальное количество опросов, которое может указать администратор Code Stream, составляет 10 000. Интервал между опросами должен быть не менее 60 секунд.</p> <p>Если установить флажок Продолжать при сбое, то в случае превышения указанного значения количества или интервала будет продолжено выполнение задачи опроса.</p> <p>POLL Iteration Count: возникает в ходе выполнения конвейера и обозначает, сколько раз задача опроса (POLL) запрашивала ответ с URL-адреса. Например, если сходное значение POLL равно 65, а фактическое количество выполнений запроса POLL составляет 4, то в выходных данных выполнения конвейера будет указано 4 итерации (из 65).</p>	См. раздел Как использовать интерфейс REST API для интеграции Code Stream с другими приложениями .

Таблица 3-6. Запуск удаленных и пользовательских сценариев

Тип задачи	Функции	Примеры и сведения
PowerShell	<p>Задача PowerShell позволяет Code Stream выполнять команды сценария на удаленном узле. Например, сценарий может автоматизировать тестовые задачи и выполнять административные команды.</p> <p>Сценарий может быть удаленным или определяемым пользователем. Он может подключаться по протоколу HTTP или HTTPS, а также использовать протокол TLS.</p> <p>Для узла Windows необходимо настроить службу winrm. Для winrm, в свою очередь, необходимо настроить параметры MaxShellsPerUser и MaxMemoryPerShellMB.</p> <p>Чтобы запустить оболочку PowerShell, должно быть открыто подключение к удаленному узлу Windows.</p> <p>Длина командной строки PowerShell</p> <p>При вводе команды PowerShell в формате base64 сначала следует вычислить ее общую длину.</p> <p>В конвейере Code Stream команда PowerShell формата base64 кодируется и упаковывается в другую команду, что увеличивает ее общую длину.</p> <p>Максимально допустимая длина команды winrm PowerShell составляет 8192 байт. С учетом кодирования и упаковки предельно допустимая длина команды оболочки PowerShell уменьшается. Поэтому прежде чем ввести команду PowerShell, необходимо вычислить ее длину.</p> <p>Ограничение длины команды для оболочки PowerShell Code Stream зависит от длины исходной команды в коде base64. Длина команды вычисляется следующим образом.</p> $3 * (\text{length of original command} / 4) - (\text{numberOfPaddingCharacters}) + 77 (\text{Length of Write-output command})$ <p>Длина команды Code Stream должна быть меньше максимального порога в 8192 байт.</p>	<p>При настройке MaxShellsPerUser и MaxMemoryPerShellMB:</p> <ul style="list-style-type: none"> ■ Допустимое значение для MaxShellsPerUser — 500 для 50 параллельных конвейеров с 5 задачами PowerShell для каждого конвейера. Чтобы задать значение, выполните: winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}' ■ Допустимое значение памяти для MaxMemoryPerShellMB — 2048. Чтобы задать значение, выполните: winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}' <p>Сценарий записывает выходные данные в файл ответа, который может использоваться другим конвейером.</p>
SSH	<p>Задача SSH позволяет задаче сценария оболочки Bash выполнять команды сценария на удаленном узле. Например, сценарий может автоматизировать тестовые задачи и выполнять административные команды.</p> <p>Сценарий может быть удаленным или определяемым пользователем. Он может подключаться по протоколу HTTP или HTTPS и требует наличия закрытого ключа или пароля.</p> <p>Служба SSH должна быть настроена на узле Linux, а для конфигурации SSHD для MaxSessions должно быть задано значение 50.</p>	<p>Сценарий может быть удаленным или определяемым пользователем. Например, сценарий может выглядеть следующим образом:</p> <pre>message="Hello World" echo \$message</pre> <p>Сценарий записывает выходные данные в файл ответа, который может использоваться другим конвейером.</p>

Таблица 3-6. Запуск удаленных и пользовательских сценариев (продолжение)

Тип задачи	Функции	Примеры и сведения
	При параллельном выполнении множества задач SSH увеличьте значения параметров <code>MaxSessions</code> и <code>MaxOpenSessions</code> для узла SSH. Не используйте экземпляр vRealize Automation в качестве узла SSH, если необходимо изменить параметры <code>MaxSessions</code> и <code>MaxOpenSessions</code> .	

Как использовать привязки переменных в конвейерах Code Stream

Привязка задачи конвейера — это создание зависимости для задачи при выполнении конвейера. Создать привязку для задачи конвейера можно несколькими способами. Можно привязать задачу к другой задаче, к переменной и выражению или к условию.

Использование привязок со знаком доллара с переменными облачных шаблонов в задаче облачного шаблона

Привязки со знаком доллара можно применить к переменным облачного шаблона в задаче облачного шаблона конвейера Code Stream. Способ изменения переменных в Code Stream зависит от кода, с помощью которого заданы свойства переменной в облачном шаблоне.

Если в задаче облачного шаблона необходимо использовать привязки в виде знака доллара, а текущая версия облачного шаблона, применяемая в задаче, этого не позволяет, измените облачный шаблон в Cloud Assembly и разверните новую версию. Затем используйте новую версию облачного шаблона в задаче облачного шаблона и добавьте привязки со знаком доллара там, где это необходимо.

Чтобы применить привязки в виде знака доллара к типам свойств, предоставляемым облачным шаблоном Cloud Assembly, требуются соответствующие разрешения.

- Необходимо иметь ту же роль, что и пользователь, создавший развертывание облачного шаблона в Cloud Assembly.
- Моделировать и запускать конвейер могут разные пользователи, роли которых могут отличаться.
- Если разработчик с ролью исполнителя в Code Stream моделирует конвейер, у этого разработчика должна быть такая же роль в Cloud Assembly, как у пользователя, развернувшего облачный шаблон. Например, это может быть роль администратора Cloud Assembly.
- Конвейер и развертывание могут быть созданы только лицом, моделирующим конвейер, так как у этого лица есть соответствующее разрешение.

Использование маркера API-интерфейса в задаче облачного шаблона

- Пользователь, моделирующий конвейер, может выдать маркер API-интерфейса другому пользователю с ролью исполнителя в Code Stream. При запуске конвейера исполнителем используются маркер API-интерфейса и учетные данные, создаваемые маркером API-интерфейса.

- Когда пользователь вводит маркер API-интерфейса в задаче облачного шаблона, создаются необходимые для конвейера учетные данные.
- Чтобы зашифровать значение маркера API-интерфейса, щелкните **Создать переменную**.
- Если переменная для маркера API-интерфейса не создана, при использовании в задаче облачного шаблона значение маркера API-интерфейса будет отображаться в виде обычного текста.

Чтобы применить привязки со знаком доллара к переменным облачного шаблона в задаче облачного шаблона, выполните следующие действия.

Начните с облачного шаблона, для которого определены свойства входных переменных, например `integerVar`, `stringVar`, `flavorVar`, `BooleanVar`, `objectVar` и `arrayVar`. Определения свойств образа можно найти в разделе `resources`. Свойства в коде облачного шаблона могут выглядеть следующим образом.

```
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}'
```

Для `image` и `flavor` можно использовать переменные со знаком доллара (\$). Например:

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

Чтобы использовать облачный шаблон в конвейере Code Stream и добавить к нему привязки в виде знака доллара, выполните следующие действия.

1. В Code Stream выберите **Конвейеры > Пустой холст**.
2. Добавьте в конвейер задачу **Облачный шаблон**.
3. В задаче «Облачный шаблон» в качестве **источника облачного шаблона** выберите **Облачные шаблоны Cloud Assembly**, введите имя облачного шаблона и выберите его версию.
4. Обратите внимание, что можно ввести маркер API-интерфейса в качестве источника учетных данных для конвейера. Чтобы создать переменную, которая шифрует маркер API-интерфейса в задаче облачного шаблона, щелкните **Создать переменную**.
5. Обратите внимание на значения параметров в таблице **Параметры и значения**. Значение по умолчанию для параметра `flavor` — `small`, для параметра `image` — `ubuntu`.
6. Предположим, требуется изменить облачный шаблон в Cloud Assembly. Для этого можно выполнить следующие действия.
 - а) Настройте конфигурацию так, чтобы в ней использовалось свойство типа массив. В Cloud Assembly разрешается указывать значения для конфигурации через запятую, если выбран тип **array**.
 - б) Щелкните **Развернуть**.
 - в) На странице «Типы развертывания» введите имя развертывания и выберите версию облачного шаблона.
 - г) На странице «Входные данные развертывания» можно определить одно или несколько значений для `Flavor`.
 - д) Следует отметить, что входные данные развертывания включают в себя все переменные, определенные в коде облачного шаблона, и отображаются так, как они заданы в коде. Например: `Integer Var`, `String Var`, `Flavor Var`, `Boolean Var`, `Object Var` и `Array Var`. `String Var` и `Flavor Var` являются строковыми значениями, а `Boolean Var` — флажком.
 - е) Щелкните **Развернуть**.
7. В Code Stream выберите новую версию облачного шаблона и введите значения в таблице **Параметры и значения**. Облачные шаблоны поддерживают следующие типы параметров, которые позволяют создавать привязки Code Stream, используя переменные в виде знака доллара. Имеются

незначительные различия между пользовательским интерфейсом задачи облачного шаблона Code Stream и пользовательским интерфейсом облачного шаблона Cloud Assembly. В зависимости от кодировки облачного шаблона в Cloud Assembly ввод значений в задаче облачного шаблона в Code Stream может быть запрещен.

- а) Для **flavorVar**: если тип облачного шаблона определен как строка или массив, введите строку или массив значений, разделенных запятыми. Пример массива: **test, test**.
- б) Для **BooleanVar**: в раскрывающемся меню выберите значение **истина** или **ложь**. Или, чтобы использовать привязку переменных, введите знак **\$** и выберите привязку переменных из

The screenshot shows a table with two columns: 'Parameter' and 'Value'. The parameters listed are stringVar (value: raj), integerVar (value: 1), flavorVar (value: medium), BooleanVar (value: \$), objectVar, and arrayVar. A dropdown menu is open for the BooleanVar parameter, showing a list of variables: var, input, comments, requestBy, executionIndex, executionId, and executionUrl. Below the table, there is a section for 'Output Parameter' with a list of output parameters: status, deployItemCreate, deployItemId, and deployItemName.

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

Output Parameter

- status
- deployItemCreate
- deployItemId
- deployItemName

- в) Для **objectVar**: укажите значение, используя фигурные скобки и кавычки, в следующем формате: **{"bkix": "bkix":}**.
- г) Переменная **objectVar** будет передаваться в облачный шаблон, и ее можно будет использовать различными способами в зависимости от облачного шаблона. Это позволяет использовать для объекта JSON строковый формат, а также добавлять пары «ключ-значение», разделенные запятыми, в таблицу «ключ-значение». Для объекта JSON можно ввести обычный текст или пару «ключ-значение» как обычный формат преобразования в строку для JSON.
- д) Для **arrayVar**: введите входное значение в виде массива с разделителями-запятыми в следующем формате: **["1", "2"]**.

8. В конвейере можно привязать входной параметр к массиву.

- а) Перейдите на вкладку **Входные данные**.
- б) Укажите имя входных данных. Например, **arrayInput**.
- в) В таблице **Параметры и значения** выберите **arrayVar** и введите **\${input.arrayInput}**.
- г) После сохранения и включения конвейера, при его запуске будет необходимо указывать входное значение массива. Например, введите **["1", "2"]** и нажмите кнопку **Запустить**.

Мы рассмотрели, как использовать привязку переменных со знаком доллара (\$) в облачном шаблоне в задаче облачного шаблона конвейера Code Stream.

Передача параметра в конвейер при его выполнении

Можно указать дополнительные входные параметры для конвейера, чтобы служба Code Stream передала их в конвейер. После этого при выполнении конвейера пользователь должен будет вводить значение соответствующих входных параметров. Если в конвейер добавлены выходные параметры, в задачах конвейера может использоваться выходное значение из какой-либо задачи. Служба Code Stream поддерживает несколько способов использования параметров для удовлетворения различных требований конвейеров.

Например, чтобы запрашивать у пользователя URL-адрес его сервера Git при выполнении конвейера с задачей REST, можно привязать задачу REST к URL-адресу сервера Git.

Чтобы создать привязку переменных, необходимо добавить переменную привязки URL-адреса в задачу REST. Когда конвейер работает и достигает задачи REST, пользователь должен ввести URL-адрес сервера Git. Чтобы создать привязку, выполните следующие действия.

1. В конвейере перейдите на вкладку **Входные данные**.
2. Чтобы задать параметр, выберите для элемента **Автоматическая вставка параметров** значение **Git**.
Отображается список параметров Git, среди которых есть **GIT_SERVER_URL**. Если необходимо использовать значение по умолчанию для URL-адреса сервера Git, измените этот параметр.
3. Щелкните **Модель** и выберите нужную задачу REST.
4. На вкладке **Задача** в области **URL-адрес** введите знак **\$**, затем выберите **Входные данные** и **GIT_SERVER_URL**.

The screenshot shows the configuration window for a task named 'Task3'. The interface includes tabs for 'Task :Task3', 'Notifications', and 'Rollback', along with a 'VALIDATE TASK' button. The configuration fields are as follows:

- Task name:** Task3
- Type:** REST
- Continue on failure:** ☐
- Execute task:** ☒ Always ☐ On condition
- REST Request:**
 - Action:** GET
 - URL:** \${input.|
 - Agent endpoint:**
 - Headers:**
- Output Parameters:** status, responseHeaders, responseBody, responseJson, responseCode

A dropdown menu is open for the 'URL' field, showing a list of variables: GIT_BRANCH_NAME, GIT_CHANGE_SUBJECT, GIT_COMMIT_ID, GIT_EVENT_DESCRIPTION, GIT_EVENT_OWNER_NAME, GIT_EVENT_TIMESTAMP, GIT_REPO_NAME, and GIT_SERVER_URL. The 'GIT_SERVER_URL' option is highlighted, and a mouse cursor is pointing at it.

Запись выглядит следующим образом: `${input.GIT_SERVER_URL}`

- Чтобы проверить целостность привязки переменной для задачи, щелкните **Проверить задачу**.

Code Stream показывает, что задача успешно проверена.

- Когда конвейер выполняет задачу REST, пользователь должен ввести URL-адрес сервера Git. В противном случае задача не завершится.

Связывание двух задач конвейера путем создания входных и выходных параметров

Чтобы связать две задачи, необходимо добавить переменную привязки во входную конфигурацию принимающей задачи. Затем, когда конвейер работает, пользователь заменяет переменную привязки на обязательные входные данные.

Для связывания задач конвейера следует указать переменную со знаком доллара (\$) во входных и выходных параметрах. Рассмотрим это на примере.

Допустим, нам требуется конвейер для вызова URL-адреса в задаче REST и вывода ответа. Чтобы вызвать URL-адрес и вывести ответ, необходимо включить в задачу REST входные и выходные параметры. Кроме того, необходим пользователь, который может подтвердить задачу, а также в разделе «Пользовательские операции» требуется указать задачу для другого пользователя, который может ее подтвердить при запуске конвейера. В этом примере показано, как использовать выражения во входных и выходных параметрах, а также как настроить в конвейере ожидание подтверждения задачи.

1. В конвейере перейдите на вкладку **Входные данные**.

rest-ix-1 Enabled ACTIONS ▾

Workspace **Input** Model Output

Input Parameters ⓘ

Auto inject parameters ☐ Gerrit ☐ Git ☐ Docker ☒ None

ADD ADD/REMOVE INJECTED PARAMETERS

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

2. В качестве значения **Параметров автоматической вставки** оставьте вариант **Нет**.
3. Нажмите кнопку **Добавить**, введите имя, значение и описание параметра и нажмите кнопку **ОК**.
Например:
 - а) Введите имя URL-адреса.
 - б) Введите значение: {Stage0.Task3.input.http://www.docs.vmware.com}
 - в) Введите описание.
4. Перейдите на вкладку **Выходные данные**, щелкните **Добавить** и введите имя и сопоставление выходного параметра.

Add Pipeline Output Parameter

Name *

Reference \$ *

- а) Введите уникальное имя выходного параметра.
- б) Щелкните область **Ссылка** и введите \$.
- в) Укажите сопоставление выходных данных задач, выбрав нужные параметры. Выберите **Stage0**, **Task3**, **output** и **responseCode**. Затем нажмите **OK**.

rest-ix-1 Enabled ACTIONS ▾

Workspace Input Model **Output**

Output Parameters ⓘ

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

5. Сохраните конвейер.
6. В меню **Действия** выберите **Запустить**.
7. Щелкните **Действия > Просмотреть выполняемые элементы**.
8. Выберите цикл выполнения конвейера и просмотрите заданные входные и выходные параметры.

The screenshot displays the vRealize Automation interface for a Code Stream execution. At the top, the execution is named 'rest-ix-1 #2' and is in a 'WAITING' state, indicated by a yellow pill. Below this, a task flow diagram shows 'Stage0' containing 'Task2' (active) and 'Task3' (inactive). The main details section lists the following information:

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

9. Чтобы утвердить конвейер, щелкните **Пользовательские операции** и просмотрите список подтверждений на вкладке **Активные элементы**. Или оставаясь в разделе «Выполняемые элементы», щелкните задачу и выберите **Подтвердить**.
10. Чтобы кнопки **Подтвердить** и **Отклонить** стали активными, установите флажок рядом с выполняемым элементом.
11. Чтобы просмотреть сведения, нажмите стрелку раскрывающегося списка.
12. Чтобы утвердить задачу, щелкните **ПОДТВЕРДИТЬ**, введите причину и нажмите кнопку **ОК**.

User Operations GUIDED SETUP

Active Items Inactive Items

✓ APPROVE ✗ REJECT

☐ Index# Execution

☑ User Operation #f0d252

Request Details

Execution	rest-ix-1 #2
Summary	hello
Approvers	kern@vmware.com, f0f2@vmware.com
Requested By	kern@vmware.com
Requested On	Feb 4, 2020, 3:17:40 PM
Expires On	Feb 7, 2020, 3:17:40 PM

APPROVE REJECT VIEW DASHBOARD

13. Щелкните **Выполняемые элементы** и убедитесь, что конвейер продолжает работу.

Executions 3,347 items GUIDED SETUP

+ NEW EXECUTION

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

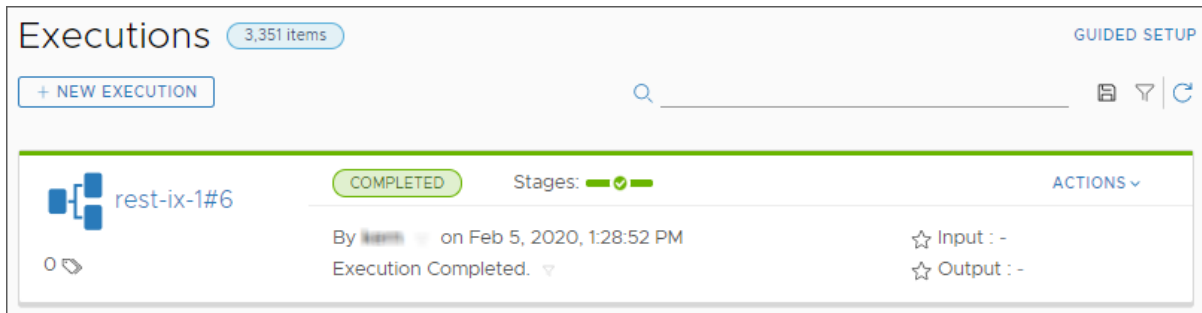
By kern on Feb 4, 2020, 3:41:05 PM

INPUT : -

OUTPUT : -

Comments: Testing

14. В случае сбоя конвейера исправьте ошибки, после чего сохраните конвейер и запустите его снова.



Дополнительные сведения о переменных и выражениях

Дополнительные сведения об использовании переменных и выражений для связывания задач конвейера см. в разделе [Какие переменные и выражения можно использовать для связывания задач конвейера в Code Stream](#).

Дополнительные сведения об использовании выходных данных задачи конвейера с условной переменной привязки см. в разделе [Использование привязок переменных в задаче «Условие» для запуска или остановки конвейера в Code Stream](#).

Использование привязок переменных в задаче «Условие» для запуска или остановки конвейера в Code Stream

Можно задать условие, в соответствии с которым работа конвейера будет продолжаться либо останавливаться в зависимости от результата выполнения той или иной задачи. Для привязки работы конвейера к результату выполнения задачи используется задача «Условие».

Задача **Условие** используется в конвейере в качестве точки принятия решения. С помощью задач «Условие» и заданных выражений условий можно выполнять оценку любых свойств конвейера, этапов и задач.

Результат выполнения задачи «Условие» определяет, будет ли запущена следующая задача в конвейере.

- Если результат соответствует выражению условия (истина), работа конвейера продолжается.
- Если результат не соответствует выражению условия (ложь), работа конвейера останавливается.

Примеры того, как результат выполнения одной задачи используется в качестве входных данных следующей задачи за счет связывания этих задач с помощью задачи «Условие», см. в разделе [Как использовать привязки переменных в конвейерах Code Stream](#).

Таблица 3-7. Связь между задачей «Условие», выражением условия и конвейером

Задача «Условие»	На что влияет	Функции
Задача «Условие»	Конвейер	В зависимости от того, является ли результат выполнения задач с типом Условие истинным или ложным, работа конвейера либо продолжается, либо останавливается.
Выражение условия	Результат выполнения задачи «Условие»	<p>В ходе работы конвейера выражение условия, входящее в задачу «Условие», присваивает результату выполнения задач значение «истина» или «ложь».</p> <p>Например, можно задать выражение условия, требующее, чтобы результат выполнения задачи «Условие» имел состояние Завершено или содержал номер сборки 74.</p> <p>Выражение условия отображается на вкладке «Задача» в области задачи «Условие».</p>

The screenshot shows the configuration interface for a 'Condition Task' in the vRealize Automation console. The task name is 'Task2' and the type is 'Condition'. A modal window titled 'Conditional Expression' is open, providing guidance on how to write the condition expression. It includes an example expression: `${Stage1.task1.output.status} == "COMPLETED" || ${input.buildNumber} == 74`. The modal also lists supported constructs and operators:

Type	Example
Pipeline variables	<code>\$(input.changeSetNumber)</code> (numeric binding) or <code>"\$(input.changeSetOwner)"</code> (string binding)
Task output variables	<code>\$(stage1.task1.output.responseCode)</code> (numeric binding) or <code>"\$(stage1.task1.output.status)"</code> (string binding)
Boolean values	<code>true / false</code>
Numeric values	<code>99</code> or <code>123.45</code> (quotes not allowed)
String values	<code>"Tested"</code> or <code>'Tested'</code>
Relational operators	<code><</code> , <code><=</code> , <code>></code> , <code>>=</code> , <code>==</code> , <code>!=</code>
Arithmetic operators	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>
Boolean	<code>&&</code> (logical and), <code> </code> (logical or)

Задача **Условие** отличается по принципу работы и поведению от параметра **При условии**, доступного в других типах задач.

В других типах задач параметр **При условии** управляет выполнением текущей задачи (а не последующих задач) в зависимости от того, было ли соблюдено предварительно заданное условие. В ходе работы конвейера выражение условия, заданное для параметра **При условии**, присваивает статус «истина» или «ложь» результату текущей задачи. Параметр **При условии** отображается на вкладке «Задача» вместе с соответствующим выражением условия.

В этом примере рассматривается использование задачи «Условие».

Необходимые условия

- Убедитесь в наличии конвейера, включающего в себя несколько этапов и задач.

Процедура

1. Задайте точку принятия решения в конвейере для вставки задачи «Условие».
2. Добавьте задачу «Условие» перед задачей, выполнение которой зависит от статуса «истина» или «ложь».
3. Добавьте выражение условия в задачу «Условие».

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

4. Проверьте задачу.
5. Сохраните конвейер, затем включите и запустите его.

Результаты

Обратите внимание на элементы конвейера при выполнении задачи «Условие»: продолжается ли его работа или же останавливается.

Следующие шаги

При откате развертывания конвейера также можно использовать задачу «Условие». Например, если добавить задачу «Условие» в конвейер с возможностью отката, служба Code Stream сможет пометить случаи аварийного завершения работы конвейера в соответствии с заданным выражением условия и инициировать запуск единой процедуры отката при возникновении различных сбоев.

Чтобы выполнить откат развертывания, см. раздел [Как откатить развертывание в Code Stream](#).

Какие переменные и выражения можно использовать для связывания задач конвейера в Code Stream

С помощью переменных и выражений в задачах конвейера можно задавать входные и выходные параметры. Вводимые параметры связывают задачу конвейера с одной или несколькими переменными, выражениями или условиями и определяют особенности работы конвейера при его выполнении.

Конвейеры могут запускать простые или сложные решения по доставке программного обеспечения

При связывании задач конвейера друг с другом можно добавлять выражения по умолчанию и сложные выражения. В результате конвейер может запускать простые или сложные решения по доставке программного обеспечения.

Чтобы создать параметры в конвейере, перейдите на вкладку **Входные данные** или **Выходные данные** и добавьте переменную, введя знак доллара **\$** и выражение. Например, следующий параметр используется в качестве входных данных задачи, вызывающей URL-адрес: `${Stage0.Task3.input.URL}`.

В формате привязки переменной используются компоненты синтаксиса, которые называются областями и ключами. `SCOPE` определяет контекст как входные или выходные данные, а `KEY` определяет сведения. В примере параметра `${Stage0.Task3.input.URL} input` — `SCOPE`, а URL-адрес — `KEY`.

Выходные свойства любой задачи можно сопоставить с любым количеством вложенных уровней привязки переменных.

Дополнительные сведения об использовании привязок переменных в конвейерах см. в разделе [Как использовать привязки переменных в конвейерах Code Stream](#).

Использование выражений со значком доллара с областями и ключами для связывания задач конвейера

Задачи конвейера можно связать с помощью выражений с переменными со значком доллара. Выражения вводятся в формате `${SCOPE.KEY.<PATH>}`.

Чтобы определить поведение задачи конвейера, в каждом выражении **SCOPE** представляет собой контекст, который используется в Code Stream. Области нужен ключ **KEY**, определяющий параметры действия, выполняемого задачей. Если значение **KEY** является вложенным объектом, можно указать дополнительный путь **PATH**.

В этих примерах содержится описание **SCOPE** и **KEY** и показано, как их можно использовать в конвейере.

Таблица 3-8. Использование ОБЛАСТИ (SCOPE) и КЛЮЧА (KEY)

SCOPE	Назначение выражения и примера	KEY	Способ применения SCOPE и KEY в конвейере
input	Входные свойства конвейера: <code>\${input.input1}</code>	Имя входного свойства	<p>Для создания ссылки на входное свойство конвейера в задаче используйте следующий формат:</p> <pre>tasks: mytask: type: REST input: url: \$ {input.url} action: get</pre> <pre>input: url: https:// www.vmware.com</pre>
output	Выходные свойства конвейера: <code>\${output.output1}</code>	Имя выходного свойства	<p>Для создания ссылки на выходное свойство для отправки уведомления используйте следующий формат:</p> <pre>notifications: email: - endpoint: MyEmailEndpoint subject: "Deployment Successful" event: COMPLETED to: - user@example.org body: Pipeline deployed the service successfully. Refer \$ {output.serviceURL}</pre>

Таблица 3-8. Использование ОБЛАСТИ (SCOPE) и КЛЮЧА (KEY) (продолжение)

SCOPE	Назначение выражения и примера	KEY	Способ применения SCOPE и KEY в конвейере
task input	Входные данные для задачи: \$ {MY_STAGE.MY_TASK.input. SOMETHING}	Показывает входные данные задачи в уведомлении	<p>При запуске задания Jenkins оно может ссылаться на имя задания, запускаемого из входных данных задачи. В этом случае используйте следующий формат для отправки уведомления:</p> <pre> notifications: email: - endpoint: MyEmailEndpoint stage: MY_STAGE task: MY_TASK subject: "Build Started" event: STARTED to: - user@example.org body: Jenkins job \$ {MY_STAGE.MY_TASK.i nput.job} started for commit id \$ {input.COMMITID}). </pre>
task output	Результат выполнения задачи: \$ {MY_STAGE.MY_TASK.output .SOMETHING}	Показывает результат выполнения задачи в следующей задаче	<p>Для создания ссылки на результат выполнения задачи 1 конвейера в задаче 2 используйте следующий формат:</p> <pre> taskOrder: - task1 - task2 tasks: task1: type: REST input: action: get url: https:// www.example.org/api /status task2: type: REST input: action: post url: https:// status.internal.exa mple.org/api/ activity payload: \$ {MY_STAGE.task1.out put.responseBody} </pre>

Таблица 3-8. Использование ОБЛАСТИ (SCOPE) и КЛЮЧА (KEY) (продолжение)

SCOPE	Назначение выражения и примера	KEY	Способ применения SCOPE и KEY в конвейере
var	Переменная: <code>\${var.myVariable}</code>	Ссылка на переменную в конечной точке	<p>Чтобы создать ссылку на секретную переменную в конечной точке для пароля, используйте следующий формат:</p> <pre> --- project: MyProject kind: ENDPOINT name: MyJenkinsServer type: jenkins properties: url: https:// jenkins.example.com username: jenkinsUser password: \$ {var.jenkinsPassword} </pre>
var	Переменная: <code>\${var.myVariable}</code>	Ссылка на переменную в конвейере	<p>Для создания ссылки на переменную в URL-адресе конвейера используйте следующий формат:</p> <pre> tasks: task1: type: REST input: action: get url: \$ {var.MY_SERVER_URL} </pre>
task status	<p>Состояние выполнения задачи:</p> <pre> \$ {MY_STAGE.MY_TASK.status} </pre> <p>\$</p> <pre> {MY_STAGE.MY_TASK.status Message} </pre>		
stage status	<p>Состояние выполнения этапа:</p> <pre> \${MY_STAGE.status} </pre> <p>\$</p> <pre> {MY_STAGE.statusMessage} </pre>		

Выражения по умолчанию

В конвейере можно использовать переменные с выражениями. Эта сводка содержит доступные для использования выражения по умолчанию.

Выражение	Описание
<code>\${comments}</code>	Комментарии, вводимые при запросе выполнения конвейера.
<code>\${duration}</code>	Продолжительность выполнения конвейера.
<code>\${endTime}</code>	Время окончания выполнения конвейера в формате UTC (если закончено).
<code>\${executedOn}</code>	Время запуска конвейера, аналогичное времени начала, в формате UTC.
<code>\${executionId}</code>	Идентификатор выполнения конвейера.
<code>\${executionUrl}</code>	URL-адрес для перехода к выполнению конвейера в пользовательском интерфейсе.
<code>\${name}</code>	Имя конвейера.
<code>\${requestBy}</code>	Имя пользователя, запросившего выполнение.
<code>\${stageName}</code>	Имя текущего этапа, используется в области этапа.
<code>\${startTime}</code>	Время начала выполнения конвейера в формате UTC.
<code>\${status}</code>	Состояние выполнения.
<code>\${statusMessage}</code>	Сообщение о состоянии выполнения конвейера.
<code>\${taskName}</code>	Имя текущей задачи, используется при вводе задачи или создании уведомления.

Использование SCOPE и KEY в задачах конвейера

Можно использовать выражения с любыми поддерживаемыми задачами конвейера. В этих примерах показано, как определить SCOPE и KEY и проверить синтаксис. В примерах кода в качестве имен этапа и задачи конвейера используются MY_STAGE и MY_TASK.

Дополнительные сведения о доступных задачах см. в разделе [Типы задач, доступные в Code Stream](#).

Таблица 3-9. Задачи перехода к следующему этапу

Задача	Scope	Key	Использование SCOPE и KEY в задаче
Пользовательская операция			
	Input	<p>summary: сводка по запросу пользовательской операции</p> <p>description: описание запроса пользовательской операции</p> <p>approvers: список адресов электронной почты утверждающих, каждая запись в котором может быть переменной, отделяемой запятой; также можно использовать точку с запятой для отдельных адресов электронной почты.</p> <p>approverGroups: список адресов групп утверждающих для платформы и удостоверения</p> <p>sendemail: если задано значение «истина», отправляется дополнительное уведомление по электронной почте при создании запроса или ответе</p> <p>expirationInDays: количество дней, соответствующее сроку действия запроса</p>	<pre> \${MY_STAGE.MY_TASK.input.summary} \${MY_STAGE.MY_TASK.input.description} \${MY_STAGE.MY_TASK.input.approvers} \$ {MY_STAGE.MY_TASK.input.approverGroups} \${MY_STAGE.MY_TASK.input.sendemail} \$ {MY_STAGE.MY_TASK.input.expirationInDays} </pre>
	Output	<p>index: шестизначная шестнадцатеричная строка, представляющая запрос</p> <p>respondedBy: имя учетной записи пользователя, утвердившего или отклонившего пользовательскую операцию</p> <p>respondedByEmail: адрес электронной почты ответившего пользователя</p> <p>comments: комментарии, предоставленные во время ответа</p>	<pre> \${MY_STAGE.MY_TASK.output.index} \${MY_STAGE.MY_TASK.output.respondedBy} \$ {MY_STAGE.MY_TASK.output.respondedByEmail} \${MY_STAGE.MY_TASK.output.comments} </pre>
Условие			
	Input	<p>condition: оцениваемое условие. Если условие принимает значение «истина», задача отмечается как завершенная; другие ответы приводят к сбою выполнения задачи.</p>	<pre> \${MY_STAGE.MY_TASK.input.condition} </pre>
	Output	<p>result: результат оценки</p>	<pre> \${MY_STAGE.MY_TASK.output.response} </pre>

Таблица 3-10. Задачи конвейера

Задача	Scope	Key	Использование SCOPE и KEY в задаче
Конвейер			
	Input	name: имя конвейера для запуска inputProperties: входные свойства, которые нужно передать в цикл выполнения вложенного конвейера	<code>\${MY_STAGE.MY_TASK.input.name}</code> <code>\${MY_STAGE.MY_TASK.input.inputProperties} #</code> Ссылка на все свойства <code>\$</code> <code>{MY_STAGE.MY_TASK.input.inputProperties.inpu</code> <code>tl1} # Ссылка на значение input1</code>
	Output	executionStatus: состояние выполнения конвейера executionIndex: индекс выполнения конвейера outputProperties: выходные свойства цикла выполнения конвейера	<code>\${MY_STAGE.MY_TASK.output.executionStatus}</code> <code>\${MY_STAGE.MY_TASK.output.executionIndex}</code> <code>\${MY_STAGE.MY_TASK.output.outputProperties}</code> <code># Ссылка на все свойства</code> <code>\$</code> <code>{MY_STAGE.MY_TASK.output.outputProperties.ou</code> <code>tput1} # Ссылка на значение output1</code>

Таблица 3-11. Автоматизация задач непрерывной интеграции

Задача	Scope	Key	Использование SCOPE и KEY в задаче
CI			
	Input	steps: набор строк, представляющих собой команды, которые требуется выполнить export: переменные среды, которые требуется сохранить после выполнения шагов artifacts: пути артефактов, которые требуется сохранить в общем пути process: набор элементов конфигурации для обработки JUnit, JaCoCo, Checkstyle, FindBugs	<code>\${MY_STAGE.MY_TASK.input.steps}</code> <code>\${MY_STAGE.MY_TASK.input.export}</code> <code>\${MY_STAGE.MY_TASK.input.artifacts}</code> <code>\${MY_STAGE.MY_TASK.input.process}</code> <code>\$</code> <code>{MY_STAGE.MY_TASK.input.process[0].path</code> <code>} # Ссылка на путь первой конфигурации</code>
	Output	exports: пара «ключ-значение», которая представляет собой экспортированные переменные среды из входного параметра export artifacts: путь успешно сохраненных артефактов processResponse: набор обработанных результатов для входного параметра process	<code>\${MY_STAGE.MY_TASK.output.exports} #</code> Ссылка на все операции экспорта <code>\$</code> <code>{MY_STAGE.MY_TASK.output.exports.myvar}</code> <code># Ссылка на значение myvar</code> <code>\${MY_STAGE.MY_TASK.output.artifacts}</code> <code>\$</code> <code>{MY_STAGE.MY_TASK.output.processRespons</code> <code>e}</code> <code>\$</code> <code>{MY_STAGE.MY_TASK.output.processRespons</code> <code>e[0].result} # Результат выполнения первой</code> конфигурации процесса
Настраиваемое			

Таблица 3-11. Автоматизация задач непрерывной интеграции (продолжение)

Задача	Scope	Key	Использование SCOPE и KEY в задаче
	Input	name: имя настраиваемой интеграции version: версия настраиваемой интеграции, опубликованная или устаревшая properties: свойства для отправки в настраиваемую интеграцию	<code>\${MY_STAGE.MY_TASK.input.name}</code> <code>\${MY_STAGE.MY_TASK.input.version}</code> <code>\${MY_STAGE.MY_TASK.input.properties} #</code> Ссылка на все свойства <code>\$</code> <code>{MY_STAGE.MY_TASK.input.properties.property1} #</code> Ссылка на значение property1
	Output	properties: выходные свойства из ответа настраиваемой интеграции	<code>\${MY_STAGE.MY_TASK.output.properties} #</code> Ссылка на все свойства <code>\$</code> <code>{MY_STAGE.MY_TASK.output.properties.property1} #</code> Ссылка на значение property1

Таблица 3-12. Автоматизация задач непрерывного развертывания: облачный шаблон

Задача	Scope	Key	Использование SCOPE и KEY в задаче
Облачный шаблон			
	Input	<p>action: одно из следующих значений: createDeployment, updateDeployment, deleteDeployment, rollbackDeployment.</p> <p>blueprintInputParams: используется для действий создание развертывания и обновление развертывания.</p> <p>allowDestroy: позволяет уничтожать компьютеры в процессе обновления развертывания.</p> <p>CREATE_DEPLOYMENT</p> <ul style="list-style-type: none"> ■ blueprintName: имя облачного шаблона ■ blueprintVersion: версия облачного шаблона <p>ИЛИ</p> <ul style="list-style-type: none"> ■ fileUrl: URL-адрес файла YAML удаленного облачного шаблона после выбора сервера GIT. <p>UPDATE_DEPLOYMENT</p> <p>Любые из следующих комбинаций:</p> <ul style="list-style-type: none"> ■ blueprintName: имя облачного шаблона ■ blueprintVersion: версия облачного шаблона <p>ИЛИ</p> <ul style="list-style-type: none"> ■ fileUrl: URL-адрес файла YAML удаленного облачного шаблона после выбора сервера GIT. <p>-----</p> <ul style="list-style-type: none"> ■ deploymentId: идентификатор развертывания. <p>ИЛИ</p> <ul style="list-style-type: none"> ■ deploymentName: имя развертывания. <p>-----</p>	

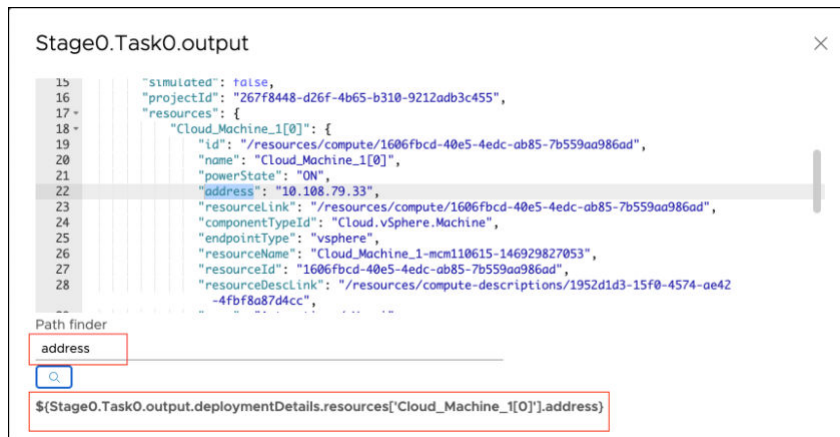
Таблица 3-12. Автоматизация задач непрерывного развертывания: облачный шаблон (продолжение)

Задача	Scope	Key	Использование SCOPE и KEY в задаче
		DELETE_DEPLOYMENT <ul style="list-style-type: none"> ■ deploymentId: идентификатор развертывания. ИЛИ <ul style="list-style-type: none"> ■ deploymentName: имя развертывания. ROLLBACK_DEPLOYMENT Любые из следующих комбинаций: <ul style="list-style-type: none"> ■ deploymentId: идентификатор развертывания. ИЛИ <ul style="list-style-type: none"> ■ deploymentName: имя развертывания. ----- <ul style="list-style-type: none"> ■ blueprintName: имя облачного шаблона ■ rollbackVersion: версия, до которой необходимо выполнить откат 	
	Output		<p>Параметры, которые можно привязать к другим задачам или выходным данным конвейера:</p> <ul style="list-style-type: none"> ■ Для доступа к имени развертывания используйте <code>\${Stage0.Task0.output.deploymentName}</code> ■ Для доступа к идентификатору развертывания используйте: <code>\${Stage0.Task0.output.deploymentId}</code> ■ Сведения о развертывании — это сложный объект. Доступ к внутренним сведениям можно получить с помощью результатов JSON. <p>Для доступа к любому свойству используйте оператор точки в соответствии с иерархией JSON. Например, для доступа к адресу ресурса Cloud_Machine_1[0] привязка <code>\$</code> будет следующей:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}</pre> <p>Аналогично, для конфигурации ресурсов привязка <code>\$</code> будет выглядеть следующим образом:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}</pre>

Таблица 3-12. Автоматизация задач непрерывного развертывания: облачный шаблон (продолжение)

Задача	Scope	Key	Использование SCOPE и KEY в задаче
			<p>В пользовательском интерфейсе Code Stream можно получить привязки \$ для любого свойства.</p> <ol style="list-style-type: none"> 1 В области выходных свойств задачи щелкните ПРОСМОТР ВЫХОДНЫХ ДАННЫХ JSON. 2 Чтобы найти привязку \$, введите любое свойство. 3 Нажмите значок поиска с соответствующей привязкой \$.

Пример выходных данных JSON:



Образец объекта сведений о развертывании:

```
{
  "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"
}

```


Таблица 3-13. Автоматизация задач непрерывного развертывания: Kubernetes

Задача	Scope	Key	Использование SCOPE и KEY в задаче
Kubernetes			
	Input	<p>action: одно из следующих значений — GET, CREATE, APPLY, DELETE, ROLLBACK.</p> <ul style="list-style-type: none"> ■ timeout: общее время ожидания для любого действия. ■ filterByLabel: дополнительная метка для фильтрации по действию GET с помощью K8S labelSelector <p>GET, CREATE, DELETE, APPLY</p> <ul style="list-style-type: none"> ■ yaml: встроенный код YAML для обработки и отправки в Kubernetes. ■ parameters: пара KEY, VALUE — замените \$\$KEY на VALUE в области ввода YAML. ■ filePath: относительный путь от конечной точки SCM Git (если указано), из которой нужно извлечь YAML. ■ scmConstants: пара KEY, VALUE — замените \$\$ {KEY} на VALUE в YAML, который извлечен по SCM. ■ continueOnConflict: если задано значение true и ресурс уже существует, выполнение задачи будет продолжено. <p>ROLLBACK</p> <ul style="list-style-type: none"> ■ resourceType: тип ресурса для отката ■ resourceName: имя ресурса для отката ■ namespace: пространство имен, в котором необходимо выполнить откат ■ revision: версия, до которой необходимо выполнить откат 	<p><code>\${MY_STAGE.MY_TASK.input.action} #</code> Определяет выполняемое действие.</p> <p><code>\${MY_STAGE.MY_TASK.input.timeout}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filterByLabel}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.yaml}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.parameters}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filePath}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.scmConstants}</code></p> <p><code>\$</code> <code>{MY_STAGE.MY_TASK.input.continueOnConflict}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceType}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceName}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.namespace}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.revision}</code></p>
	Output	<p>response: получение полного ответа</p> <p>response.<RESOURCE>: ресурс соответствует следующим значениям: configMaps, deployments, endpoints, ingresses, jobs, namespaces, pods, replicaSets, replicationControllers, secrets, services, statefulSets, nodes, loadBalancers.</p> <p>response.<RESOURCE>.<KEY> — ключ соответствует одному из следующих значений: apiVersion, kind, metadata, spec.</p>	<p><code>\${MY_STAGE.MY_TASK.output.response}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.response.}</code></p>

Таблица 3-14. Интеграция приложений для разработки, тестирования и развертывания

Задача	Scope	Key	Использование SCOPE и KEY в задаче
Bamboo			
	Input	plan: имя плана planKey: ключ плана variables: переменные, которые будут переданы в план parameters: параметры, которые будут переданы в план	<code>\${MY_STAGE.MY_TASK.input.plan}</code> <code>\${MY_STAGE.MY_TASK.input.planKey}</code> <code>\${MY_STAGE.MY_TASK.input.variables}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code> # Ссылка на все параметры <code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # ссылка на значение param1
	Output	resultUrl: URL-адрес результирующей сборки buildResultKey: ключ результирующей сборки buildNumber: номер сборки buildTestSummary: сводка выполненных тестов. successfulTestCount: тест пройден failedTestCount: тест не пройден skippedTestCount: тест пропущен artifacts: артефакты из сборки	<code>\${MY_STAGE.MY_TASK.output.resultUrl}</code> <code>\${MY_STAGE.MY_TASK.output.buildResultKey}</code> <code>\${MY_STAGE.MY_TASK.output.buildNumber}</code> <code>\${MY_STAGE.MY_TASK.output.buildTestSummary}</code> # Ссылка на все результаты <code>\${MY_STAGE.MY_TASK.output.successfulTestCount}</code> # Ссылка на определенный тестовый результат <code>\${MY_STAGE.MY_TASK.output.buildNumber}</code>
Jenkins			
	Input	job: имя задания Jenkins parameters: параметры, которые будут переданы в задание	<code>\${MY_STAGE.MY_TASK.input.job}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code> # Ссылка на все параметры <code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # Ссылка на значение параметра
	Output	job: имя задания Jenkins jobId: идентификатор результирующего задания, например 1234 jobStatus: состояние в Jenkins jobResults: результаты по объему протестированного/проверенного кода jobUrl: URL-адрес результирующего цикла выполнения задания	<code>\${MY_STAGE.MY_TASK.output.job}</code> <code>\${MY_STAGE.MY_TASK.output.jobId}</code> <code>\${MY_STAGE.MY_TASK.output.jobStatus}</code> <code>\${MY_STAGE.MY_TASK.output.jobResults}</code> # Ссылка на все результаты <code>\${MY_STAGE.MY_TASK.output.jobResults.junitResponse}</code> # Ссылка на результаты JUnit <code>\${MY_STAGE.MY_TASK.output.jobResults.jacocoResponse}</code> # Ссылка на результаты JaCoCo <code>\${MY_STAGE.MY_TASK.output.jobUrl}</code>
TFS			

Таблица 3-14. Интеграция приложений для разработки, тестирования и развертывания (продолжение)

Задача	Scope	Key	Использование SCOPE и KEY в задаче
	Input	projectCollection: коллекция проектов из TFS teamProject: выбранный проект из доступной коллекции buildDefinitionId: идентификатор определения сборки, которую требуется запустить	<code>\${MY_STAGE.MY_TASK.input.projectCollection}</code> <code>\${MY_STAGE.MY_TASK.input.teamProject}</code> <code>\${MY_STAGE.MY_TASK.input.buildDefinitionId}</code>
	Output	buildId: результирующий идентификатор сборки buildUrl: URL-адрес для перехода к сводке по сборке logUrl: URL-адрес для перехода к журналам dropLocation: расположение сброса артефактов (при наличии)	<code>\${MY_STAGE.MY_TASK.output.buildId}</code> <code>\${MY_STAGE.MY_TASK.output.buildUrl}</code> <code>\${MY_STAGE.MY_TASK.output.logUrl}</code> <code>\${MY_STAGE.MY_TASK.output.dropLocation}</code>
vRO			
	Input	workflowId: идентификатор рабочего процесса, который требуется запустить parameters: параметры, которые будут переданы в рабочий процесс	<code>\${MY_STAGE.MY_TASK.input.workflowId}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code>
	Output	workflowExecutionId: идентификатор цикла выполнения рабочего процесса properties: выходные свойства из цикла выполнения рабочего процесса	<code>\${MY_STAGE.MY_TASK.output.workflowExecutionId}</code> <code>\${MY_STAGE.MY_TASK.output.properties}</code>

Таблица 3-15. Интеграция других приложений с помощью API-интерфейса

Задача	Scope	Key	Использование SCOPE и KEY в задаче
REST			
	Input	url: URL-адрес вызова action: используемый метод HTTP headers: передаваемые заголовки HTTP payload: полезные данные запроса fingerprint: отпечаток пальца для сопоставления, если для URL-адреса используется протокол https allowAllCerts: если задано значение «истина», принимается любой сертификат с URL-адресом типа https	<code>\${MY_STAGE.MY_TASK.input.url}</code> <code>\${MY_STAGE.MY_TASK.input.action}</code> <code>\${MY_STAGE.MY_TASK.input.headers}</code> <code>\${MY_STAGE.MY_TASK.input.payload}</code> <code>\${MY_STAGE.MY_TASK.input.fingerprint}</code> <code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code>

Таблица 3-15. Интеграция других приложений с помощью API-интерфейса (продолжение)

Задача	Scope	Key	Использование SCOPE и KEY в задаче
	Output	<p>responseCode: код ответа HTTP</p> <p>responseHeaders: заголовки ответа HTTP</p> <p>responseBody: строковый формат полученного ответа</p> <p>responseJson: активный ответ, если параметр content-type имеет значение application/json</p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseHeaders}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.output.responseHeaders.header1} # Ссылка на заголовок ответа header1</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson} # Ссылка на ответ в формате JSON</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson.a.b.c} # Ссылка на вложенный объект после пути JSON "a.b.c" в ответе</code></p>
Опрос			
	Input	<p>url: URL-адрес вызова</p> <p>headers: передаваемые заголовки HTTP</p> <p>exitCriteria: критерии успешного или неудачного выполнения задачи. Пара ключ-значение: "успех" → выражение, "сбой" → выражение</p> <p>pollCount: количество выполняемых итераций. Максимальное количество опросов, которое может указать администратор Code Stream, составляет 10 000.</p> <p>pollIntervalSeconds: время ожидания между итерациями в секундах. Интервал между опросами должен быть не менее 60 секунд.</p> <p>ignoreFailure: если задано значение «истина», ошибки в промежуточных ответах игнорируются</p> <p>fingerprint: отпечаток пальца для сопоставления, если для URL-адреса используется протокол https</p> <p>allowAllCerts: если задано значение «истина», принимается любой сертификат с URL-адресом типа https</p>	<p><code>\${MY_STAGE.MY_TASK.input.url}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.headers}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.exitCriteria}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollCount}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollIntervalSeconds}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.ignoreFailure}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.fingerprint}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code></p>
	Output	<p>responseCode: код ответа HTTP</p> <p>responseBody: строковый формат полученного ответа</p> <p>responseJson: активный ответ, если параметр content-type имеет значение application/json</p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson} # Refer to response as JSON</code></p>

Таблица 3-16. Запуск удаленных и пользовательских сценариев

Задача	Scope	Key	Использование SCOPE и KEY в задаче
PowerShell Чтобы запустить оболочку PowerShell, необходимо, <ul style="list-style-type: none"> ■ чтобы было открыто подключение к удаленному узлу Windows. ■ Если предполагается ввести команду PowerShell в формате base64, сначала следует вычислить общую длину команды. Подробности см. в разделе Типы задач, доступные в Code Stream. 			
	Входные данные	host: IP-адрес или имя узла компьютера username: имя пользователя, используемое для подключения password: пароль, используемый для подключения useTLS: попытка подключения по протоколу https trustCert: если задано значение «истина», разрешено доверять самоподписанным сертификатам. script: запускаемый сценарий workingDirectory: путь к каталогу, на который требуется переключиться перед запуском сценария environmentVariables: пара «ключ-значение» переменной среды, которую требуется задать arguments: аргументы для передачи в сценарий	<pre> \${MY_STAGE.MY_TASK.input.host} \${MY_STAGE.MY_TASK.input.username} \${MY_STAGE.MY_TASK.input.password} \${MY_STAGE.MY_TASK.input.useTLS} \${MY_STAGE.MY_TASK.input.trustCert} \${MY_STAGE.MY_TASK.input.script} \$ {MY_STAGE.MY_TASK.input.workingDirectory} } \$ {MY_STAGE.MY_TASK.input.environmentVariables} \${MY_STAGE.MY_TASK.input.arguments} </pre>
	Выходные данные	response: содержимое файла \$SCRIPT_RESPONSE_FILE responseFilePath: значение \$SCRIPT_RESPONSE_FILE exitCode: код завершения процесса logFilePath: путь к файлу, содержащему stdout errorFilePath: путь к файлу, содержащему stderr	<pre> \${MY_STAGE.MY_TASK.output.response} \$ {MY_STAGE.MY_TASK.output.responseFilePath} \${MY_STAGE.MY_TASK.output.exitCode} \${MY_STAGE.MY_TASK.output.logFilePath} \${MY_STAGE.MY_TASK.output.errorFilePath} </pre>
SSH			

Таблица 3-16. Запуск удаленных и пользовательских сценариев (продолжение)

Задача	Scope	Key	Использование SCOPE и KEY в задаче
	Input	host: IP-адрес или имя узла компьютера username: имя пользователя, используемое для подключения password: пароль, используемый для подключения (при необходимости может использоваться закрытый ключ privateKey) privateKey: закрытый ключ privateKey, используемый для подключения passphrase: дополнительная парольная фраза для разблокировки закрытого ключа privateKey script: запускаемый сценарий workingDirectory: путь к каталогу, на который требуется переключиться перед запуском сценария environmentVariables: пара «ключ-значение» переменной среды, которую требуется задать	<pre> \${MY_STAGE.MY_TASK.input.host} \${MY_STAGE.MY_TASK.input.username} \${MY_STAGE.MY_TASK.input.password} \${MY_STAGE.MY_TASK.input.privateKey} \${MY_STAGE.MY_TASK.input.passphrase} \${MY_STAGE.MY_TASK.input.script} \$ {MY_STAGE.MY_TASK.input.workingDirectory} } \$ {MY_STAGE.MY_TASK.input.environmentVariables} </pre>
	Output	response: содержимое файла \$SCRIPT_RESPONSE_FILE responseFilePath: значение \$SCRIPT_RESPONSE_FILE exitCode: код завершения процесса logFilePath: путь к файлу, содержащему stdout errorFilePath: путь к файлу, содержащему stderr	<pre> \${MY_STAGE.MY_TASK.output.response} \$ {MY_STAGE.MY_TASK.output.responseFilePath} \${MY_STAGE.MY_TASK.output.exitCode} \${MY_STAGE.MY_TASK.output.logFilePath} \${MY_STAGE.MY_TASK.output.errorFilePath} </pre>

Использование привязки переменных в разных задачах

В этом примере показано, как использовать привязки переменных в задачах конвейера.

Таблица 3-17. Примеры форматов синтаксиса

Пример	Синтаксис
Использование результата выполнения задачи для уведомлений конвейера и выходных свойств конвейера	<code>\${<Stage Key>.<Task Key>.output.<Task output key>}</code>
Использование результата выполнения предыдущей задачи в качестве входных данных для текущей	<code>\${<Previous/Current Stage key>.<Previous task key not in current Task group>.output.<task output key>}</code>

Дополнительные сведения

Дополнительные сведения о переменных привязки в задачах см. здесь:

- [Как использовать привязки переменных в конвейерах Code Stream](#)
- [Использование привязок переменных в задаче «Условие» для запуска или остановки конвейера в Code Stream](#)
- [Типы задач, доступные в Code Stream](#)

Отправка уведомлений о работе конвейера в службу Code Stream

Уведомления позволяют обмениваться данными с рабочими группами и сообщать им состояние конвейеров в службе Code Stream.

Можно настроить отправку уведомлений Code Stream во время работы конвейера в зависимости от состояния конвейера, этапа или задачи в целом.

- Уведомление по электронной почте отправляется в следующих случаях.
 - Завершение, ожидание, сбой, отмена или запуск конвейера.
 - Завершение, сбой или запуск этапа.
 - Завершение, ожидание, сбой или запуск задачи.
- Уведомление о запросе создает запрос в службу поддержки и назначает его участнику рабочей группы в следующих случаях.
 - Сбой или завершение конвейера.
 - Сбой этапа.
 - Сбой задачи.
- Уведомление с веб-перехватчиком отправляет запрос в другое приложение в следующих случаях.
 - Сбой, завершение, ожидание, отмена или запуск конвейера.
 - Сбой, завершение или запуск этапа.

- Сбой, завершение, ожидание или запуск задачи.

Например, можно настроить отправку уведомления по электронной почте, относящегося к задаче, связанной с пользовательскими операциями, чтобы получить подтверждение на определенном этапе выполнения конвейера. При выполнении конвейера эта задача отправляет сообщение электронной почты пользователю, который должен подтвердить задачу. Если срок действия задачи «Пользовательская операция» задан в днях, часах или минутах, пользователь должен одобрить конвейер до его истечения. В противном случае работа контейнера ожидаемо завершается сбоем.

Для создания запроса в службу Jira при сбое задачи конвейера можно настроить соответствующее уведомление. Кроме того, для отправки запросов о состоянии конвейера в канал Slack, в зависимости от события конвейера, можно настроить уведомление с веб-перехватчиком.

Во всех типах уведомлений можно использовать переменные. Например, в URL-адресе уведомления с веб-перехватчиком можно использовать переменную `${var}`.

Необходимые условия

- Убедитесь в наличии нескольких конвейеров. Примеры использования см. в разделе [Глава 5 Учебники по использованию Code Stream](#).
- Чтобы отправлять уведомления по электронной почте, убедитесь в наличии доступа к работоспособному почтовому серверу. Для получения справки обратитесь к администратору.
- Чтобы создавать запросы в службу поддержки, например запросы Jira, убедитесь в наличии конечной точки. См. раздел [Что такое конечные точки в Code Stream](#).
- Чтобы отправить уведомление на основе интеграции, создайте уведомление с веб-перехватчиком. Затем убедитесь, что веб-перехватчик добавлен и работает. Уведомления можно использовать с такими приложениями, как Slack, GitHub или GitLab.

Процедура

1. Откройте конвейер.
2. Создание уведомлений по общему состоянию выполнения конвейера либо состоянию выполнения этапа или задачи:

Состояния, при которых создается уведомление:	Действия:
Состояние выполнения конвейера	Щелкните пустую область в рабочем окне конвейера.
Состояние выполнения этапа	Щелкните пустую область в этапе конвейера.
Состояние выполнения задачи	Щелкните задачу в этапе конвейера.

3. Откройте вкладку **Уведомления**.
4. Нажмите **Добавить**, выберите тип уведомления и введите подробные сведения об уведомлении.

5. Чтобы настроить отправку уведомления для Slack при успешном выполнении конвейера, создайте уведомление с веб-перехватчиком.
- Нажмите **Веб-перехватчик**.
 - Чтобы настроить уведомление Slack, введите следующую информацию.
 - Нажмите **Сохранить**.
 - В ходе выполнения конвейера в канал Slack отправляются уведомления о состоянии конвейера. Например, сообщения в канале Slack могут выглядеть следующим образом:

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

6. Чтобы создать запрос Jira, введите сведения о запросе.
- Нажмите **Запрос**.
 - Чтобы настроить уведомление Jira, введите информацию.
 - Нажмите **Сохранить**.

Notification

Send notification type

☐ Email
 ☒ Ticket
 ☐ Webhook

When pipeline *

☒ Fails
 ☐ Completes

Jira endpoint *

Jira-Notification ▼

Create Ticket

Jira project *

YourProject

Issue type *

Bug

Assignee *

username@yourcompany.com

Summary \$ *

Pipeline failed

Description \$

Research and correct

CANCEL

SAVE

Результаты

Поздравляем! Вы узнали о том, как создавать различные типы уведомлений для разных областей конвейера в службе Code Stream.

Следующие шаги

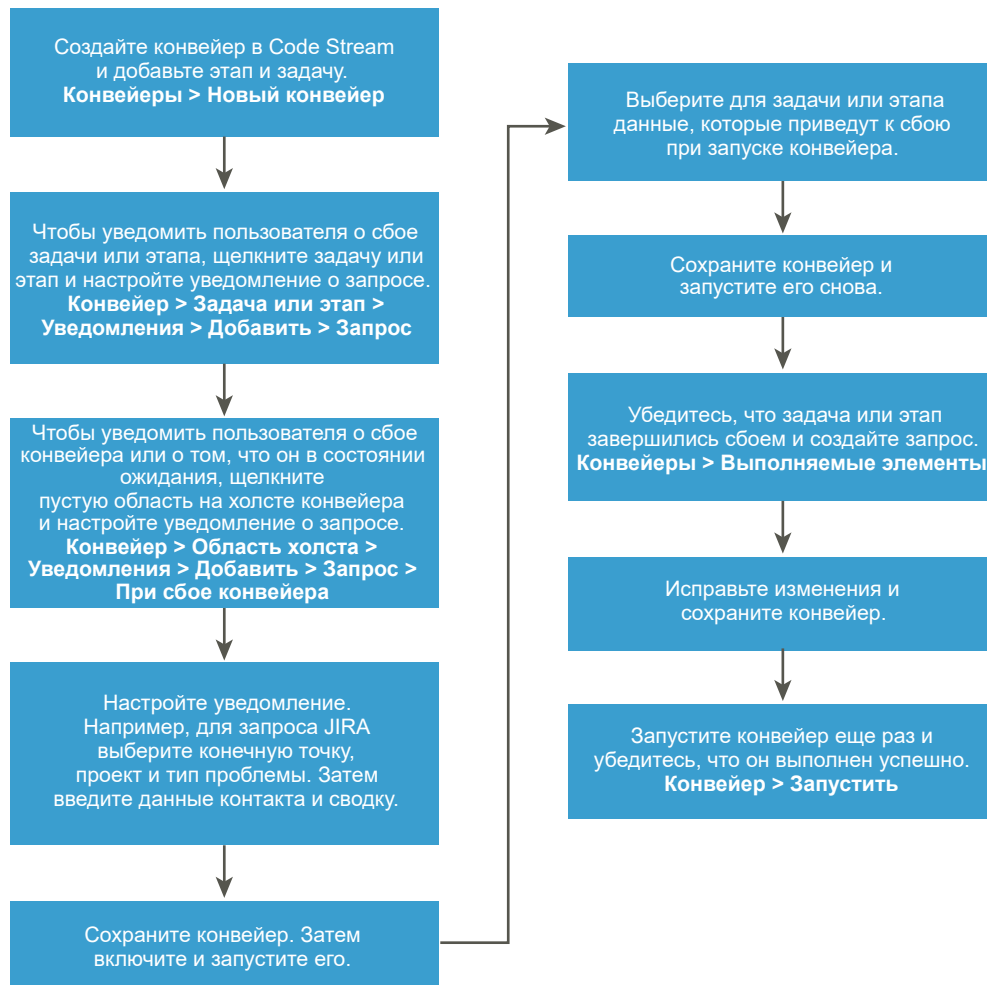
Подробный пример создания уведомлений см. в разделе [Создание запроса Jira в Code Stream при неудачном выполнении задачи конвейера](#).

Создание запроса Jira в Code Stream при неудачном выполнении задачи конвейера

Можно настроить создание запроса Jira службой Code Stream при неудачном выполнении этапа или задачи в конвейере. Запрос можно назначить лицу, которое должно устранить проблему. Можно также настроить создание запросов при успешном выполнении работы конвейера или когда конвейер находится в состоянии ожидания.

Для задачи, этапа или конвейера можно добавлять и настраивать уведомления. Служба Code Stream создает запрос на основании состояния выполнения задачи, этапа или конвейера, для которых настроены уведомления. Например, если недоступна конечная точка, можно настроить службу Code Stream так, чтобы при сбое задачи из-за невозможности подключиться к конечной точке создавался запрос Jira.

Можно также настроить отправку уведомлений при успешном завершении работы конвейера. Например, можно проинформировать рабочую группу по контролю качества об успешном завершении работы конвейера, чтобы ее участники могли проверить сборку и запустить другой тестовый конвейер. Либо можно отправлять уведомления рабочей группе по обеспечению производительности, чтобы ее участники могли оценить эффективность работы конвейера для подготовки к предпроизводственному тестированию или использованию в производственной среде.



В этом примере запрос Jira создается при неудачном выполнении задачи конвейера.

Необходимые условия

- Убедитесь в наличии действующей учетной записи Jira, позволяющей войти в экземпляр Jira.
- Убедитесь в наличии работоспособной конечной точки Jira.

Процедура

1. Щелкните задачу в конвейере.
2. В области настройки задач нажмите **Уведомления**.
3. Нажмите **Добавить** и настройте сведения о запросе.
 - а) Нажмите **Запрос**.
 - б) Выберите конечную точку Jira.
 - в) Введите имя проекта Jira и тип проблемы.

- г) Введите адрес электронной почты лица, которое должно получить запрос.
- д) Введите сводную информацию о запросе и описание проблемы, затем нажмите **Сохранить**.

Notification

Send notification type

☐ Email
 ☒ Ticket
 ☐ Webhook

When task *

☒ Fails

Jira endpoint *

TestJira

Create Ticket

Jira project *

YourProject

Issue type *

Bug

Assignee *

username@yourcompany.com

Summary \$ *

CI task failed

Description \$

Research and correct

CANCEL

SAVE

4. Сохраните конвейер, затем включите и запустите его.
5. Проверьте запрос.
 - а) Измените сведения о задаче, добавив данные, приводящие к сбою при ее выполнении.
 - б) Сохраните конвейер и запустите его снова.
 - в) Нажмите **Выполняемые элементы** и убедитесь, что работа конвейера завершилась сбоем.
 - г) В ходе выполнения работы конвейера убедитесь, что служба Code Stream создала и отправила соответствующий запрос.
 - д) Измените сведения о задаче, удалив приводящие к сбою данные, затем снова запустите конвейер и убедитесь, что его работа выполняется успешно.

Результаты

Поздравляем! Автоматический запрос Jira в службе Code Stream при неудачном выполнении задачи конвейера создан и назначен лицу, которое должно устранить проблему.

Следующие шаги

Добавьте дополнительные уведомления, чтобы сообщать рабочей группе о результатах выполнения задач и этапов конвейеров.

Как откатить развертывание в Code Stream

Откат настраивается в виде конвейера, включающего задачи, которые возвращают развертывание в предыдущее стабильное состояние после сбоя конвейера развертывания. Чтобы выполнить откат в случае сбоя, необходимо добавить конвейер отката к задачам или этапам.

Пользователи с разными ролями могут выполнять откат по разным причинам.

- Релиз-инженерам нужно получать от службы Code Stream подтверждения того, что выпуск успешно проходит все проверки. Эта информация помогает им понять, можно ли продолжать работу с этим выпуском или же нужно выполнить откат. Возможные причины для отката включают сбой при выполнении задач, отказ при прохождении проверки на удобство использования и эксплуатации, превышение пороговых значений показателей.
- Владелец среды требуется возможность повторить развертывание предыдущего выпуска, чтобы оперативно восстановить работоспособное состояние среды.
- Владелец среды требуется возможность отката развертывания в параллельные идентичные среды (Blue-Green) для сведения к минимуму простоев, связанных с неудачными выпусками.

При создании конвейера непрерывного предоставления на основе смарт-шаблона конвейера с выбранным параметром отката задача отката автоматически добавляется в задачи данного конвейера. В данном примере с помощью смарт-шаблона конвейера задается откат развертывания приложения в кластере Kubernetes, где используется модель динамического развертывания. Смарт-шаблон конвейера позволяет создать конвейер развертывания и один конвейер отката либо несколько.

- В конвейере развертывания происходит откат при неудачном выполнении задач «Обновление развертывания» или «Проверка развертывания».
- В конвейере отката происходит обновление развертывания с использованием более раннего образа.

Можно также вручную создать конвейер отката на основе пустого шаблона. Необходимо распланировать процесс отката, прежде чем создавать конвейер. Дополнительные сведения об откате см. в разделе [Планирование отката в Code Stream](#).

Необходимые условия


- Проверьте, являетесь ли вы участником проекта в Code Stream. Если нет, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).
- Настройте кластеры Kubernetes, в которых конвейер будет выполнять развертывание приложения. Настройте один кластер как кластер разработки, а другой — как производственный кластер.
- Убедитесь в наличии настроенного реестра Docker.
- Задайте проект, объединяющий все ваши рабочие объекты, включая конвейер, конечные точки и панели управления.


- Ознакомьтесь со смарт-шаблоном непрерывного предоставления, как описано в подразделе о непрерывном предоставлении в разделе [Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream перед использованием смарт-шаблона конвейера](#), например:
 - Создайте конечные точки разработки и производственной среды Kubernetes, которые будут развертывать образ приложения в кластерах Kubernetes.
 - Подготовьте файл YAML Kubernetes, который создаст пространство имен, службу и развертывание. Если необходимо загрузить образ из частного репозитория, файл YAML должен содержать раздел с секретом конфигурации Docker.

Процедура


1. Выберите пункт **Конвейеры > Новый конвейер > Смарт-шаблон > Непрерывное предоставление**.
2. Введите информацию в смарт-шаблон конвейера.
 - а) Выберите проект.
 - б) Введите имя конвейера, например **RollingUpgrade-Example**.
 - в) Выберите среды для приложения. Чтобы добавить откат в развертывание, необходимо выбрать среду **Произв..**
 - г) Нажмите **Выбрать**, выберите файл YAML Kubernetes и нажмите **Процесс**.
В смарт-шаблоне конвейера отображаются доступные службы и среды развертывания.
 - д) Выберите службу, которую конвейер будет использовать для развертывания.
 - е) Выберите конечные точки кластера для среды разработки и производственной среды.
 - ж) В качестве источника образа выберите **Входные данные среды выполнения конвейера**.
 - з) В качестве модели развертывания выберите **Динамическое обновление**.
 - и) Нажмите **Откат**.
 - к) Укажите **URL-адрес для проверки работоспособности**.

Smart Template: Continuous Delivery

Endpoint prerequisites  Kubernetes Docker Registry


Project * 

Pipeline name *

Environment  ☒ Development ☒ Production

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

Select service

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

1 services

Deployment



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


Image source * ☐ Docker trigger ☒ Pipeline runtime input

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

Rollback ☒

Health check URL *

CREATE CANCEL



3. Чтобы создать конвейер с именем RollbackUpgrade-Example, нажмите **Создать**.

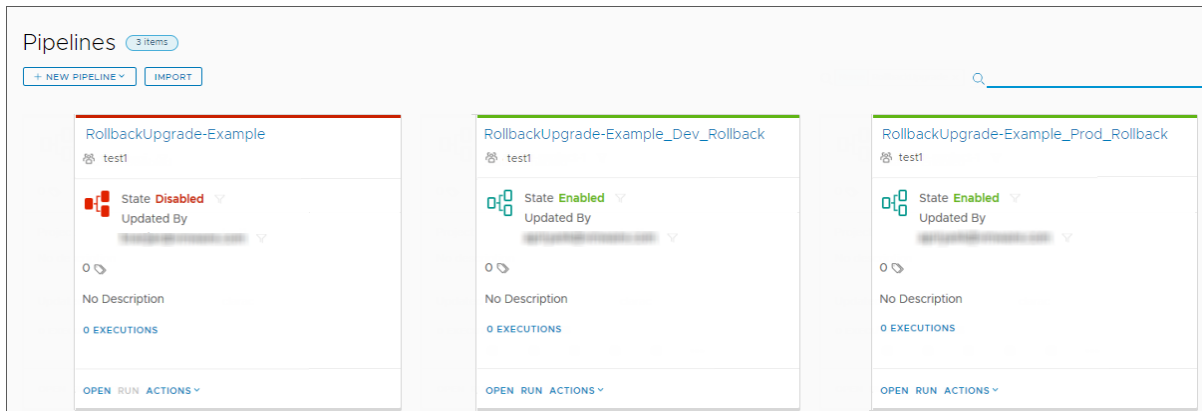
Отображается конвейер с именем RollbackUpgrade-Example и появляется значок отката для задач, которые могут выполнять откат на этапе разработки и на этапе производства.

The screenshot displays the vRealize Automation Code Stream interface. The main workspace shows a workflow titled 'RollbackUpgrade-Example' which is currently 'Disabled'. The workflow is organized into two stages: 'Development' and 'Production'. The 'Development' stage contains tasks like 'Create Namespace', 'Create Secret', and 'Create Service'. The 'Production' stage contains 'Update Deployment' and 'Verify Deployment'. A right-hand panel is open for configuring the 'Create Secret' task. This panel includes fields for 'Task name', 'Type' (set to 'Kubernetes'), 'Continue on failure', 'Execute task' (set to 'Always'), 'Kubernetes Task Properties' (cluster, timeout, action), 'Continue on conflict', 'Payload source' (set to 'Local definition'), and a 'Local YAML definition' field containing a Kubernetes secret manifest. At the bottom, there are buttons for 'EDIT', 'RUN', and 'CLOSE', along with a 'Last saved 9 minutes ago' timestamp.

4. Закройте конвейер.

На странице «Конвейеры» появляются созданный конвейер, а также новый конвейер для каждого этапа в этом конвейере.

- **RollingUpgrade-Example.** Code Stream по умолчанию деактивирует созданный конвейер. Это гарантирует просмотр конвейера перед запуском.
- **RollingUpgrade-Example_Dev_Rollback.** Сбой задач на этапе разработки, таких как **Создание службы**, **Создание секретного элемента**, **Создание развертывания** и **Проверка развертывания**, запускают этот конвейер отката разработки. Чтобы гарантировать выполнение отката задач разработки, Code Stream по умолчанию включает конвейер отката для разработки.
- **RollingUpgrade-Example_Prod_Rollback.** Сбой задач на этапе производства, таких как **Этап развертывания 1**, **Этап проверки 1**, **Развертывание этапа выпуска**, **Завершение этапа выпуска** и **Проверка этапа выпуска**, запускает этот конвейер отката для производственной среды. Чтобы гарантировать откат производственных задач, Code Stream по умолчанию включает конвейер отката для производства.



5. Включите и запустите созданный конвейер.

При запуске цикла выполнения Code Stream запрашивает входные параметры. Укажите образ и отметьте используемую конечную точку в репозитории Docker.

6. На странице «Выполняемые элементы» выберите пункт **Действия > Просмотр выполнения** и просмотрите выполнение конвейера.

Конвейер переходит в состояние **ВЫПОЛНЯЕТСЯ** и поочередно выполняет задачи этапа разработки. Если конвейеру не удастся выполнить задачу на этапе разработки, запускается конвейер с именем RollbackUpgrade-Example_Dev_Rollback, который выполняет откат развертывания, и состояние конвейера меняется на **ОТКАТ**.

[< BACK](#)

RollbackUpgrade-Example #1
ROLLING_BACK
0
ACTIONS ▾

Development

✓ Create Namespace
✓ Create Secret
✓ Create Service
● Create Deployment
⌂
● Verify Dep

Project

test1

Execution

RollbackUpgrade-Example #1

Status

ROLLING_BACK
RUNNING

Updated by

Executed by

Duration

12m 9s 186ms (01/11/2019 1:24 PM -)

Input Parameters ▾

image

demo-image-cs

tag

latest

Workspace

Details not available

Output Parameters ▾

The Execution did not output any properties

После отката на странице «Выполняемые элементы» отображаются два цикла выполнения конвейера RollbackUpgrade-Example.

- Для созданного конвейера выполнен откат и отображается состояние **ВЫПОЛНЕН_ОТКАТ**.
- Конвейер отката для разработки, который запустил и выполнил откат, отображает состояние **ЗАВЕРШЕНО**.

Executions
604 items

[+ NEW EXECUTION](#)
🔍

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

1
Rollback for RollbackUpgrade-Example#1

By started on 01/11/2019 1:36 PM

Execution Completed.

Comments: Triggered to rollback Development. Create Deployment of RollbackUpgrade-Example#1

RollbackUpgrade-Example#1
ROLLBACK_COMPLETED
Stages: 1/1

0

By started on 01/11/2019 1:24 PM

Create Deployment ROLLBACK_COMPLETED

Результаты

Поздравляем! Вы успешно настроили конвейер с откатом и просмотрели выполнение отката службой Code Stream при сбое конвейера.

Планирование сборки, интеграции и предоставления кода встроенными средствами в Code Stream

Прежде чем настроить в Code Stream выполнение сборки, интеграции и предоставления кода с помощью встроенной функции создания конвейера непрерывной интеграции и предоставления (CI/CD), непрерывной интеграции (CI) или непрерывного предоставления (CD), необходимо спланировать этот процесс. Затем можно создать конвейер с помощью одного из смарт-шаблонов конвейера или путем добавления этапов и задач вручную.

Чтобы спланировать сборку непрерывной интеграции и непрерывной доставки используйте примеры, которые показывают, как это можно сделать. В этих планах описываются предварительные требования и приведены общие сведения, которые помогут подготовить и эффективно использовать встроенную функцию сборки конвейеров.

В эту главу входят следующие разделы:

- Настройка рабочей области конвейера
- Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream перед использованием смарт-шаблона конвейера
- Планирование собственной сборки по модели непрерывной интеграции в Code Stream перед использованием смарт-шаблона конвейера
- Планирование собственной сборки по модели непрерывного предоставления в Code Stream перед использованием смарт-шаблона конвейера
- Планирование собственной сборки по модели непрерывной интеграции и предоставления в Code Stream перед добавлением задач вручную
- Планирование отката в Code Stream

Настройка рабочей области конвейера

Чтобы выполнять задачи непрерывной интеграции и настраиваемые задачи, необходимо настроить рабочую область конвейера Code Stream.

В рабочей области конвейера выберите Docker или Kubernetes для параметра **Тип** и укажите соответствующие конечные точки. Платформы Docker и Kubernetes обеспечивают управление всем жизненным циклом контейнера, развертываемого Code Stream для выполнения задачи непрерывной интеграции или настраиваемой задачи.

- Для рабочей области Docker требуется конечная точка узла Docker, URL-адрес образа построителя, реестр образов, рабочий каталог, кэш, переменные среды, ограничение ЦП и ограничение памяти. Также можно создать клон репозитория Git.
- Для рабочей области Kubernetes требуется конечная точка API-интерфейса Kubernetes, URL-адрес образа построителя, реестр образов, пространство имен, порт узла, запрос постоянного тома, рабочий каталог, переменные среды, ограничение ЦП и ограничение памяти. Также можно создать клон репозитория Git.

При настройке рабочей области конвейера используется много общих параметров, а также параметры, специфичные для типа рабочей области, как описано в следующей таблице.

Таблица 4-1. Зоны рабочей области, сведения и доступность

Выбор	Описание	Сведения и доступность
Тип	Тип рабочей области.	Параметр, доступный в Docker и Kubernetes.
Конечная точка узла	Конечная точка узла, где выполняется непрерывная интеграция и настраиваемые задачи.	Параметр доступен в рабочей области Docker при выборе конечной точки узла Docker. Параметр доступен в рабочей области Kubernetes при выборе конечной точки API-интерфейса Kubernetes.
URL-адрес образа построителя	Имя и расположение образа построителя. С помощью этого образа создается контейнер на узле Docker и кластере Kubernetes. Задачи непрерывной интеграции и настраиваемые задачи выполняются внутри этого контейнера.	Пример: fedora:latest Образ построителя должен включать <code>curl</code> или <code>wget</code> .
Реестр образов	Если образ построителя доступен в реестре и для реестра требуются учетные данные, необходимо создать конечную точку реестра образов, а затем выбрать ее здесь, чтобы образ можно было извлечь из реестра.	Параметр доступен в рабочих областях Docker и Kubernetes.
Рабочий каталог	Рабочий каталог — это расположение в контейнере, где выполняются действия задачи непрерывной интеграции, а также клонируется код, когда веб-перехватчик Git запускает выполнение конвейера.	Параметр, доступный в Docker и Kubernetes.
Пространство имен	Если не ввести пространство имен, Code Stream создаст уникальное имя в указанном кластере Kubernetes.	Параметр рабочей области Kubernetes.

Таблица 4-1. Зоны рабочей области, сведения и доступность (продолжение)

Выбор	Описание	Сведения и доступность
Прокси-сервер	<p>Чтобы обеспечить связь с модулем рабочей области в кластере Kubernetes, Code Stream разворачивает в пространстве имен <code>codestream-proxy</code> один экземпляр прокси-сервера для каждого кластера Kubernetes.</p> <p>Можно выбрать один из типов — Порт узла или Подсистема балансировки нагрузки — в зависимости от конфигурации кластера.</p> <p>Выбранный параметр зависит от характера развернутых кластеров Kubernetes.</p> <ul style="list-style-type: none"> ■ Обычно если URL-адрес сервера API Kubernetes, указанный в конечной точке, отображается в одном из главных узлов, рекомендуется выбрать Порт узла. ■ Если URL-адрес сервера API Kubernetes отображается в подсистеме балансировки нагрузки, как в случае с Amazon EKS (Elastic Kubernetes Service), выберите Подсистема балансировки нагрузки. 	
Порт узла	<p>Code Stream использует порт узла для обмена данными с контейнером, выполняемым в кластере Kubernetes.</p> <p>Если порт не выбран, Code Stream будет использовать временный порт, назначаемый Kubernetes. Необходимо убедиться, что правила брандмауэра настроены так, чтобы разрешать вхождение в диапазон временных портов (30000–32767).</p> <p>При вводе порта необходимо убедиться, что он не используется другой службой в кластере, а также что этот порт разрешен правилами брандмауэра.</p>	Параметр рабочей области Kubernetes.
Запрос постоянного тома	<p>Обеспечивает сохранение файлов в рабочей области Kubernetes для всех циклов выполнения конвейера. При указании имени запроса постоянного тома в нем можно хранить журналы, артефакты и кэш.</p> <p>Дополнительные сведения о создании запроса постоянного тома см. в документации Kubernetes на странице https://kubernetes.io/docs/concepts/storage/persistent-volumes/.</p>	Параметр рабочей области Kubernetes.
Переменные сред	<p>Пары «ключ-значение», которые здесь передаются, будут доступны для всех задач непрерывной интеграции и настраиваемых задач в конвейере при его выполнении.</p>	<p>Параметр, доступный в Docker и Kubernetes.</p> <p>Здесь можно передать ссылки на переменные.</p> <p>Переменные среды, указываемые в рабочей области, передаются во все задачи непрерывной интеграции и настраиваемые задачи в конвейере.</p> <p>Если не передать переменные среды здесь, их необходимо явно передать в каждую задачу непрерывной интеграции и настраиваемую задачу в конвейере.</p>

Таблица 4-1. Зоны рабочей области, сведения и доступность (продолжение)

Выбор	Описание	Сведения и доступность
Ограничения ЦП	Ограничения ресурсов ЦП для контейнера непрерывной интеграции или контейнера настраиваемых задач.	Значение по умолчанию — 1.
Ограничения памяти	Ограничения памяти для контейнера непрерывной интеграции или контейнера настраиваемых задач.	Единицей является МБ.
Клон Git	Когда вы выбираете клон Git и веб-перехватчик Git инициирует конвейер, код клонируется в рабочую область (контейнер).	Если вы не включили параметр клон Git , необходимо настроить другую явную задачу непрерывной интеграции в конвейере, чтобы сначала клонировать код, а затем выполнять другие шаги, такие как сборка и тестирование.
Кэш	<p>Рабочая область Code Stream позволяет кэшировать набор каталогов или файлов для ускорения последующих циклов выполнения конвейера. К примерам этих каталогов относятся <code>.m2</code> и <code>npm_modules</code>. Если вам не нужно кэширование данных циклов выполнения конвейера, запрос постоянного тома не требуется.</p> <p>Такие артефакты, как файлы или каталоги в контейнере, кэшируются для повторного использования в циклах выполнения конвейера. Например, можно кэшировать папку <code>node_modules</code> или <code>.m2</code>. Кэш принимает список путей.</p> <p>Например:</p> <pre>workspace: type: K8S endpoint: K8S-Micro image: fedora:latest registry: Docker Registry path: '' cache: - /path/to/m2 - /path/to/node_modules</pre>	<p>Параметр, специфичный для типа рабочей области.</p> <p>В рабочей области Docker кэш получают за счет использования общего пути на узле Docker для сохранения кэшируемых данных, артефактов и журналов.</p> <p>Чтобы разрешить использование кэша в рабочей области Kubernetes, необходимо указать запрос постоянного тома. В противном случае кэш будет недоступен.</p>

При использовании конечной точки API-интерфейса Kubernetes в рабочей области конвейера Code Stream создает необходимые для выполнения задачи непрерывной интеграции или настраиваемой задачи ресурсы Kubernetes, такие как ConfigMap, Secret и Pod. Code Stream обменивается данными с контейнером с помощью порта узла.

Чтобы предоставить общий доступ к данным для всех циклов выполнения конвейера, необходимо указать запрос постоянного тома. Code Stream подключит запрос постоянного тома к контейнеру, чтобы обеспечить хранение данных и его использование для последующих циклов выполнения конвейера.

Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream перед использованием смарт-шаблона конвейера

Для создания конвейера непрерывной интеграции и предоставления (CICD) в Code Stream можно использовать смарт-шаблон конвейера CICD. Чтобы спланировать собственную сборку по модели непрерывной интеграции и непрерывного предоставления, необходимо собрать сведения, которые требуются для заполнения смарт-шаблона конвейера, прежде чем создавать конвейер в данном примере плана.

Чтобы создать конвейер непрерывной интеграции и непрерывного предоставления, необходимо спланировать для него этапы непрерывной интеграции и непрерывного предоставления.

После ввода информации в смарт-шаблон конвейера при его сохранении создается конвейер, который включает в себя этапы и задачи. Он также указывает место развертывания образа на основе выбранных типов среды, таких как «Разработка» и «Производственная среда». Конвейер опубликует образ контейнера и выполнит действия, необходимые для его запуска. После запуска конвейера можно отслеживать тенденции по всем циклам его выполнения.

Если конвейер включает в себя образ из Docker Hub, перед запуском конвейера необходимо убедиться, что в этом образе есть встроенная функция `cURL` или `wget`. При выполнении конвейера Code Stream загружает двоичный файл, в котором для выполнения команд используется функция `cURL` или `wget`.

Сведения о настройке рабочей области см. в разделе [Настройка рабочей области конвейера](#).

Планирование этапа непрерывной интеграции (CI)

Чтобы спланировать этап непрерывной интеграции для конвейера, необходимо задать внешние и внутренние требования, а также определить сведения, которые нужно ввести в разделе непрерывной интеграции в смарт-шаблоне конвейера. Здесь приведен краткий обзор.

В этом примере используется рабочая область Docker.

Необходимые конечные точки и репозитории:

- Репозиторий исходного кода Git, куда разработчики могут сохранять код. Когда разработчики сохраняют изменения, Code Stream отправляет последнюю порцию кода в конвейер.
- Конечная точка Git для репозитория, в котором находится исходный код разработчика.
- Конечная точка Docker для узла сборки Docker, который будет выполнять команды сборки внутри контейнера.
- Конечная точка Kubernetes, благодаря которой Code Stream может развертывать образ в кластере Kubernetes.
- Образ построителя, который создает контейнер, в котором выполняются тесты непрерывной интеграции.
- Конечная точка реестра образов, с помощью которой узел сборки Docker может извлекать из него образ построителя.

Потребуется доступ к проекту. В проекте собираются все рабочие элементы, в том числе конвейер, конечные точки и панели управления. Проверьте, являетесь ли вы участником проекта в Code Stream. Если нет, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).

Потребуется веб-перехватчик Git, позволяющий Code Stream использовать триггер Git, который запускает конвейер, когда разработчики вносят изменения в код. См. раздел [Как использовать триггер Git в Code Stream для запуска конвейера](#).

Наборы средств сборки:

- Тип сборки, например, Maven.
- Все используемые средства сборки после обработки, такие как JUnit, JaCoCo, Checkstyle и FindBugs.

Средство публикации:

- Средство, например Docker, которое будет развертывать контейнер сборки.
- Тег образа, который может быть идентификатором отправленного кода или номером сборки.

Рабочая область сборки:

- Узел сборки Docker, который является конечной точкой Docker.
- Реестр образов. Раздел непрерывной интеграции в конвейере извлекает образ из выбранной конечной точки реестра. Контейнер запускает задачи CI и развертывает образ. Если реестру требуются учетные данные, сначала необходимо создать конечную точку реестра образов, а затем выбрать ее, чтобы узел мог извлечь образ из реестра.
- URL-адрес образа построителя, который создает контейнер, где выполняются тесты непрерывной интеграции.

Планирование этапа непрерывного предоставления (CD)

Чтобы спланировать этап непрерывного предоставления для конвейера, необходимо задать внешние и внутренние требования, а также определить сведения, которые нужно ввести в разделе непрерывного предоставления в смарт-шаблоне конвейера.

Необходимые конечные точки:

- Конечная точка Kubernetes, благодаря которой Code Stream может развертывать образ в кластере Kubernetes.

Типы среды и файлы:

- Все типы сред, где Code Stream будет развертывать приложение, например «Разработка» и «Производственная среда». Смарт-шаблон конвейера создает этапы и задачи в конвейере на основе выбранных типов среды.

Таблица 4-2. Этапы конвейера, создаваемые смарт-шаблоном конвейера непрерывной интеграции и предоставления

Содержимое конвейера	Функции
Этап сборки — публикации	Сборка и тестирование кода, создание образа построителя и публикация образа на узле Docker.
Этап разработки	Использование кластера разработки (AWS) Amazon Web Services для создания и развертывания образа. На этом этапе можно создать пространство имен в кластере и секретный ключ.
Этап переноса в производственную среду	Использование производственной версии выпуска VMware Tanzu Kubernetes Grid Integrated Edition (ранее — VMware Enterprise PKS) для развертывания образа в производственном кластере Kubernetes.

- Файл Kubernetes YAML, выбираемый в разделе непрерывного предоставления в смарт-шаблоне конвейера CICD.

Файл YAML Kubernetes включает в себя три обязательных раздела для пространства имен, службы и развертывания и один необязательный раздел для секретного элемента. Если планируется создать конвейер путем загрузки образа из частного репозитория, необходимо добавить раздел с секретом конфигурации Docker. Если в создаваемом конвейере используются только общедоступные образы, секретный элемент не требуется. В следующем примере файла YAML содержатся четыре раздела.

```

apiVersion: v1
kind: Namespace
metadata:
  name: codestream
  namespace: codestream
---
apiVersion: v1
data:
  .dockerconfigjson:
eyJhdXRocyI6eyJodHRwczovL2luZ1234567890lci5pby92MS8iOmsidXNlcm5hbWUiOiJhdXRvbWF0aW9uYmV0YSIsInBhc3N3b3JkIjo1Vk13YXJlQDEyMyIsImVtYWlsIjo1YXV0b21hdGlvbmJldGF1c2VyQGdtYWlsLmNvbSIsImF1dGgiOiJZWfYwYjIxaGRhbHZibUpsZEdFNlZrMTNZWEpsUURFeU13PT0ifX19
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

```

Примечание Файл YAML Kubernetes также используется в смарт-шаблоне конвейера непрерывного предоставления, например, в следующих примерах использования.

- Как развернуть приложение Code Stream в среде Blue-Green
 - Как откатить развертывание в Code Stream
 - Как использовать триггер Docker в Code Stream для запуска конвейера непрерывного предоставления
-

Чтобы применить файл в смарт-шаблоне, нажмите **Выбрать** и выберите файл YAML Kubernetes. Затем нажмите **Обработать**. В смарт-шаблоне конвейера отображаются доступные службы и среды развертывания. Выберите службу, конечную точку кластера и стратегию развертывания. Например, чтобы использовать модель развертывания Canary, выберите **Canary** и введите значение в процентах для этапа развертывания.

Smart Template: CI/CD

Step 2 of 2

Environment ⓘ ☒ Development ☒ Production

Kubernetes YAML files ⓘ
 Processed files: codestream.yaml

Select service

Deployment name	Service	Namespace	Image
codestream-demo	codestream-demo	codestream	https://codestream/Myapp

1 services

Deployment

Environment	Cluster Endpoint	Namespace
Development	Dev-AWS-Cluster	codestream-454709
Production	Prod-AWS-Cluster	codestream

Image source ⓘ ☐ Docker trigger ☒ Pipeline runtime input

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

Phase 1 ⓘ 20 %

Rollback ☐

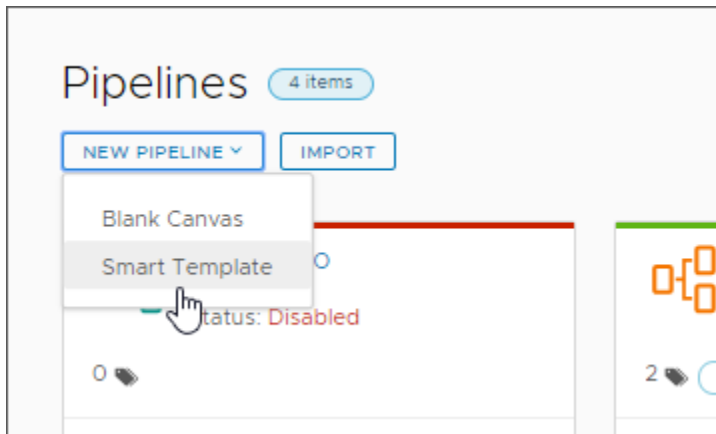
Health check URL ⓘ

Пример использования смарт-шаблона конвейера для создания конвейера для развертывания Blue-Green см. в разделе [Как развернуть приложение Code Stream в среде Blue-Green](#).

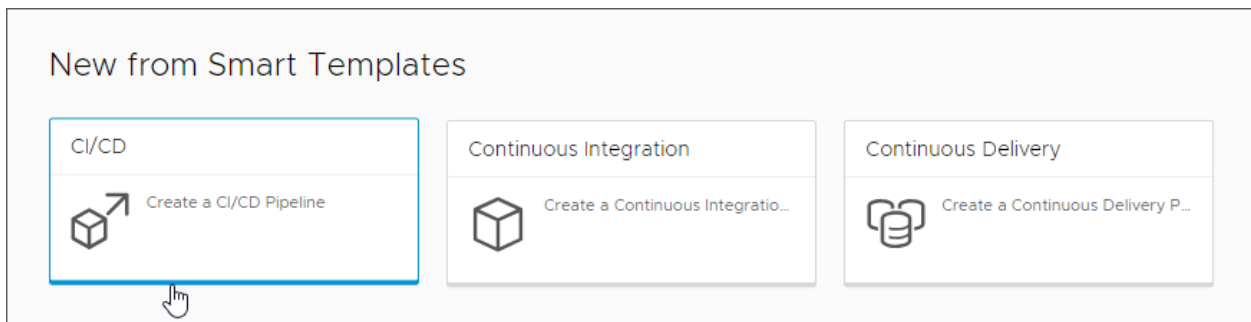
Создание конвейера CI/CD с помощью смарт-шаблона конвейера

После сбора всей информации и настройки нужных параметров можно создать конвейер на основе смарт-шаблона конвейера CI/CD.

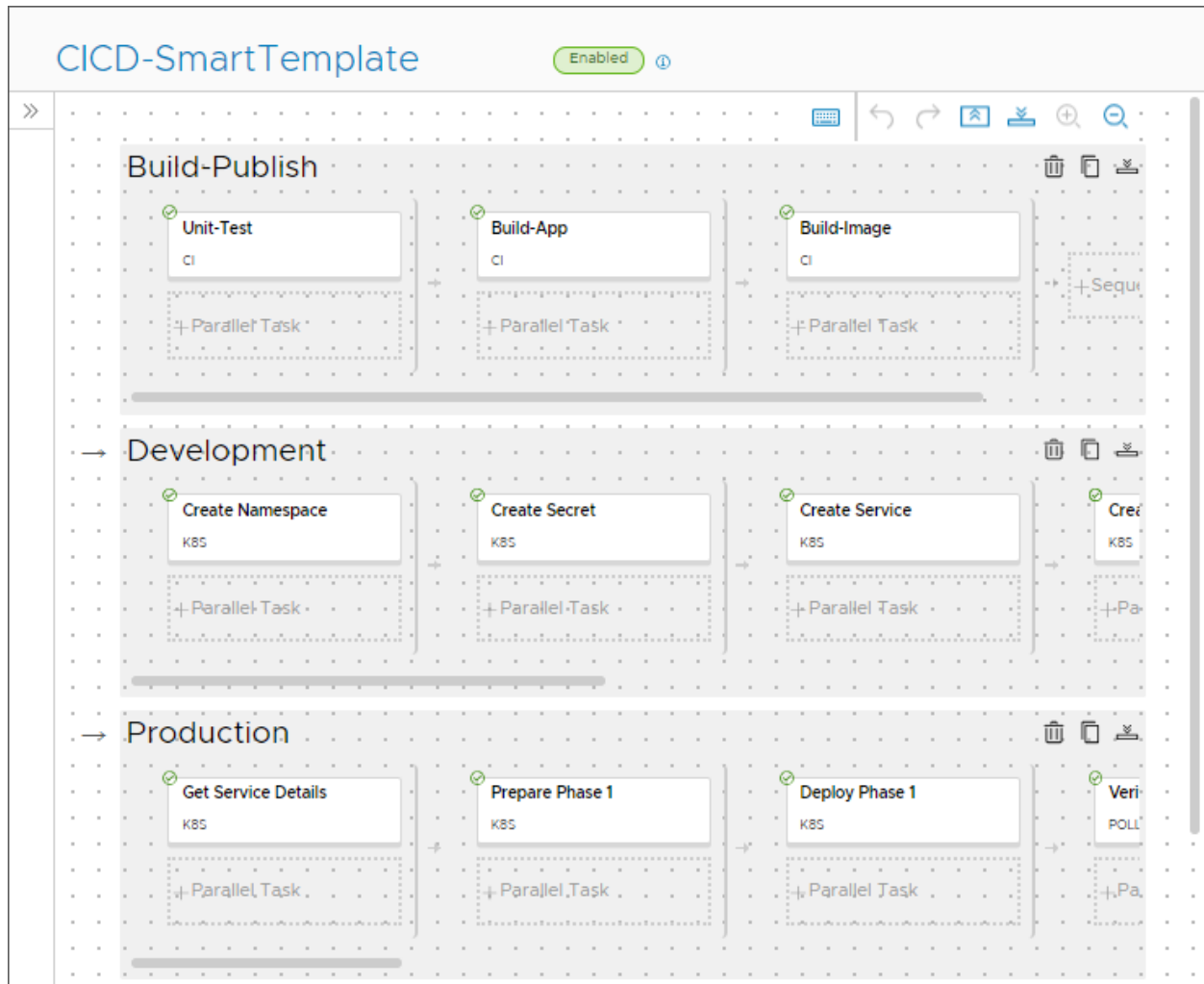
В разделе «Конвейеры» выберите **Создать конвейер > Смарт-шаблоны**.



Выберите смарт-шаблон конвейера CI/CD.



Заполните шаблон и сохраните конвейер с этапами, которые он формирует. Если необходимо внести окончательные изменения, конвейер можно отредактировать и сохранить.



Затем можно включить конвейер и запустить его. По окончании его работы можно выполнить следующие операции.

- Убедитесь, что конвейер успешно выполнен. Щелкните **Выполняемые элементы** и найдите свой конвейер. В случае сбоя исправьте ошибки и запустите его повторно.
- Убедитесь, что веб-перехватчик Git работает правильно. На вкладке Git **Действие** отобразятся события. Щелкните **Триггеры > Git > Действие**.
- Перейдите на панель управления конвейером и просмотрите тенденции. Щелкните **Панели управления** и найдите панель управления своего конвейера. Кроме того, можно создать настраиваемую панель управления для формирования отчета о дополнительных ключевых показателях эффективности.

См. подробный пример в разделе [Настройка непрерывной интеграции кода из репозитория GitHub или GitLab в конвейер Code Stream](#).

Планирование собственной сборки по модели непрерывной интеграции в Code Stream перед использованием смарт-шаблона конвейера

Для создания конвейера непрерывной интеграции в VMware Code Stream можно использовать смарт-шаблон конвейера непрерывной интеграции. Чтобы спланировать собственную сборку по модели непрерывной интеграции, необходимо собрать сведения, которые требуются для заполнения смарт-шаблона конвейера, прежде чем создавать конвейер в данном примере плана.

При заполнении смарт-шаблона конвейера создается конвейер непрерывной интеграции в репозитории и выполняются необходимые действия для его запуска. После запуска конвейера можно отслеживать тенденции по всем циклам его выполнения.

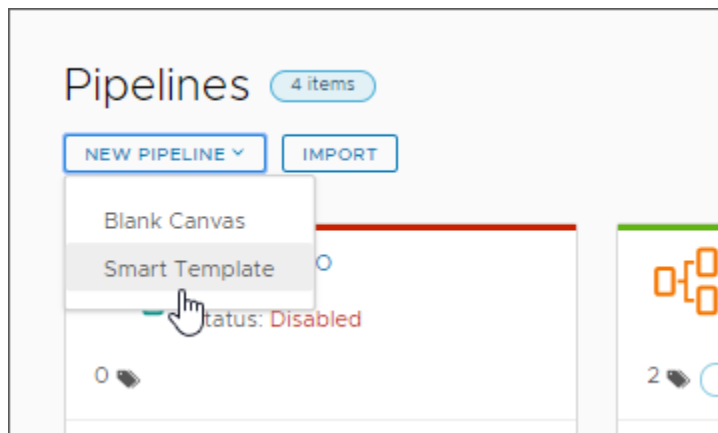
Для планирования собственной сборки перед использованием смарт-шаблона конвейера непрерывной интеграции выполните следующие действия.

- Задайте проект, объединяющий все ваши рабочие объекты, включая конвейер, конечные точки и панели управления.
- Соберите сведения для сборки, как описано в подразделе о непрерывном предоставлении в разделе [Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream перед использованием смарт-шаблона конвейера](#).

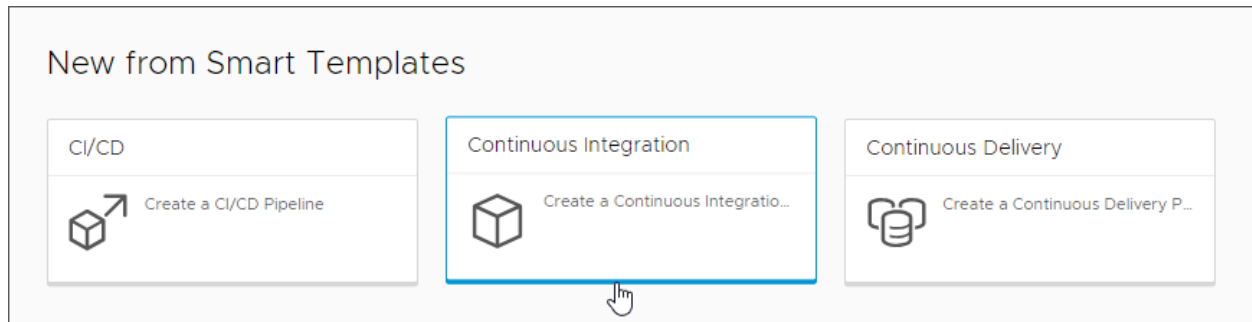
Например, добавьте конечную точку Kubernetes, в которой Code Stream будет проводить развертывание контейнера.

Затем необходимо создать конвейер с помощью смарт-шаблона конвейера непрерывной интеграции.

В разделе «Конвейеры» выберите **Смарт-шаблоны**.



Выберите смарт-шаблон конвейера непрерывной интеграции.



Чтобы сохранить конвейер с этапами, которые он создает, заполните шаблон и введите имя конвейера. Чтобы сохранить конвейер с этапами, которые он создает, щелкните **Создать**.

Рабочая область конвейера Code Stream поддерживает Docker и Kubernetes для выполнения задач непрерывной интеграции и настраиваемых задач.

Сведения о настройке рабочей области см. в разделе [Настройка рабочей области конвейера](#).

Для внесения финальных изменений можно изменить конвейер. Затем можно включить конвейер и запустить его. После запуска конвейера выполните следующие действия.

- Убедитесь, что конвейер успешно выполнен. Щелкните **Выполняемые элементы** и найдите свой конвейер. В случае сбоя исправьте ошибки и запустите его повторно.
- Убедитесь, что веб-перехватчик Git работает правильно. На вкладке Git **Действие** отобразятся события. Щелкните **Триггеры > Git > Действие**.
- Перейдите на панель управления конвейером и просмотрите тенденции. Щелкните **Панели управления** и найдите панель управления своего конвейера. Для отчетов о дополнительных ключевых показателях эффективности можно создать настраиваемую панель управления.

См. подробный пример в разделе [Настройка непрерывной интеграции кода из репозитория GitHub или GitLab в конвейер Code Stream](#).

Планирование собственной сборки по модели непрерывного предоставления в Code Stream перед использованием смарт-шаблона конвейера

Для создания конвейера непрерывного предоставления в Code Stream можно использовать смарт-шаблон конвейера непрерывного предоставления. Чтобы спланировать собственную сборку по модели непрерывного предоставления, необходимо собрать сведения, которые требуются для заполнения смарт-шаблона конвейера, прежде чем создавать конвейер в данном примере плана.

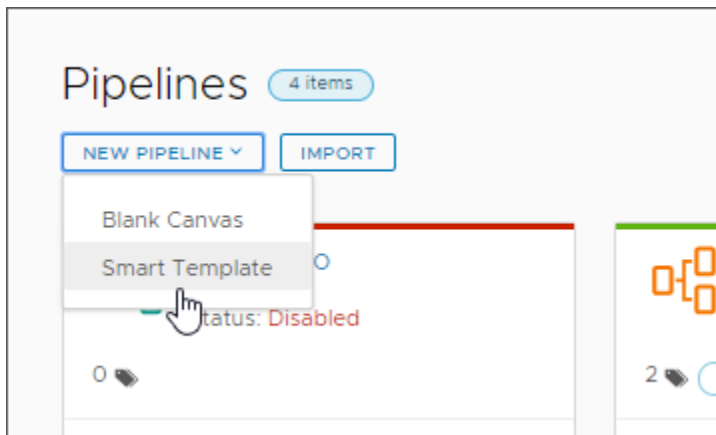
При заполнении смарт-шаблона конвейера создается конвейер непрерывного предоставления в репозитории и выполняются необходимые действия для его запуска. После запуска конвейера можно отслеживать тенденции по всем циклам его выполнения.

Для планирования собственной сборки перед использованием смарт-шаблона конвейера непрерывного предоставления выполните следующие действия.

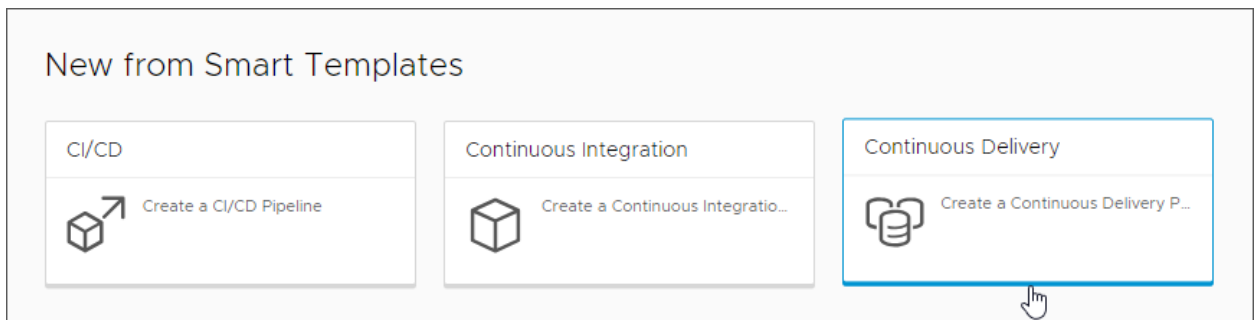
- Задайте проект, объединяющий все ваши рабочие объекты, включая конвейер, конечные точки и панели управления.
- Соберите сведения для сборки, как описано в подразделе о непрерывном предоставлении в разделе [Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream перед использованием смарт-шаблона конвейера](#). Например:
 - Добавьте конечную точку Kubernetes, в которой Code Stream будет развертывать контейнер.
 - Подготовьте файл YAML Kubernetes, который создает пространство имен, службу и развертывание. Если необходимо загрузить образ из частного репозитория, файл YAML должен содержать раздел с секретом конфигурации Docker.

Затем необходимо создать конвейер с помощью смарт-шаблона конвейера непрерывного предоставления.

В разделе «Конвейеры» выберите **Смарт-шаблоны**.



Выберите смарт-шаблон конвейера непрерывного предоставления.



Заполните шаблон и введите имя конвейера. Чтобы сохранить конвейер с этапами, которые он создает, щелкните **Создать**.

Рабочая область конвейера Code Stream поддерживает Docker и Kubernetes для выполнения задач непрерывной интеграции и настраиваемых задач.

Сведения о настройке рабочей области см. в разделе [Настройка рабочей области конвейера](#).

Для внесения финальных изменений можно изменить конвейер. Затем можно включить конвейер и запустить его. После запуска конвейера выполните следующие действия.

- Убедитесь, что конвейер успешно выполнен. Щелкните **Выполняемые элементы** и найдите свой конвейер. В случае сбоя исправьте ошибки и запустите его повторно.
- Убедитесь, что веб-перехватчик Git работает правильно. На вкладке Git **Действие** отобразятся события. Щелкните **Триггеры > Git > Действие**.
- Перейдите на панель управления конвейером и просмотрите тенденции. Щелкните **Панели управления** и найдите панель управления своего конвейера. Для отчетов о дополнительных ключевых показателях эффективности можно создать настраиваемую панель управления.

См. подробный пример в разделе [Настройка непрерывной интеграции кода из репозитория GitHub или GitLab в конвейер Code Stream](#).

Планирование собственной сборки по модели непрерывной интеграции и предоставления в Code Stream перед добавлением задач вручную

При создании конвейера непрерывной интеграции и предоставления (CICD) в Code Stream можно вручную добавить этапы и задачи. Чтобы спланировать собственную сборку CICD, требуется собрать необходимую информацию, затем создать конвейер и вручную добавить в него этапы и задачи.

Необходимо спланировать этапы непрерывной интеграции (CI) и непрерывного предоставления (CD) для конвейера. После создания конвейера и его запуска можно отслеживать тенденции по всем циклам его выполнения.

Если конвейер включает в себя образ из Docker Hub, перед запуском конвейера необходимо убедиться, что в этом образе есть встроенная функция cURL или wget. При выполнении конвейера Code Stream загружает двоичный файл, в котором для выполнения команд используется функция cURL или wget.

Рабочая область конвейера Code Stream поддерживает Docker и Kubernetes для выполнения задач непрерывной интеграции и настраиваемых задач.

Сведения о настройке рабочей области см. в разделе [Настройка рабочей области конвейера](#).

Планирование внешних и внутренних требований

Чтобы спланировать этапы CI и CD для конвейера, до его создания необходимо выполнить следующие требования.

В этом примере используется рабочая область Docker.

Чтобы создать конвейер на основе этого образца плана, нужно использовать узел Docker, репозиторий Git, Maven и несколько средств сборки для последующей обработки.

Необходимые конечные точки и репозитории:

- Репозиторий исходного кода Git, куда разработчики могут сохранять код. Когда разработчики сохраняют изменения, Code Stream отправляет последнюю порцию кода в конвейер.

- Конечная точка Docker для узла сборки Docker, который будет выполнять команды сборки внутри контейнера.
- Образ построителя, который создает контейнер, в котором выполняются тесты непрерывной интеграции.
- Конечная точка реестра образов, с помощью которой узел сборки Docker может извлекать из него образ построителя.

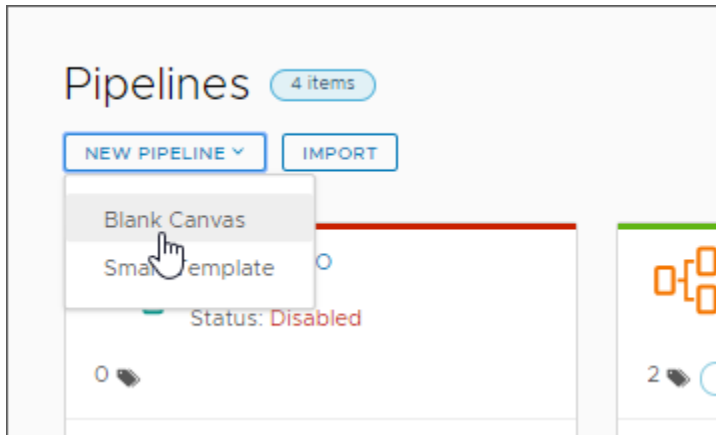
Потребуется доступ к проекту. В проекте собираются все рабочие элементы, в том числе конвейер, конечные точки и панели управления. Проверьте, являетесь ли вы участником проекта в Code Stream. Если нет, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).

Потребуется веб-перехватчик Git, позволяющий Code Stream использовать триггер Git, который запускает конвейер, когда разработчики вносят изменения в код. См. раздел [Как использовать триггер Git в Code Stream для запуска конвейера](#).

Создание конвейера непрерывной интеграции и предоставления и настройка рабочей области

Необходимо создать конвейер, затем настроить рабочую область, входные параметры конвейера и задачи.

Чтобы создать конвейер, выберите **Конвейеры > Создать конвейер > Пустой холст**.



На вкладке «Рабочая область» введите сведения о непрерывной интеграции.

- Добавьте узел сборки Docker.
- Введите URL-адрес образа построителя.
- Выберите конечную точку реестра образов, чтобы конвейер мог извлекать из него образ. Контейнер запускает задачи CI и развертывает образ. Если реестру требуются учетные данные, сначала необходимо создать конечную точку реестра образов, а затем выбрать ее, чтобы узел мог извлечь образ из реестра.

- Добавьте артефакты, которые необходимо кэшировать. Для успешного выполнения сборки артефакты, например, каталоги, загружаются как зависимости. Кэш — это расположение, в котором находятся такие артефакты. Например, зависимые артефакты могут включать в себя каталог `.m2` для Maven и каталог `node_modules` для Node.js. Эти каталоги кэшируются для различных циклов конвейера с целью экономии времени в ходе сборки.

The screenshot shows the 'Workspace' tab in the vRealize Automation interface. It contains a form for configuring continuous integration tasks. The form includes the following fields and options:

- Type:** Radio buttons for 'Docker' (selected) and 'Kubernetes'.
- Host endpoint:** A dropdown menu showing 'codestream-ci-test'. Below it is a note: 'Host endpoint for build location in CI task or for running Custom Integration Task code.'
- Builder image URL:** A text field containing 'automationbeta/cs-builder:latest'. Below it is a note: 'Name and location of the builder image. The CI tasks run on the container that the image creates.'
- Image registry:** A dropdown menu showing 'Docker Registry'. Below it is a note: 'The pipeline pulls the image from the selected registry endpoint. The container runs the CI tasks, and deploys your image. If the host can pull the image from the registry.'
- Working directory:** A text field with a placeholder '\$'.
- Cache:** A text field with a placeholder '\$' and a blue plus icon to its right.

На вкладке «Входные данные» настройте входные параметры конвейера.

- Если конвейер будет использовать входные параметры из события запуска Git, Gerrit или Docker, выберите тип триггера для автоматической вставки параметров. В число событий могут входить «Изменение темы» для Gerrit или Git или «Имя владельца события» для Docker. Если в конвейере не будут использоваться входные параметры, передаваемые событием, оставьте значение параметра «Автоматическая вставка параметров» **Нет**.
- Чтобы применить значение и описание к входному параметру конвейера, щелкните три вертикальные точки, а затем **Изменить**. Введенное значение используется в качестве входных данных для задач, этапов или уведомлений.
- Чтобы добавить входной параметр конвейера, щелкните **Добавить**. Например, можно добавить параметр `approvers`, который показывает значение по умолчанию для каждого цикла выполнения; его можно переопределить для отображения другого утверждающего лица во время выполнения.
- Чтобы добавить или удалить вставленный параметр, щелкните **Добавить/удалить вставленный параметр**. Например, можно удалить неиспользуемый параметр, чтобы уменьшить помехи на странице результатов и отображать только используемые входные параметры.

Input

Workspace

Model

Output

Input Parameters

The input parameters for this pipeline are passed to the pipeline before it runs.

When you add input parameters, and star the most useful or unique input parameter for each pipeline, the parameter appears in locations like the pipeline execution cards. For example, if you include the committer ID (GIT_COMMIT_ID) as an input parameter, you can select it as the starred input parameter to identify which developer commits trigger a pipeline execution before the pipeline runs.

Auto inject parameters

☐ Gerrit
 ☐ Git
 ☐ Docker
 ☒ None

[ADD](#)
[ADD/REMOVE INJECTED PARAMETERS](#)

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

8 items

Настройте конвейер, чтобы протестировать код.

- Добавьте и настройте задачу CI.
- Добавьте в код шаги для запуска `mvn test`.
- Чтобы выявить проблемы после запуска задачи, запустите средства сборки после обработки, такие как JUnit, JaCoCo, FindBugs и Checkstyle.

Task: Unit-Test

Notifications

Rollback

VALIDATE TASK

Task name *

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

Type *

Precondition

[SYNTAX GUIDE](#)

Continue on failure ☐

Continuous Integration

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

Steps

```

1 to demo-project
2 mvn test
  
```

Preserve artifacts

Specify the paths of artifact to preserve.

Export

Enter comma separated values

JUnit

JaCoCo

FindBugs

Checkstyle

Настройте конвейер для сборки кода.

- Добавьте и настройте задачу CI.

- Добавьте в код шаги для запуска `mvn clean install`.
- Укажите расположение и имя файла JAR, чтобы сохранить артефакт.

Task Build-App Notifications Rollback VALIDATE TASK

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

Type * CI

Precondition [SYNTAX GUIDE](#)

Continue on failure ☐

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

Steps *

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

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

Export
Enter comma separated values

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

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

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

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

Настройте конвейер для публикации образа на узле Docker:

- Добавьте и настройте задачу CI.
- Добавьте действия по отправке, экспорту, сборке и передаче образа.
- Добавьте ключ экспорта для `IMAGE` для применения в следующей задаче.

The screenshot shows the 'Build-Image' task configuration window. At the top, there are tabs for 'Task', 'Build-Image', 'Notifications', and 'Rollback', along with a 'VALIDATE TASK' button. The 'Task name' field is set to 'Build-Image' with a note: 'Can contain alphanumeric (a-z, A-Z, 0-9), whitespace, hyphen(-), and underscore(_) characters. Dot(.) is not allowed.' The 'Type' is set to 'CI'. There is a 'Precondition' field with a 'SYNTAX GUIDE' button. The 'Continue on failure' checkbox is unchecked. Under 'Continuous Integration', a note states: 'A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.' The 'Steps' section contains a code editor with the following script:

```
1 cd demo-project
2 export IMAGE=automationbeta/demo-cicd-smart-template:{{executionIndex}}
3 export DOCKER_HOST=http://10.211.211.27:4243
4 docker login --username=automationbeta --password=
5 docker build -t $IMAGE --file ./docker/Dockerfile .
6 docker push $IMAGE
```

Below the code editor are sections for 'Preserve artifacts', 'Export' (with an 'IMAGE' button), and configuration options for 'JUnit', 'JaCoCo', 'FindBugs', and 'Checkstyle', each with 'Label' and 'Path' fields.

После настройки рабочей области, входных параметров, задач тестирования и задач сборки сохраните конвейер.

Включение и запуск конвейера

После настройки конвейера с определением этапов и задач его можно сохранить и включить.

Затем дождитесь завершения работы конвейера и убедитесь, что он успешно выполнен. В случае сбоя исправьте ошибки и запустите его повторно.

После успешного выполнения конвейера могут потребоваться следующие действия.

- Проверьте выполнение конвейера и просмотрите результаты этапов, содержащих задачи.
- В рабочей области выполнения конвейера найдите сведения о своем контейнере и клонированном репозитории Git.
- В рабочей области просмотрите результаты работы средств последующей обработки и проверьте наличие ошибок, объем проверенного кода, наличие проблем со стилем.
- Убедитесь, что артефакт сохранен. Также убедитесь, что образ экспортирован с именем и значением IMAGE.
- Перейдите в репозиторий Docker и убедитесь, что конвейер опубликовал контейнер.

Подробный пример, показывающий, как Code Stream непрерывно интегрирует код, см. в разделе [Настройка непрерывной интеграции кода из репозитория GitHub или GitLab в конвейер Code Stream](#).

Планирование отката в Code Stream

В случае сбоя выполнения конвейера можно использовать откат, чтобы вернуть среду к предыдущему стабильному состоянию. Чтобы использовать функцию отката, необходимо спланировать алгоритм отката и понять, как его реализовать.

В алгоритме отката указываются этапы, необходимые для устранения сбоя при развертывании. Такой алгоритм представляет собой конвейер отката, включающий в себя одну или несколько последовательных задач, зависящих от типа развертывания, выполнение которого завершилось сбоем. Например, развертывание и откат традиционного приложения отличается от развертывания и отката приложения контейнера.

Чтобы вернуться к исправному состоянию развертывания, конвейер отката обычно выполняет следующие задачи.

- Сброс состояний и очистка сред.
- Запуск пользовательского сценария для отмены изменения.
- Развертывание предыдущей редакции развертывания.

Чтобы добавить откат в существующий конвейер развертывания, перед запуском конвейера развертывания необходимо присоединить конвейер отката к задачам или этапам в конвейере развертывания, который может потребовать отката.

Настройка отката

Чтобы настроить откат в развертывании, необходимо выполнить следующие действия.

- Создайте конвейер развертывания.
- Выявите потенциальные точки отказа в конвейере развертывания, которые будут инициировать откат, чтобы было можно присоединить конвейер отката. Например, можно присоединить конвейер отката к задаче условия или опроса, которая проверяет, успешно ли выполнена предыдущая задача. Дополнительные сведения о задачах условия см. в разделе [Использование привязок переменных в задаче «Условие» для запуска или остановки конвейера в Code Stream](#).
- Определите уровень отказа, который будет запускать конвейер отката, например сбой задачи или этапа. Откат также можно присоединить к этапу.
- Определите, какую задачу или задачи отката нужно будет выполнить в случае сбоя. Создайте конвейер отката с этими задачами.

Конвейер отката можно создать вручную, либо его может создать Code Stream.

- С помощью пустого холста можно вручную создать конвейер отката, который выполняется параллельно с существующим конвейером развертывания. После этого можно присоединить конвейер отката к одной или нескольким задачам в конвейере развертывания, которые запускают откат при сбое.
- Используйте смарт-шаблон конвейера, чтобы настроить конвейер развертывания с действием отката. Затем Code Stream автоматически создаст один или несколько конвейеров отката по умолчанию с предварительно определенными задачами, которые откатят развертывание в случае сбоя.

Подробный пример настройки конвейера непрерывного предоставления, когда для отката применяется смарт-шаблон конвейера, см. в разделе [Как откатить развертывание в Code Stream](#).

Что происходит, если в моем конвейере развертывания есть несколько задач или этапов с откатом

Если имеется несколько задач или задач и этапов с откатом, следует иметь в виду, что последовательность отката может отличаться.

Таблица 4-3. Определение последовательности отката

При добавлении отката в...	Когда выполняется откат...
Параллельные задачи	Если одна из параллельных задач завершается сбоем, откат этой задачи происходит после завершения или сбоя всех параллельных задач. Откат не выполняется сразу после сбоя задачи.
Как задача в рамках этапа, так и этап	Если происходит сбой задачи, выполняется ее откат. Если задача находится в группе параллельных задач, откат задачи выполняется после завершения или сбоя всех параллельных задач. Откат этапа выполняется после успешного или неудачного завершения отката задачи.

Рассмотрим конвейер, который содержит следующее:

- этап переноса в производственную среду с откатом;
- группу параллельных задач, для каждой из которых настроен отдельный процесс отката.

Для задачи с именем **UPD Deploy** настроен конвейер отката **RB_Deploy_US**. Если задача **UPD Deploy US** завершается сбоем, откат выполняется в соответствии с процессом, заданным в конвейере **RB_Deploy_US**.

The screenshot displays the vRealize Automation Code Stream interface for a workflow titled "RollbackUpgrade-Example", which is currently "Enabled". The interface is divided into several sections:

- Top Bar:** Contains tabs for "Workspace", "Input", "Model" (the active tab), and "Output".
- Workflow Canvas:** Shows a "Production" stage with three sequential tasks: "UPD Deploy US", "UPD Deploy UK", and "UPD Deploy AU", all using the "Kubernetes" provider. Below these tasks is a "+ Parallel Task" block. A "+ Stage" block is also visible at the bottom left of the canvas.
- Rollback Section:** On the right, the "Rollback" tab is active, showing a "Pipeline" named "RB_Deploy_US".
- Bottom Bar:** Includes buttons for "SAVE", "RUN", and "CLOSE", along with the text "Last saved 12 days ago".

Если **UPD Deploy US** завершается сбоем, конвейер **RB_Deploy_US** запускается после того, как **UPD Deploy UK** и **UPD Deploy AU** также были выполнены или завершились сбоем. Откат не выполняется сразу после сбоя **UPD Deploy US**. Так как для этапа переноса в производственную среду также настроен откат, после выполнения конвейера **RB_Deploy_US** выполняется конвейер отката этапа.

Учебники по использованию Code Stream

5

Code Stream позволяет моделировать и поддерживает жизненный цикл выпуска DevOps, а также непрерывно тестирует и публикует приложения в средах разработки и производственных средах.

Вы уже настроили все необходимое, и теперь можно использовать Code Stream. См. раздел [Глава 2 Настройка Code Stream для моделирования процесса выпуска](#).

Теперь можно создать конвейеры, которые автоматизируют сборку и тестирование кода разработчика до его выпуска в производственную среду. Code Stream можно использовать для развертывания приложений на основе контейнеров или традиционных приложений.

Таблица 5-1. Использование Code Stream в жизненном цикле DevOps

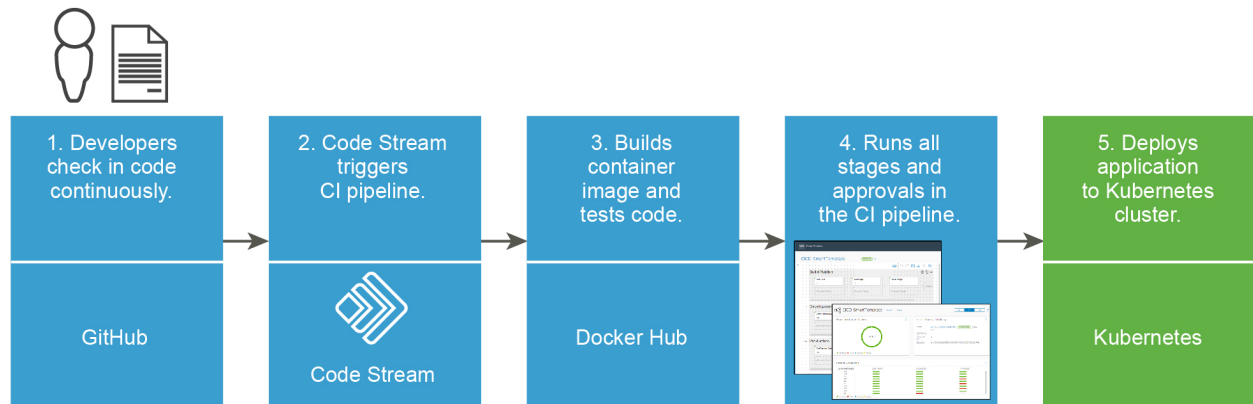
Компоненты	Примеры использования
Использование встроенной функции сборки в Code Stream.	<p>Создание конвейеров непрерывной интеграции и предоставления, непрерывной интеграции и непрерывного предоставления, которые обеспечивают непрерывную интеграцию, контейнеризацию и предоставление кода.</p> <ul style="list-style-type: none">■ Использование смарт-шаблона конвейера для автоматического создания конвейера.■ Добавление этапов и задач в конвейер вручную.
Выпуск приложений и автоматизация выпуска.	<p>Интеграция и выпуск приложений различными способами.</p> <ul style="list-style-type: none">■ Непрерывная интеграция кода из GitHub или репозитория GitLab в конвейер.■ Интеграция узла Docker для запуска задач непрерывной интеграции, соответствующая описанию в статье блога Создание узла Docker для vRealize Automation Code Stream.■ Автоматизация развертывания приложения с помощью облачного шаблона YAML.■ Автоматизация развертывания приложения в кластере Kubernetes.■ Выпуск приложения в развертывание по модели Blue-Green.■ Интеграция Code Stream с собственными средствами сборки, тестирования и развертывания.■ Использование интерфейса REST API, который интегрирует Code Stream с другими приложениями.
Отслеживание тенденций, показателей и ключевых показателей эффективности (КПЭ).	<p>Создание настраиваемых панелей управления и получение данных о производительности конвейеров.</p>
Устранение проблем.	<p>В случае сбоя выполнения конвейера используйте функцию создания запроса Jira в Code Stream.</p>

В эту главу входят следующие разделы:

- Настройка непрерывной интеграции кода из репозитория GitHub или GitLab в конвейер Code Stream
- Как автоматизировать выпуск приложения, развернутого с использованием облачного шаблона YAML в Code Stream
- Автоматизация выпуска приложения в кластере Kubernetes с помощью Code Stream
- Как развернуть приложение Code Stream в среде Blue-Green
- Интеграция собственных средств сборки, тестирования и развертывания со службой Code Stream
- Как использовать свойства ресурса из задачи облачного шаблона в следующей задаче
- Как использовать интерфейс REST API для интеграции Code Stream с другими приложениями
- Использование конвейера в виде кода в Code Stream

Настройка непрерывной интеграции кода из репозитория GitHub или GitLab в конвейер Code Stream

Разработчикам требуется непрерывная интеграция кода, хранящегося в репозитории GitHub или GitLab Enterprise. Служба Code Stream позволяет отслеживать изменения кода в репозитории и запускать работу конвейера каждый раз, когда разработчики вносят и сохраняют изменения кода.



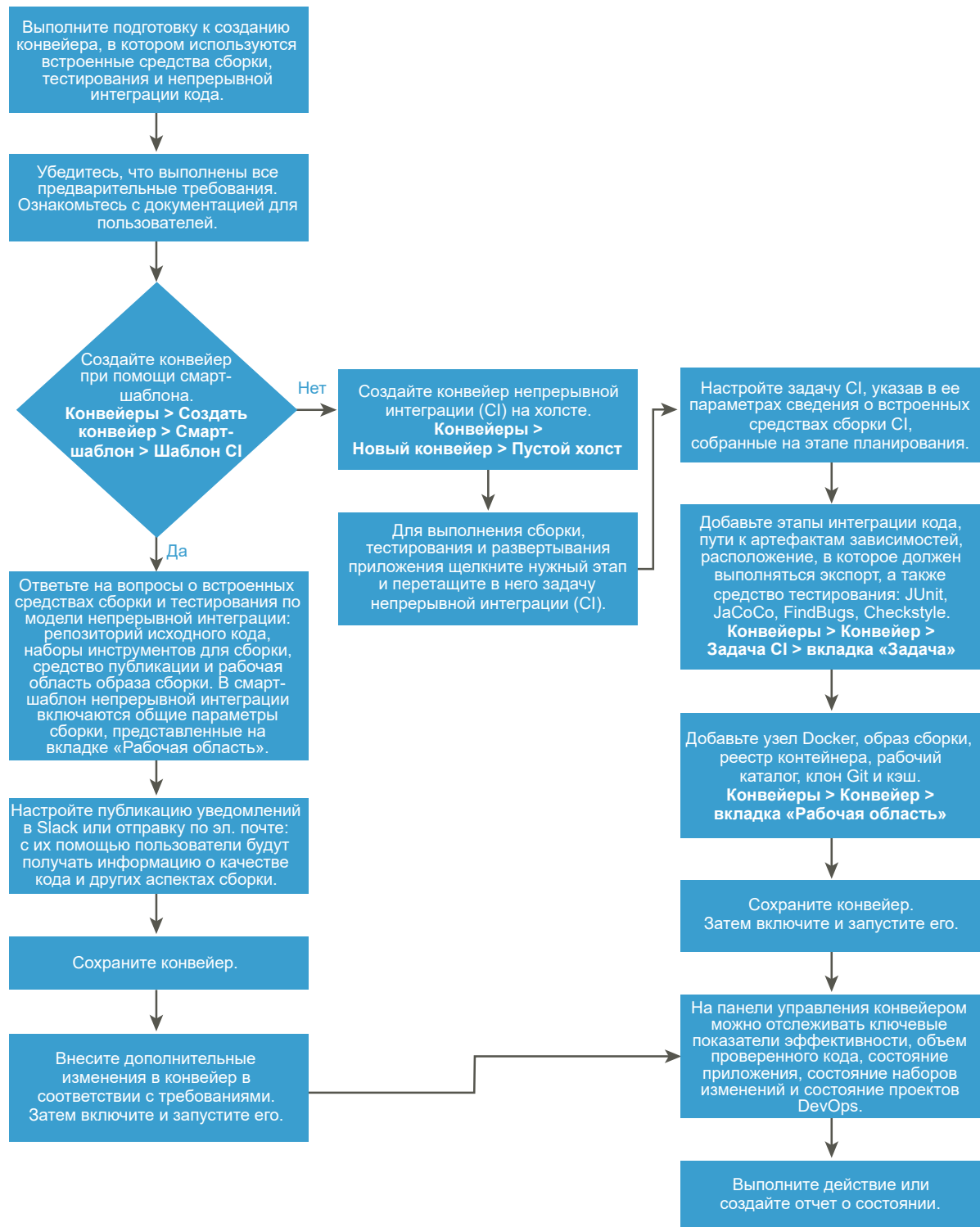
Чтобы служба Code Stream запускала работу конвейера при каждом изменении кода, необходимо использовать триггер Git. Каждый раз, когда разработчик будет сохранять изменения кода, служба Code Stream будет запускать конвейер.

Рабочая область конвейера Code Stream поддерживает Docker и Kubernetes для выполнения задач непрерывной интеграции и настраиваемых задач.

Дополнительные сведения о настройке рабочей области см. в разделе [Настройка рабочей области конвейера](#).

На следующей блок-схеме показан рабочий процесс, который можно использовать, если для создания конвейера используется смарт-шаблон или если конвейер создается вручную.

Рис. 5-1. Рабочий процесс, использующий смарт-шаблон конвейера или создающий конвейер вручную



В следующем примере используется рабочая область Docker.

Для сборки кода используйте узел Docker. В качестве средств тестирования, которые будут выполнять модульные тесты и контролировать объем протестированного кода, используйте JUnit и JaCoCo. Их также требуется включить в конвейер.

Затем можно использовать смарт-шаблон конвейера непрерывной интеграции, который создает конвейер непрерывной интеграции, выполняющий сборку, тестирование и развертывание кода в кластере Kubernetes рабочей группы проекта, размещенном в службе AWS. Для хранения артефактов зависимостей кода для задачи непрерывной интеграции можно использовать кэш; это позволяет экономить время при сборке кода.

В задачу конвейера, выполняющую сборку и тестирование кода, можно добавить несколько шагов по непрерывной интеграции. Эти шаги могут размещаться в том же рабочем каталоге, где Code Stream клонирует исходный код при запуске конвейера.

Чтобы выполнить развертывание кода в кластере Kubernetes, можно использовать задачу Kubernetes в конвейере. Затем необходимо включить и запустить конвейер. Затем внесите изменения в код, хранящийся в репозитории, и убедитесь, что это изменение запустило конвейер. На панелях управления можно отслеживать выполнение конвейера и составить отчет о тенденциях его работы после завершения выполнения.

В следующем примере для создания конвейера непрерывной интеграции, который непрерывно интегрирует код в конвейер, необходимо использовать соответствующий смарт-шаблон. В этом примере используется рабочая область Docker.

При необходимости конвейер можно создать вручную, самостоятельно добавив в него этапы и задачи. Дополнительные сведения о планировании сборок для обеспечения непрерывной интеграции и создании конвейера вручную см. в разделе [Планирование собственной сборки по модели непрерывной интеграции и предоставления в Code Stream перед добавлением задач вручную](#).

Необходимые условия

- Планирование сборок помогает добиться непрерывной интеграции. См. раздел [Планирование собственной сборки по модели непрерывной интеграции в Code Stream перед использованием смарт-шаблона конвейера](#).
- Убедитесь в наличии репозитория исходного кода GitLab. Для получения справки обратитесь к администратору Code Stream.
- Добавьте конечную точку Git. См. пример в разделе [Как использовать триггер Git в Code Stream для запуска конвейера](#).
- Добавьте веб-перехватчик, чтобы служба Code Stream могла выявлять изменения, вносимые в репозиторий GitHub или GitLab, и запускать работу конвейера в таких ситуациях. См. пример в разделе [Как использовать триггер Git в Code Stream для запуска конвейера](#).
- Добавьте конечную точку узла Docker, чтобы создать контейнер для задачи непрерывной интеграции. Этот контейнер может использоваться сразу для нескольких задач непрерывной интеграции. Дополнительные сведения о конечных точках см. в разделе [Что такое конечные точки в Code Stream](#).
- Получите URL-адрес образа, узел сборки и URL-адрес для образа сборки. Для получения справки обратитесь к администратору Code Stream.

- Убедитесь, что для тестирования используются средства JUnit и JaCoCo.
- Настройте внешний экземпляр для сборки непрерывной интеграции: Jenkins, TFS или Bamboo. Подключаемый модуль Kubernetes выполняет развертывание кода. Для получения справки обратитесь к администратору Code Stream.

Процедура

1. Убедитесь, что все необходимые предварительные условия выполнены.
2. Чтобы создать конвейер на основе смарт-шаблона, откройте смарт-шаблон конвейера непрерывной интеграции и заполните форму.
 - а) Выберите пункт **Конвейеры > Создать конвейер > Смарт-шаблон > Непрерывная интеграция**.
 - б) Введите в шаблон информацию о репозитории исходного кода, наборах инструментов для сборок, средстве публикации и рабочей области образа сборки.
 - в) Настройте отправку рабочей группе уведомлений по электронной почте или через Slack.
 - г) Чтобы создать конвейер на основе смарт-шаблона конвейера, нажмите **Создать**.
 - д) Чтобы внести дополнительные изменения в конвейер, нажмите **Изменить**, внесите правки и нажмите **Сохранить**.
 - е) Включите и запустите конвейер.
3. Чтобы создать конвейер вручную, добавьте этапы и задачи на холст и введите сведения о собственной сборке непрерывной интеграции в задачу непрерывной интеграции.
 - а) Выберите пункт **Конвейеры > Создать конвейер > Пустой холст**.
 - б) Щелкните этап и перетащите в него несколько задач непрерывной интеграции с панели навигации.
 - в) Чтобы настроить задачу непрерывной интеграции, щелкните ее, а затем перейдите на вкладку **Задача**.
 - г) Добавьте шаги, необходимые для непрерывной интеграции кода.
 - д) Добавьте пути к артефактам зависимостей.
 - е) Добавьте расположение для экспорта.
 - ж) Добавьте средства тестирования, которые планируется использовать.
 - з) Добавьте узел Docker и образ сборки.
 - и) Добавьте реестр контейнера, рабочий каталог и кэш.
 - к) Сохраните конвейер, затем включите его.
4. Внесите изменения в код, находящийся в репозитории GitHub или GitLab.
Триггер Git инициирует запуск конвейера.
5. Чтобы убедиться, что изменение кода привело к запуску контейнера, выберите параметр **Триггеры > Git > Действие**.

6. Чтобы посмотреть, как выполняется работа конвейера, нажмите **Выполняемые элементы** и убедитесь, что образ сборки создан и экспортирован.

The screenshot displays the vRealize Automation Code Stream interface. On the left is a navigation sidebar with options: Dashboards, Executions, User Operations, Pipelines, Manage (with sub-options: Endpoints, Variables, Triggers, Gerrit, Git), and Triggers. The main area shows the execution details for a pipeline named 'CICD-SmartTemplate #51'. The pipeline is in a 'COMPLETED' state. The 'Build-Publish' stage includes tasks 'Unit-Test', 'Build-App', and 'Build-Image'. The 'Development' stage includes 'Create Namespace', 'Create Secret', 'Create Service', and 'Create Deployment'. The 'Build-Image' task is selected, showing its details: Task name 'Build-Image', Type 'CI', Status 'COMPLETED' (Execution Completed), Duration '5s (09/11/2018 7:16 AM - 09/11/2018 7:16 AM)', and Execute Task set to 'Always'. The 'Result' section shows the steps executed successfully, including setting environment variables, logging into Docker, and building the image. The 'Preserved Artifacts' section shows the path '/sharedPath/pipelines/CICD-SmartTemplate/51/Build-Publish.Build-Image/artifacts/'. The 'Exports' section shows a table with one row: 'IMAGE' with value 'automation/cicd-smart-template:51'. The 'Process' section indicates 'No process results available'.

7. Для отображения панели управления конвейера, на которой можно отслеживать тенденции и ключевые показатели эффективности, выберите пункт **Панели управления > Панели управления конвейерами**.

Результаты

Поздравляем! Завершено создание конвейера, который непрерывно интегрирует код из репозитория GitHub или GitLab и выполняет развертывание образа сборки.

Следующие шаги

Дополнительные сведения см. в разделе [Дополнительные ресурсы для администраторов и разработчиков Code Stream](#).

Как автоматизировать выпуск приложения, развернутого с использованием облачного шаблона YAML в Code Stream

Разработчику требуется конвейер, который извлекает облачный шаблон автоматизации из локального экземпляра GitHub при каждом внесении изменений. Конвейер необходим для развертывания приложения

WordPress в Amazon Web Services (AWS) EC2 или центре обработки данных. Служба Code Stream вызывает облачный шаблон из конвейера и автоматизирует процесс его непрерывной интеграции и предоставления (continuous integration and continuous delivery, CI/CD) для развертывания приложения.

Для создания и запуска конвейера потребуется облачный шаблон VMware.

В качестве **источника облачного шаблона** в задаче облачного шаблона Code Stream можно выбрать одно из следующих значений.

- **Шаблон Cloud Assembly** в качестве средства управления версиями. В этом случае репозиторий GitLab или GitHub не требуется.
- **Система управления версиями**, если в качестве средства управления версиями используются GitLab или GitHub. В этом случае требуется веб-перехватчик Git, и конвейер должен запускаться с помощью этого веб-перехватчика.

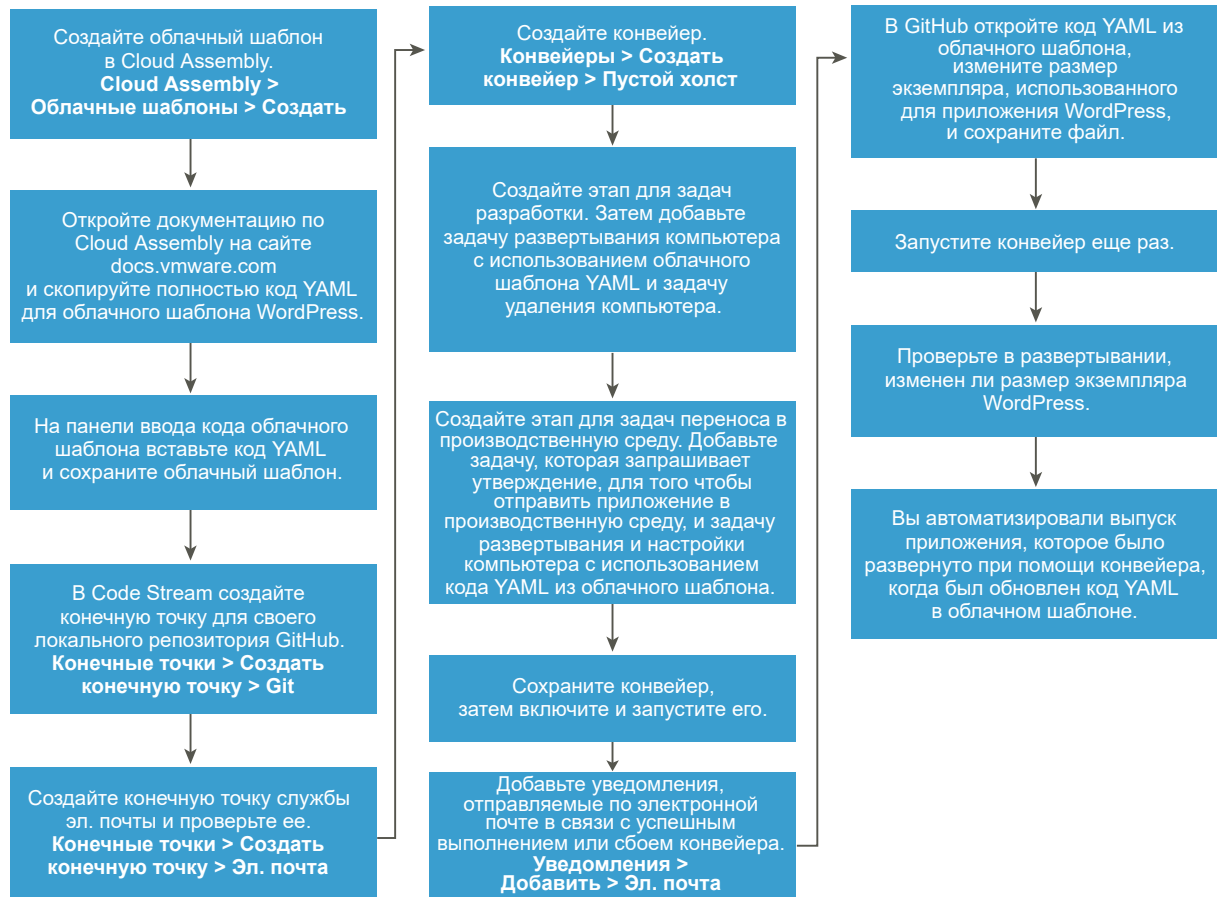
Если в репозитории GitHub есть облачный шаблон YAML, для его использования в конвейере необходимо выполнить следующие действия.

1. В Cloud Assembly отправьте облачный шаблон в репозиторий GitHub.
2. В Code Stream создайте конечную точку Git. Затем создайте веб-перехватчик Git, использующий конечную точку Git и конвейер.
3. Чтобы запустить конвейер, обновите любой файл в репозитории GitHub и сохраните изменение.

Если в репозитории GitHub нет облачного шаблона YAML и требуется использовать облачный шаблон из системы управления версиями, используйте следующую процедуру, чтобы узнать, как это сделать. Здесь показано, как создать облачный шаблон для приложения WordPress и запустить его из локального репозитория GitHub. При внесении изменений в облачный шаблон YAML запускается конвейер, который автоматизирует выпуск приложения.

- В Cloud Assembly нужно добавить облачную учетную запись, облачную зону и создать облачный шаблон.
- В Code Stream нужно добавить конечную точку для локального репозитория GitHub, в котором размещен облачный шаблон. Затем нужно добавить облачный шаблон в конвейер.

В этом примере показано, как использовать облачный шаблон из локального репозитория GitHub.

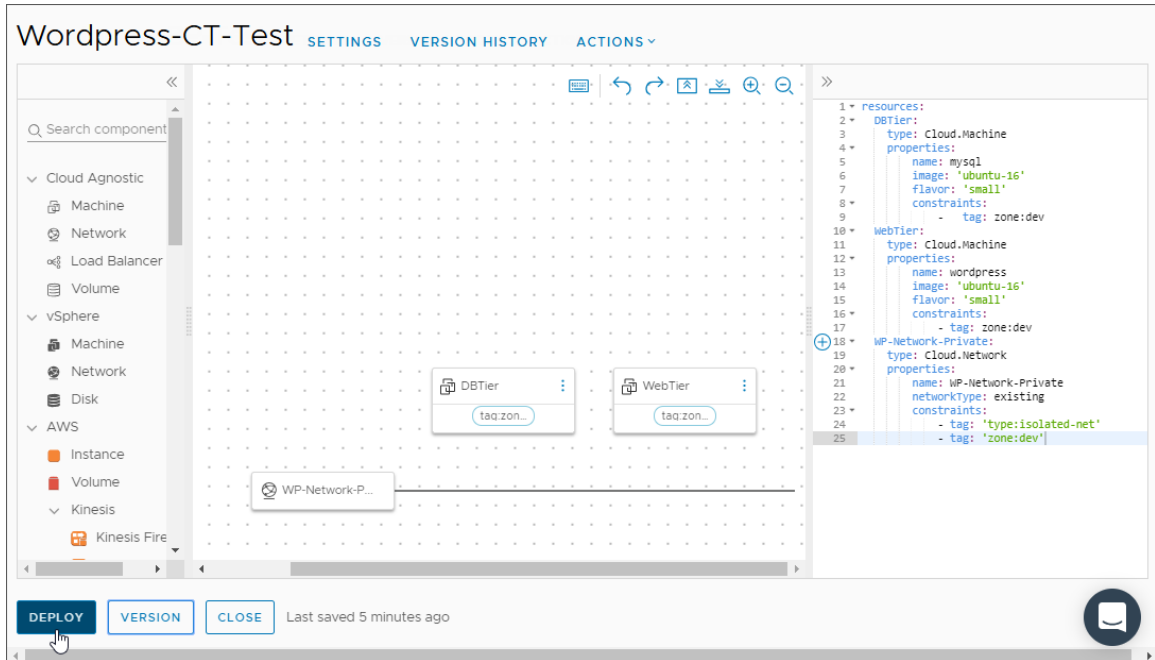


Необходимые условия

- Добавьте облачную учетную запись и облачную зону в инфраструктуру vRealize Automation Cloud Assembly. См. документацию по vRealize Automation Cloud Assembly.
- Чтобы создать облачный шаблон для выполнения следующей процедуры, скопируйте код YAML для WordPress в буфер обмена. См. код YAML облачного шаблона в примере использования WordPress в документации по vRealize Automation Cloud Assembly.
- Добавьте код YAML для приложения WordPress в экземпляр GitHub.
- Добавьте веб-перехватчик для триггера Git, чтобы конвейер мог извлекать код YAML каждый раз, когда разработчик вносит в него изменения. В Code Stream выберите параметр **Триггеры > Git > Веб-перехватчики для Git**.
- Для работы с задачей облачного шаблона пользователю должна быть присвоена любая роль в Cloud Assembly.

Процедура

1. В Cloud Assembly выполните следующие действия.
 - а) Нажмите **VMware Cloud Templates**, затем создайте облачный шаблон и развертывание для приложения WordPress.
 - б) Вставьте код YAML для WordPress, скопированный в буфер обмена, в облачный шаблон и выполните его развертывание.



2. В Code Stream создайте конечные точки.

- a) Создайте конечную точку Git для локального репозитория GitHub, в котором находится файл YAML.
- б) Добавьте конечную точку электронной почты, которая может отправлять пользователям уведомления о состоянии работы запущенного конвейера.

Add Endpoint

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

CREATE

VALIDATE

CANCEL

3. Создайте конвейер и настройте отправку уведомлений при успешном или неудачном завершении его работы.

Notification

Send notification type

☒ Email ☐ Ticket ☐ Webhook

When pipeline

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

Email server ⓘ *

--Select Email server-- ▾

Send Email

To ⓘ \$ *

Email IDs of recipients

Subject \$ *

Email Subject

Body ⓘ \$ *

1

CANCEL

SAVE

4. Добавьте этап разработки и задачу облачного шаблона.

- а) Добавьте задачу облачного шаблона, чтобы выполнить развертывание компьютера, и настройте в ней применение файла YAML облачного шаблона для приложения 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'
```

- б) Добавьте задачу облачного шаблона, чтобы уничтожить компьютер и освободить ресурсы.

5. Добавьте производственный этап, включающий задачи утверждения и развертывания.

- а) Добавьте задачу «Пользовательская операция», чтобы настроить получение запроса на утверждение отправки приложения WordPress в производственную среду.
- б) Добавьте задачу облачного шаблона, чтобы развернуть компьютер и настроить его с помощью файла YAML облачного шаблона для приложения WordPress.

При выборе команды **Создать** имя развертывания должно быть уникальным. Если оставить это поле пустым, Code Stream назначает ему случайное уникальное имя.

Вот некоторые важные сведения об использовании функции **Откат** в вашей задаче. Если вы выбрали действие **Откат**, значение в поле **Версия отката** необходимо указывать в формате **n-X**. Например, **n-1**, **n-2**, **n-3** и т. д. При создании и обновлении развертывания в любом расположении, кроме Code Stream, откат будет разрешен.

Когда пользователь входит в службу Code Stream, она получает маркер пользователя, действительный в течение 30 минут. Для длительных периодов работы конвейера, когда выполнение задачи, предшествующей задаче облачного шаблона, занимает 30 минут и более, истекает срок действия маркера пользователя. В результате задача облачного шаблона завершается ошибкой.

Для того чтобы конвейер мог выполняться более 30 минут, можно ввести дополнительный маркер API-интерфейса. Когда Code Stream вызывает облачный шаблон, маркер API-интерфейса сохраняется, а задача облачного шаблона продолжает использовать маркер API-интерфейса.

При использовании маркера API-интерфейса в качестве переменной он шифруется. В противном случае он используется в виде обычного текста.

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

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

6. Запустите конвейер.

Чтобы убедиться, что каждая задача выполнена успешно, щелкните выполняемую задачу и проверьте состояние в сведениях о развертывании, где доступна подробная информация о ресурсах.

7. В GitHub измените конфигурацию ресурсов экземпляра сервера WordPress с `small` на `medium`.

При внесении изменений запускается конвейер. Он извлекает обновленный код из репозитория GitHub и выполняет сборку приложения.

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

8. Снова запустите конвейер и убедитесь, что он работает нормально и что конфигурация ресурсов экземпляра WordPress изменилась с `малый` на `средний`.

Результаты

Поздравляем! Автоматизация выпуска приложения, развернутого с использованием облачного шаблона YAML, завершена.

Следующие шаги

Дополнительные сведения об использовании службы Code Stream см. в разделе [Глава 5 Учебники по использованию Code Stream](#).

Ссылки на дополнительные ресурсы см. в разделе [Дополнительные ресурсы для администраторов и разработчиков Code Stream](#).

Автоматизация выпуска приложения в кластере Kubernetes с помощью Code Stream

Администратор или разработчик Code Stream могут использовать Code Stream и решение VMware Tanzu Kubernetes Grid Integrated Edition (ранее известное как VMware Enterprise PKS) для автоматизации развертывания программных приложений в кластере Kubernetes. В этом примере упоминаются другие методы, которые также можно использовать для автоматизации выпуска приложения.

В данном примере создается конвейер, состоящий из двух этапов и использующий Jenkins для сборки и развертывания приложения.

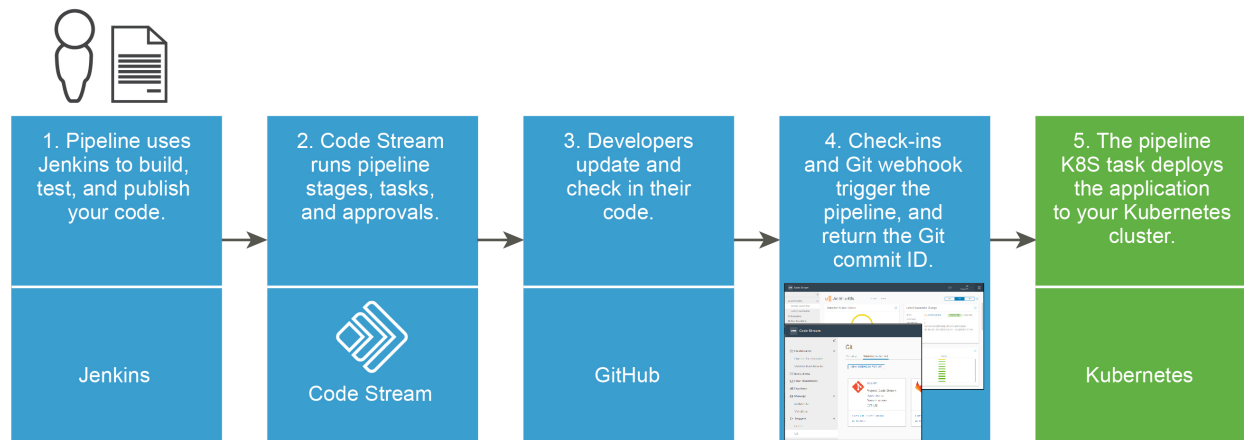
- Первый этап — разработка. На этом этапе Jenkins извлекает код из ветви репозитория GitHub, а затем выполняет его сборку, тестирование и публикацию.
- Второй этап — развертывание. В рамках этого этапа запускается задача по выполнению пользовательского действия: прежде чем конвейер сможет выполнить развертывание приложения в кластере Kubernetes, потребуется утверждение со стороны ключевых пользователей.

При использовании конечной точки API-интерфейса Kubernetes в рабочей области конвейера Code Stream создает необходимые для выполнения задачи непрерывной интеграции или настраиваемой задачи ресурсы Kubernetes, такие как ConfigMap, Secret и Pod. Code Stream обменивается данными с контейнером с помощью порта узла.

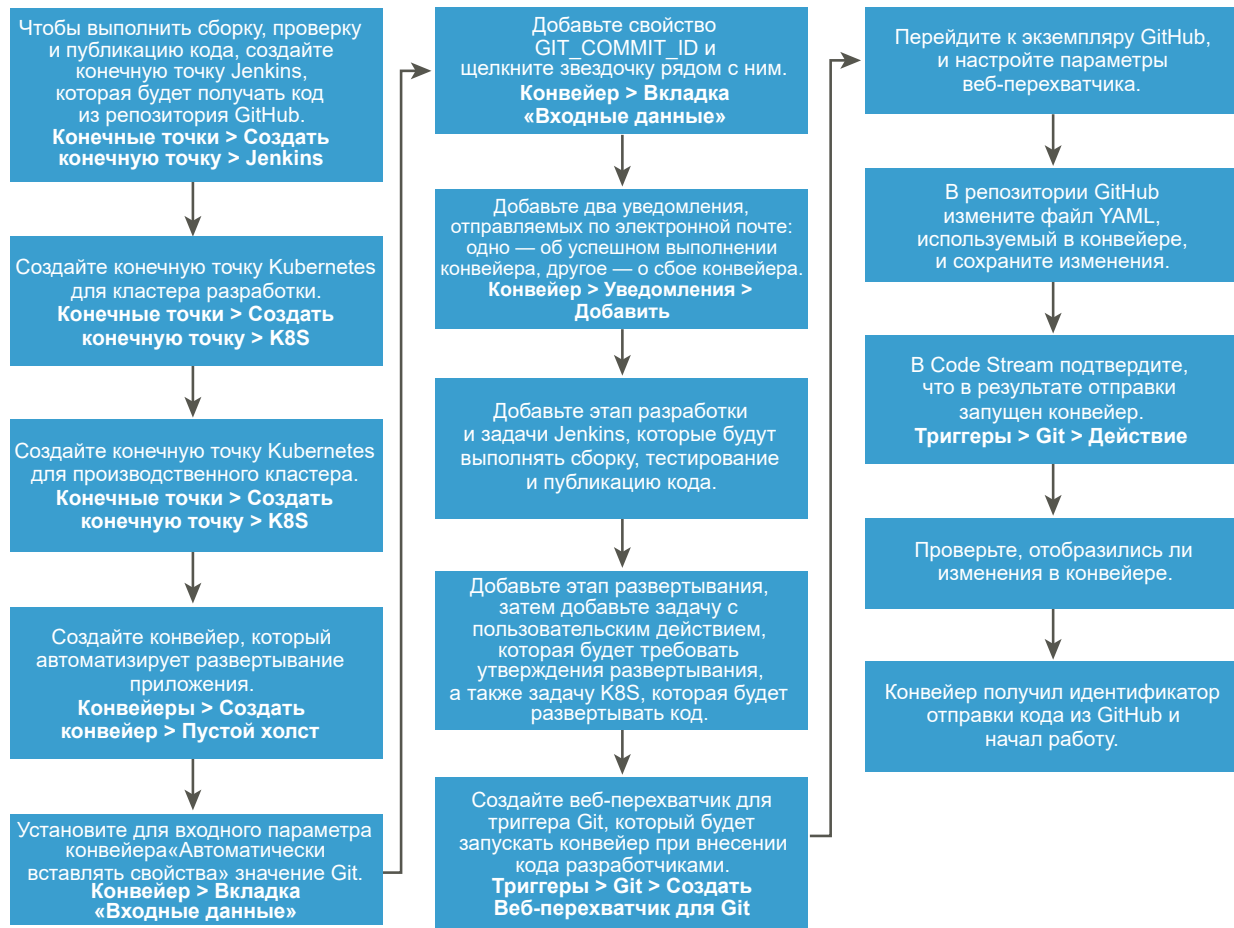
Чтобы предоставить общий доступ к данным для всех циклов выполнения конвейера, необходимо указать запрос постоянного тома. Code Stream подключит запрос постоянного тома к контейнеру, чтобы обеспечить хранение данных и его использование для последующих циклов выполнения конвейера.

Рабочая область конвейера Code Stream поддерживает Docker и Kubernetes для выполнения задач непрерывной интеграции и настраиваемых задач.

Дополнительные сведения о настройке рабочей области см. в разделе [Настройка рабочей области конвейера](#).



Чтобы конвейер мог выполнить сборку, тестирование, публикацию и развертывание приложения, должны быть доступны средства разработки, экземпляры развертывания и файл YAML. Конвейер выполнит развертывание приложения в кластерах разработки и производства Kubernetes в службе AWS.



Другие методы, позволяющие автоматизировать выпуск приложения.

- Вместо сборки приложения с помощью Jenkins можно использовать встроенные возможности сборки Code Stream и узел сборки Docker.
- Вместо развертывания приложения в кластере Kubernetes можно развернуть его в кластере Amazon Web Services (AWS).

Дополнительные сведения об использовании встроенных средств сборки Code Stream и узла Docker см. в разделе

- Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream перед использованием смарт-шаблона конвейера
- Планирование собственной сборки по модели непрерывной интеграции и предоставления в Code Stream перед добавлением задач вручную

Необходимые условия

- Убедитесь, что развертываемый код приложения находится в работоспособном репозитории GitHub.
- Убедитесь в наличии работоспособного экземпляра Jenkins.
- Убедитесь в наличии работоспособного почтового сервера.

- В Code Stream создайте конечную точку службы электронной почты для подключения к почтовому серверу.
- Настройте два кластера Kubernetes (для разработки и производственной среды) в службе Amazon Web Services (AWS), в которые конвейер будет развертывать приложение.
- Убедитесь, что репозиторий GitHub содержит либо код YAML для конвейера, либо файл YAML, в котором определены метаданные и характеристики среды.

Процедура

1. В Code Stream нажмите **Конечные точки > Создать конечную точку** и создайте конечную точку Jenkins, которая будет использоваться в конвейере для извлечения кода из репозитория GitHub.
2. Для создания конечных точек Kubernetes нажмите **Новая конечная точка**.
 - а) Создайте конечную точку для кластера разработки Kubernetes.
 - б) Создайте конечную точку для производственного кластера Kubernetes.

URL-адрес кластера Kubernetes может включать или не включать в себя номер порта.

Например:

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

```
https://api.kubernetesserver.fa2c1d78-9f00-4e30-8268-4ab81862080d.k8s-user.com
```
3. Создайте конвейер, выполняющий развертывание контейнера приложения, например Wordpress, в кластере разработки Kubernetes и задайте входные свойства конвейера.
 - а) Чтобы конвейер мог определять события отправки кода в GitHub, которые будут запускать его работу, щелкните вкладку **Входные данные** в конвейере и выберите **Автоматически вставлять свойства**.
 - б) Добавьте свойство с именем **GIT_COMMIT_ID** и щелкните звездочку рядом с ним.

При выполнении конвейера будет отображаться идентификатор отправки кода, возвращенный триггером Git.

The screenshot displays the Jenkins-K8s pipeline configuration. The left pane shows a pipeline graph with stages 'Dev' (Build-DemoApp, Test-DemoApp, Publish-DemoApp) and 'Deploy' (Approve-Deployment, tpm-K8s-AWS). The right pane shows the 'Input' tab for 'Pipeline Input Parameters' with a table of properties.

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

8 input parameters

4. Добавьте уведомления, чтобы настроить отправку сообщений электронной почты при успешном или неудачном завершении работы конвейера.

- а) В конвейере щелкните вкладку **Уведомления** и нажмите **Добавить**.
- б) Чтобы настроить отправку уведомления по электронной почте при завершении работы конвейера, выберите пункт **Электронная почта**, а затем **Завершение**. Затем выберите почтовый сервер, введите адреса электронной почты и нажмите **Сохранить**.
- в) Чтобы настроить отправку уведомления по электронной почте при сбое конвейера, выберите пункт **Сбой** и нажмите **Сохранить**.

Notification

Send notification type ☒ Email ☐ Ticket ☐ Webhook

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

Email server ⓘ * --Select Email server-- ▾

Send Email

To ⓘ \$ * Email IDs of recipients

Subject \$ * Email Subject

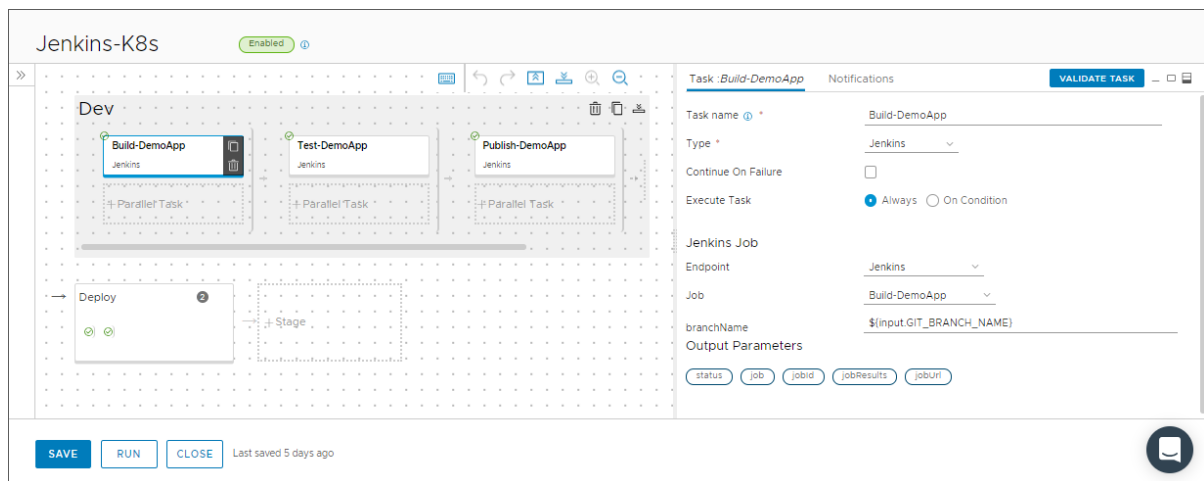
Body ⓘ \$ *

1

CANCEL SAVE

5. Добавьте в конвейер этап разработки, а затем задачи по сборке, тестированию и публикации приложения. Затем проверьте каждую задачу.

- а) Чтобы выполнить сборку приложения, добавьте задачу Jenkins, в которой используется конечная точка Jenkins и запускается задание сборки с сервера Jenkins. Затем, чтобы конвейер мог извлечь код, укажите ветвь Git в следующем формате: `${input.GIT_BRANCH_NAME}`
- б) Чтобы выполнить тестирование приложения, добавьте задачу Jenkins, использующую ту же конечную точку Jenkins и запускающую задание тестирования с сервера Jenkins. Затем введите ту же ветвь Git.
- в) Чтобы выполнить публикацию приложения, добавьте задачу Jenkins, использующую ту же конечную точку Jenkins и запускающую задание публикации с сервера Jenkins. Затем введите ту же ветвь Git.



6. Добавьте в конвейер этап развертывания, затем добавьте в этот этап две задачи: запрос на утверждение развертывания приложения и выполнение развертывания в кластере Kubernetes. Затем проверьте каждую задачу.
 - а) Чтобы настроить отправку запроса на утверждение развертывания приложения, добавьте задачу «Пользовательская операция», укажите адреса электронной почты утверждающих лиц и введите текст сообщения. Затем установите флажок **Отправить по электронной почте**.
 - б) Чтобы выполнить развертывание приложения, добавьте задачу Kubernetes. Затем в свойствах задачи Kubernetes выберите кластер разработки Kubernetes, выберите действие **Создать** и укажите источник полезных данных **Локальное определение**. Затем выберите локальный файл YAML.

7. Добавьте веб-перехватчик Git, который позволит Code Stream использовать триггер Git, запускающий работу конвейера после отправки кода разработчиками.

The screenshot shows the 'Git' configuration page with the 'Webhooks for Git' tab selected. The form includes the following fields and controls:

- Webhook URL**: `https://...vmware.com/pipeline/api/git-webhook-listeners/d4c4b02804780`
- Project**: Code Stream
- Name**: muser-Demo-WH
- Description**: (empty text area)
- Endpoint**: tpm-GitHub
- Branch**: master
- Secret token**: (masked with dots) and a **GENERATE** button.
- File**: (empty text area)
- Inclusions**: --Select-- dropdown and Value field with a plus icon.
- Exclusions**: --Select-- dropdown and Value field with a plus icon.
- Prioritize Exclusion**: Toggle switch (currently off).
- Trigger**:
 - For Git**: Radio buttons for **PUSH** (selected) and **PULL REQUEST**.
 - API token**: (masked with dots), a red error icon, and **CREATE VARIABLE** and **GENERATE TOKEN** buttons.
- Pipeline**: Jenkins-K8s (with a close icon).
- Comments**: (empty text area).

8. Чтобы проверить работу конвейера, перейдите в репозиторий GitHub, обновите файл YAML приложения и сохраните изменения.
- Убедитесь, что отправка зафиксирована в Code Stream.
 - Щелкните **Триггеры > Git > Действие**.
 - Найдите триггер конвейера.
 - Выберите параметр **Панели управления > Панели управления конвейерами**.
 - На панели управления конвейером найдите `GIT_COMMIT_ID` в области последних успешных изменений.
9. Проверьте код в конвейере и убедитесь, что изменения отображаются.

Результаты

Поздравляем! Автоматизация развертывания программного приложения в кластере Kubernetes завершена.

Пример. Пример кода YAML конвейера, выполняющего развертывание приложения в кластере Kubernetes

В конвейерах, аналогичных приведенному в данному примере, код YAML выглядит следующим образом:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
eyJzeWlwaG9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJlc2VybmFtZSI6InRhbmdvLWJldGEyIiwicGFzc3dvcmQI Oi
JhRGstcmVOLWlUQi1IejciLCJlbWFnYmCI6InRhbmdvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoizEdGdVoyOHRZbVYw
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

```

Следующие шаги

Чтобы провести развертывание программного приложения в производственном кластере Kubernetes, выполните приведенные выше шаги снова, на этот раз выбрав производственный кластер.

Дополнительные сведения об интеграции Jenkins и Code Stream см. в разделе [Интеграция Jenkins со службой Code Stream](#).

Как развернуть приложение Code Stream в среде Blue-Green

Blue-Green — это модель развертывания, которая подразумевает использование двух узлов Docker, идентичным образом развертываемых и настраиваемых в кластере Kubernetes. Использование модели Blue-Green позволяет уменьшить время простоев в среде, возникающих при развертывании приложений с помощью конвейеров Code Stream.

Экземпляры Blue и Green в модели развертывания выполняют разные функции. В каждый момент только один экземпляр принимает трафик в реальном времени, связанный с развертыванием приложения, при этом каждый из экземпляров принимает трафик в определенное время. Экземпляр Blue принимает первую версию приложения, экземпляр Green — вторую.

Подсистема балансировки нагрузки в среде Blue-Green определяет, по какому маршруту направляется трафик в реальном времени при развертывании приложения. При использовании развертывания Blue-Green сохраняется работоспособность среды, пользователи не сталкиваются с простоями, в то время как конвейер выполняет непрерывную интеграцию и развертывание приложения в производственную среду.

Конвейер, создаваемый в службе Code Stream, представляет собой развертывание Blue-Green, состоящее из двух этапов. Первый этап — разработка, второй — производство.

Рабочая область конвейера Code Stream поддерживает Docker и Kubernetes для выполнения задач непрерывной интеграции и настраиваемых задач.

Сведения о настройке рабочей области см. в разделе [Настройка рабочей области конвейера](#).

Таблица 5-2. Задачи этапа разработки для развертывания Blue-Green

Тип задач	Задача
Kubernetes	Создайте пространство имен для развертывания Blue-Green.
Kubernetes	Создайте секретный ключ для центра Docker.
Kubernetes	Создайте службу, используемую для развертывания приложения.
Kubernetes	Создайте развертывание Blue.
Опрос	Проверьте развертывание Blue.
Kubernetes	Удалите пространство имен.

Таблица 5-3. Задачи производственного этапа для развертывания Blue-Green

Тип задач	Задача
Kubernetes	Экземпляр Green получает служебные данные от экземпляра Blue.
Kubernetes	Получите сведения о наборе реплик Green.
Kubernetes	Создайте развертывание Green и используйте секретный ключ для получения образа контейнера.
Kubernetes	Обновите службу.
Опрос	Убедитесь, что развертывание успешно выполнено, открыв URL-адрес производственной среды.
Kubernetes	Завершите развертывание Blue.
Kubernetes	Удалите развертывание Blue.

Чтобы развернуть приложение в собственном развертывании Blue-Green, создайте в Code Stream конвейер из двух этапов. Первый этап включает в себя задачи, обеспечивающие развертывание приложения в экземпляре Blue, второй — задачи, обеспечивающие развертывание приложения в экземпляре Green.

Конвейер можно создать с помощью смарт-шаблона конвейера CI/CD. При использовании шаблона в конвейер автоматически добавляются этапы и задачи. Шаблон также включает в себя выбранные параметры развертывания.

Если конвейер создается вручную, нужно самостоятельно спланировать этапы работы конвейера. См. пример в разделе [Планирование собственной сборки по модели непрерывной интеграции и предоставления в Code Stream перед добавлением задач вручную](#).

В этом примере для создания конвейера Blue-Green используется смарт-шаблон конвейера CI/CD.

Необходимые условия

- Убедитесь в наличии доступа к работоспособному кластеру Kubernetes в службе AWS.
- Убедитесь в наличии настроенного развертывания Blue-Green и в том, что оба экземпляра (Blue и Green) идентичны.

- Создайте в Code Stream конечную точку Kubernetes, с помощью которой будет осуществляться развертывание образа приложения в кластере Kubernetes службы AWS.
- Ознакомьтесь с использованием смарт-шаблона конвейера CI/CD. См. раздел [Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream](#) перед использованием смарт-шаблона конвейера.

Процедура

1. Выберите пункт **Конвейеры > Новый конвейер > Смарт-шаблон > Шаблон CI/CD**.
2. Введите сведения о непрерывной интеграции (CI) для смарт-шаблона конвейера CI/CD и нажмите **Далее**.

Для справки см. раздел [Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream](#) перед использованием смарт-шаблона конвейера.

3. Введите сведения о непрерывном предоставлении (CD) для смарт-шаблона конвейера.
 - а) Выберите среды для развертывания приложения. Например: **Разраб.** и **Прозв..**
 - б) Выберите службу, которую конвейер будет использовать для развертывания.
 - в) В области «Развертывание» выберите конечную точку кластера для среды разработки и производственной среды.
 - г) В качестве модели производственного развертывания выберите **Развертывание Blue-Green**, затем нажмите **Создать**.

Smart Template: CI/CD

Step 2 of 2

Environment * ☒ Dev ☒ Prod

K8s YAML files * SELECT PROCESS

Processed files: codestream.yaml

Select service

Deployment name	Service	Namespace	Image
codestream-demo	codestream-demo	codestream	https://codestream/Myapp

1 services

Deployment


Environment	Cluster Endpoint	Namespace
Dev	Dev-AWS-Cluster	codestream-139606
Prod	Prod-AWS-Cluster	codestream

Prod deployment model * ☐ Canary ☐ Rolling Upgrade ☒ Blue-Green

Rollback strategy ☐

Health check URL *

CREATE BACK CANCEL



Результаты

Поздравляем! С использованием смарт-шаблона конвейера создан конвейер, выполняющий развертывание приложения в экземплярах Blue-Green производственного кластера Kubernetes в службе AWS.

Пример. Пример кода YAML для некоторых задач развертывания Blue-Green

Код YAML, присутствующий в задачах конвейера Kubernetes для развертывания Blue-Green, может выглядеть аналогично приведенным ниже примерам, которые создают пространство имен, службу и развертывание. Если необходимо загрузить образ из частного репозитория, файл YAML должен содержать раздел с секретом конфигурации Docker. См. подраздел о непрерывном предоставлении в разделе [Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream](#) перед использованием смарт-шаблона конвейера.

Создав конвейер на основе смарт-шаблона конвейера, можно модифицировать задачи в соответствии со спецификой конкретного развертывания.

Пример кода YAML для создания пространства имен:

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

Пример кода YAML для создания службы:

```
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
```

Пример кода YAML для создания развертывания:

```
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
```

Следующие шаги

Дополнительные сведения об использовании службы Code Stream см. в разделе [Глава 5 Учебники по использованию Code Stream](#).

Чтобы выполнить откат развертывания, см. раздел [Как откатить развертывание в Code Stream](#).

Ссылки на дополнительные ресурсы см. в разделе [Дополнительные ресурсы для администраторов и разработчиков Code Stream](#).

Интеграция собственных средств сборки, тестирования и развертывания со службой Code Stream

Пользователь с правами разработчика или администратора DevOps может создавать настраиваемые сценарии, расширяющие возможности Code Stream.

С помощью таких сценариев службу Code Stream можно интегрировать с собственными средствами непрерывной интеграции (Continuous Integration, CI) и непрерывного предоставления (Continuous Delivery, CD), а также API-интерфейсами для сборки, тестирования и развертывания приложений. Настраиваемые сценарии особенно полезны в случаях, когда API-интерфейсы приложений не размещаются в общем доступе.

Настраиваемый сценарий может делать практически все, что требуется для интеграции инструментов сборки, тестирования и развертывания с Code Stream. Например, сценарий может взаимодействовать с рабочей областью конвейера для поддержки задач непрерывной интеграции, которые обеспечивают сборку и тестирование приложения, а также задач непрерывного предоставления, которые обеспечивают развертывание приложения. Сценарий можно отправлять сообщения в Slack по завершении работы конвейера, а также выполнять большое количество других функций.

Рабочая область конвейера Code Stream поддерживает Docker и Kubernetes для выполнения задач непрерывной интеграции и настраиваемых задач.

Дополнительные сведения о настройке рабочей области см. в разделе [Настройка рабочей области конвейера](#).

Настраиваемый сценарий должен быть написан на одном из поддерживаемых языков. В сценарии прописывается бизнес-логика и определяются входные и выходные данные. К выходным относятся, помимо прочего, такие типы данных, как число, строка, текст и пароль. Можно создать несколько версий настраиваемого сценария с разной бизнес-логикой, входными и выходными данными.

Версия сценария выполняется в рамках настраиваемой задачи конвейера. Созданные сценарии хранятся в экземпляре Code Stream.

Если в конвейере используется настраиваемая интеграция, то при попытке ее удаления появляется сообщение об ошибке, указывающее на то, что ее нельзя удалить.

При удалении настраиваемой интеграции удаляются все версии настраиваемого сценария. Если в существующем конвейере есть настраиваемая задача, в которой применяется какая-либо версия этого сценария, выполнение этого конвейера завершится сбоем. Чтобы выполнение существующих конвейеров не завершалось сбоем, неиспользуемую версию сценария можно обозначить как устаревшую или отозвать. Если эта версия не используется ни одним из конвейеров, ее можно удалить.

Таблица 5-4. После написания настраиваемого сценария

Действия...	Сведения о действии...
Добавьте настраиваемую задачу в конвейер.	<p>Настраиваемая задача:</p> <ul style="list-style-type: none"> ■ выполняется в том же контейнере, что и другие задачи CI в конвейере; ■ содержит входные и выходные переменные, значения которых заполняются сценарием перед тем, как конвейер запустит выполнение настраиваемой задачи; ■ поддерживает несколько типов данных и различные типы метаданных, определяемых в качестве входных и выходных данных в сценарии.
Выберите сценарий в настраиваемой задаче.	Объявите входные и выходные свойства в сценарии.
Сохраните конвейер, затем включите и запустите его.	В ходе работы конвейера настраиваемая задача вызывает указанную версию сценария и запускает его бизнес-логику. В результате ваше средство сборки, тестирования и развертывания интегрируется со службой Code Stream.
После завершения работы конвейера просмотрите сведения о выполнении задач и этапов.	Убедитесь, что результаты работы конвейера соответствуют ожиданиям.

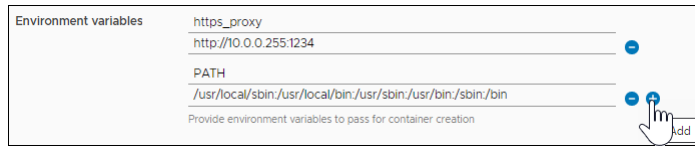
При использовании настраиваемой задачи, которая вызывает версию настраиваемой интеграции, можно добавить настраиваемые переменные среды в качестве пар «имя — значение» на вкладке **Рабочая область** конвейера. Когда образ построителя создает контейнер рабочей области, который выполняет задачу непрерывной интеграции (CI) и развертывает образ, Code Stream передает переменные среды в этот контейнер.

Например, если для экземпляра Code Stream требуется веб-прокси, а для создания контейнера для настраиваемой интеграции используется узел Docker, Code Stream запускает конвейер и передает переменные настроек веб-прокси в этот контейнер.

Таблица 5-5. Пример пар «имя — значение» для переменных среды

Имя	Значение
HTTPS_PROXY	http://10.0.0.255:1234
https_proxy	http://10.0.0.255:1234
NO_PROXY	10.0.0.32, *.dept.vsphere.local
no_proxy	10.0.0.32, *.dept.vsphere.local
HTTP_PROXY	http://10.0.0.254:1234
http_proxy	http://10.0.0.254:1234
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

В пользовательском интерфейсе отобразятся пары «имя — значение», например следующие.



В этом примере создается настраиваемая интеграция, которая связывает службу Code Stream с экземпляром Slack и публикует сообщения в канале Slack.

Необходимые условия

- Чтобы написать настраиваемый сценарий, необходимо использовать язык Python 2, Python 3 или Node.js либо один из языков командных оболочек (Bash, sh или zsh).
- Создайте образ контейнера с помощью установленного файла Node.js или среды выполнения Python.

Процедура

1. Создайте настраиваемую интеграцию.

- а) Выберите пункт **Настраиваемые интеграции > Создать** и введите соответствующее имя.
- б) Выберите предпочитаемую среду выполнения.
- в) Щелкните **Создать**.

Сценарий откроется, и в нем отобразится код, включая нужную среду выполнения. Например, `runtime: "nodejs"`. Сценарий должен включать в себя среду выполнения, которую использует образ построителя, чтобы добавленная в конвейер настраиваемая задача была успешно выполнена в ходе работы конвейера. В противном случае настраиваемая задача завершится сбоем.

Основные области файла YAML настраиваемой интеграции включают в себя среду выполнения, код, входные и выходные свойства. В этой процедуре описываются различные типы и синтаксис.

Ключи YAML настраиваемой интеграции	Описание
runtime	<p>Среда выполнения задач, где Code Stream выполняет код, который может представлять собой одну из следующих строк, нечувствительных к регистру.</p> <ul style="list-style-type: none"> ■ nodejs ■ python2 ■ python3 ■ shell <p>Если ничего не указано, по умолчанию используется shell.</p>
code	Настраиваемая бизнес-логика для выполнения в рамках настраиваемой задачи.
inputProperties	Массив входных свойств для сбора данных в ходе определения параметров настраиваемой задачи. Эти свойства обычно используются в коде.
outputProperties	Массив выходных свойств, которые можно экспортировать из настраиваемой задачи для передачи в конвейер.

2. Объявите входные свойства в сценарии, используя доступные типы данных и метаданных.

Входные свойства передаются в качестве контекста для сценария в разделе `code:` файла YAML.

Входные ключи YAML настраиваемых задач	Описание	Обязательный
type	Типы входных данных для отображения: <ul style="list-style-type: none"> ■ text ■ textarea ■ number ■ checkbox ■ password ■ select 	Да
name	Имя или строка входных данных для настраиваемой задачи, которые будут вставлены в код YAML настраиваемой интеграции. Значение должно быть уникальным для каждого входного свойства, определенного для настраиваемой интеграции.	Да
title	Метка текстовых строк входного свойства для настраиваемой задачи на холсте модели конвейера. Если оставить это поле пустым, по умолчанию используется name .	Нет
required	Определяет, должен ли пользователь вводить входное свойство при настройке задачи. Задайте значение true или false . Если задано значение true и пользователь не вводит значение при настройке задачи на холсте конвейера, задача остается в состоянии «Не настроено».	Нет
placeholder	Текст по умолчанию для области ввода входного свойства, если значение отсутствует. Сопоставляется с атрибутом html-заполнителя . Поддерживается только для некоторых типов входных свойств.	Нет
defaultValue	Значение по умолчанию, подставляемое в область ввода входного свойства при отображении настраиваемой задачи на странице модели конвейера.	Нет
bindable	Определяет, принимает ли входное свойство переменные со знаком доллара при моделировании настраиваемой задачи на холсте конвейера. Добавляется индикатор \$ рядом с заголовком. Поддерживается только для некоторых типов входных свойств.	Нет
labelMessage	Строка, которая выступает в качестве всплывающей подсказки для пользователей. Добавляет значок всплывающей подсказки i рядом с заголовком входных данных.	Нет

Входные ключи YAML настраиваемых задач	Описание	Обязательный
enum	<p>Принимает массив значений, который отображает параметры входного свойства select. Поддерживается только для некоторых типов входных свойств.</p> <p>Если пользователь выбирает параметр и сохраняет его для настраиваемой задачи, значение inputProperty соответствует данному значению и отображается при моделировании настраиваемых задач.</p> <p>Например, значение 2015.</p> <ul style="list-style-type: none"> ■ 2015 ■ 2016 ■ 2017 ■ 2018 ■ 2019 ■ 2020 г. 	Нет
options	<p>Принимает массив объектов с использованием параметров optionKey и optionValue.</p> <ul style="list-style-type: none"> ■ optionKey. Значение, передаваемое в раздел кода задачи. ■ optionValue. Строка, которая отображает параметр в пользовательском интерфейсе. <p>Поддерживается только для некоторых типов входных свойств.</p> <p>Параметры</p> <p>optionKey: key1. В случае выбора и сохранения для настраиваемой задачи значение свойства inputProperty соответствует значению key1 в разделе кода.</p> <p>optionValue: «метка значения 1». Отображаемое значение для key1 в пользовательском интерфейсе, не отображается в других местах для настраиваемой задачи.</p> <p>optionKey: key2</p> <p>optionValue: «метка значения 2»</p> <p>optionKey: key3</p> <p>optionValue: «метка значения 3»</p>	Нет
minimum	<p>Принимает число, которое является минимальным допустимым значением данного входного свойства. Поддерживается только для числовых входных свойств.</p>	Нет
maximum	<p>Принимает число, которое является максимальным допустимым значением данного входного свойства. Поддерживается только для числовых входных свойств.</p>	Нет

Таблица 5-6. Поддерживаемые типы данных и метаданных для настраиваемых сценариев

Поддерживаемые типы данных	Поддерживаемые метаданные для входных данных
<ul style="list-style-type: none"> ■ Строка ■ Text (текст) ■ List: список любого типа ■ Map: сопоставление в формате map[string]any ■ Secure: отображается как текстовое поле для ввода пароля; при сохранении настраиваемой задачи к данным этого типа применяется шифрование ■ Number (число) ■ Boolean: (логический) отображается в виде текстовых полей ■ URL: аналогично String, с дополнительной проверкой ■ Selection, radio button (выбор значения, переключатель) 	<ul style="list-style-type: none"> ■ type: (тип) либо String, либо Text ■ default: значение по умолчанию ■ options: список или сопоставление параметров, которые будут использоваться для выбора значения или переключателя ■ min: минимальное значение или размер ■ max: максимальное значение или размер ■ title: подробное имя текстового поля ■ placeholder: заполнитель для пользовательского интерфейса ■ description: (описание) отображается в виде всплывающей подсказки

Например:

```
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. Объявите выходные свойства в сценарии.

Сценарий получает выходные свойства из раздела бизнес-логики `code`: сценария, в котором объявлен контекст для выходных данных.

В ходе работы конвейера можно ввести код отклика для выходных данных задачи. Например: 200.

Ключи, поддерживаемые Code Stream для каждого параметра **outputProperty**.

key	Описание
type	В настоящее время доступно одно значение — label .
name	Ключ, передаваемый блоком кода YAML настраиваемой интеграции.
title	Метка в пользовательском интерфейсе, которая отображает параметр outputProperty .

Например:

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

4. Для взаимодействия с входными и выходными данными настраиваемого сценария используйте оператор **context** для получения входного или настройки выходного свойства.

Для входного свойства: `(context.getInput("key"))`

Для выходного свойства: `(context.setOutput("key", "value"))`

Для Node.js:

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

Для Python:

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

Для оболочки:

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

5. В разделе `code`: объявите всю бизнес-логику настраиваемой интеграции.

Пример для среды выполнения 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. Прежде чем присвоить сценарию настраиваемой интеграции версию и опубликовать его, загрузите файл контекста для Python или Node.js и проверьте бизнес-логику, включенную в сценарий.
 - а) Поместите курсор в сценарий, а затем нажмите кнопку файла контекста в верхней части холста. Например, если сценарий написан на языке Python, нажмите **CONTEXT.PY**.
 - б) Внесите изменения в файл и сохраните его.
 - в) Запустите и протестируйте настраиваемый сценарий в системе разработки, используя файл контекста.
7. Назначьте сценарию настраиваемой интеграции версию.
 - а) Нажмите **Версия**.
 - б) Введите сведения о версии.
 - в) Нажмите **Версия выпуска**, чтобы сценарий можно было выбрать в настраиваемой задаче.
 - г) Чтобы создать версию, нажмите **Создать**.

Creating Version

Version * 1.0

Description New

Change Log New for 1.0

Release Version ☒

CANCEL CREATE

8. Чтобы сохранить сценарий, нажмите **Сохранить**.

9. Настройте рабочую область в конвейере.

В этом примере используется рабочая область Docker.

- а) Откройте вкладку **Рабочая область**.
- б) Выберите узел Docker и URL-адрес образа построителя.

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

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

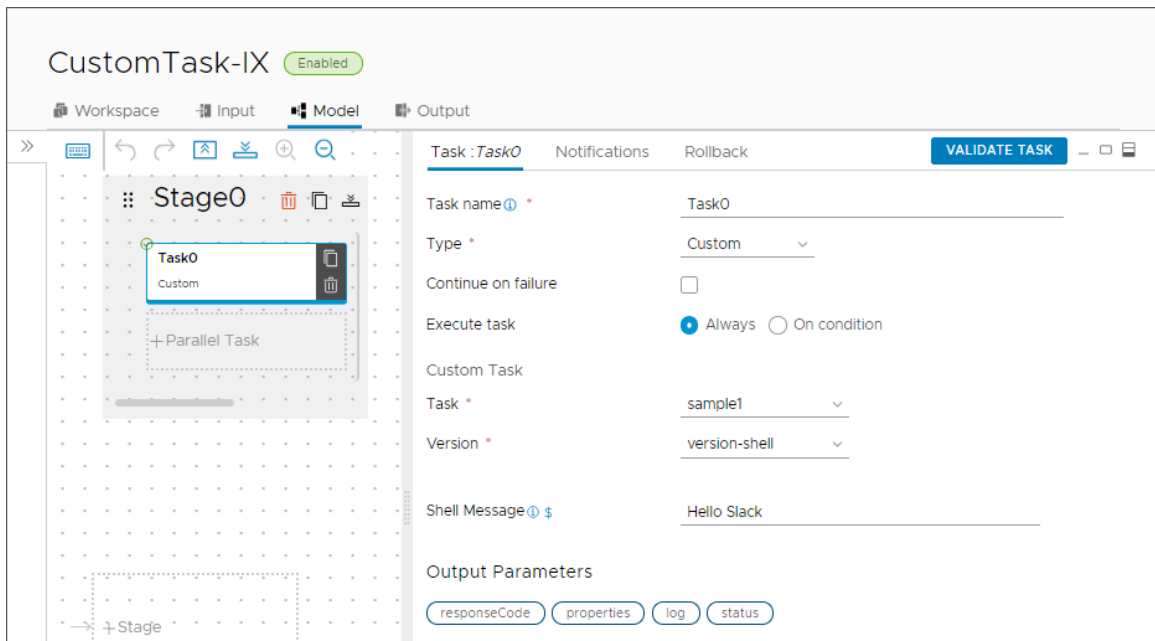
At the bottom right, there is a text box with the following message: "If this pipeline links to Git through a webhook, the pipeline triggers on Git events. For CI tasks, the linked Git repository, which receives details from the Git webhook, automatically clones the workspace."

10. Добавьте в конвейер настраиваемую задачу и настройте ее параметры.

- а) Откройте вкладку **Модель**.
- б) Добавьте задачу, выберите **Настраиваемый** в качестве типа и введите соответствующее имя.

- в) Выберите сценарий настраиваемой интеграции и его версию.
- г) Введите текст настраиваемого сообщения, которое требуется отобразить в Slack.

Введенный текст заменит `defaultValue` в сценарии настраиваемой интеграции. Например:



11. Сохраните и включите конвейер.

- а) Нажмите **Сохранить**.
- б) На вкладке «Конвейер» нажмите **Включить конвейер** (кружок должен переместиться вправо).

12. Запустите конвейер.

- а) Щелкните **Запустить**.
- б) Проверьте выполнение конвейера.

- в) Убедитесь, что данные, полученные на выходе, включают в себя ожидаемый код состояния, код отклика, состояние и объявленные выходные данные.

Компонент **statusCode** был определен как выходное свойство. Например, **statusCode** со значением 200 может указывать на успешную отправку сообщения в Slack, а **responseCode** со значением 0 — на выполнение сценария без ошибок.

- г) Чтобы проверить выходные данные, зафиксированные в журналах выполнения, нажмите **Выполняемые элементы**, щелкните ссылку на конвейер, выберите задачу и просмотрите данные журнала. Например:

The screenshot displays the execution details for a task named "Task1" within a workflow titled "custom-int-demo #5". The workflow is in a "COMPLETED" state. The task "Task1" is also in a "COMPLETED" state, with the message "Execution Completed." The task's duration is 6 seconds, occurring on 12/21/2018 at 3:04 AM. The task is configured to "Continue on failure" (unchecked) and to "Execute task" "Always". The output section shows "statusCode" as 200 and "Response code" as 0. The logs section shows a terminal snippet with the command "node -r ./context.js app.js".

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

[View Full Log](#)

13. При возникновении ошибки устраните проблему и повторно запустите конвейер.

Например, если в базовом образе отсутствует файл или модуль, необходимо создать другой базовый образ, который будет включать отсутствующий файл. Затем укажите файл Docker и запустите обработку образа в конвейере.

Результаты

Поздравляем! Создан сценарий настраиваемой интеграции, который связывает службу Code Stream с экземпляром Slack и публикует сообщения в канале Slack.

Следующие шаги

Создайте дополнительные настраиваемые интеграции, поддерживающие использование настраиваемых задач в конвейерах и расширяющие возможности службы Code Stream по автоматизации жизненного цикла выпуска программного обеспечения.

Как использовать свойства ресурса из задачи облачного шаблона в следующей задаче

При использовании задачи облачного шаблона в Code Stream часто возникает вопрос, как использовать выходные данные этой задачи в последующих задачах конвейера. Чтобы использовать выходные данные задачи облачного шаблона, например облачный компьютер, необходимо знать, как найти свойства ресурса в сведениях о развертывании задачи облачного шаблона, а также IP-адрес облачного компьютера.

Например, в сведениях о развертывании облачного шаблона VMware указаны облачный компьютер и его IP-адрес. Их можно использовать в конвейере в качестве переменной, чтобы привязать задачу облачного шаблона к задаче REST.

Это нестандартный метод поиска IP-адреса облачного компьютера, так как сведения о развертывании облачного шаблона VMware становятся доступными только после его завершения. Тогда можно использовать ресурсы из развертывания облачного шаблона VMware для привязки задач конвейера.

- Свойства ресурса, которые отображаются в задаче облачного шаблона в конвейере, определяются в облачном шаблоне VMware в Cloud Assembly.
- Пользователь может не знать, когда завершилось развертывание облачного шаблона.
- Выходные свойства облачного шаблона VMware отображаются в задаче облачного шаблона в Code Stream только после завершения развертывания.

Это может пригодиться при развертывании приложения и вызове различных API-интерфейсов. Например, если использовать задачу облачного шаблона, которая вызывает шаблон VMware для развертывания приложения Wordpress с REST API, можно найти IP-адрес развернутого компьютера в сведениях о развертывании и провести тест с помощью API-интерфейса.

Задача облачного шаблона отображает сведения об упреждающем вводе, что позволяет использовать привязку переменных. Способ привязки переменной выбирает пользователь.

В этом примере показано, как выполнить следующие действия.

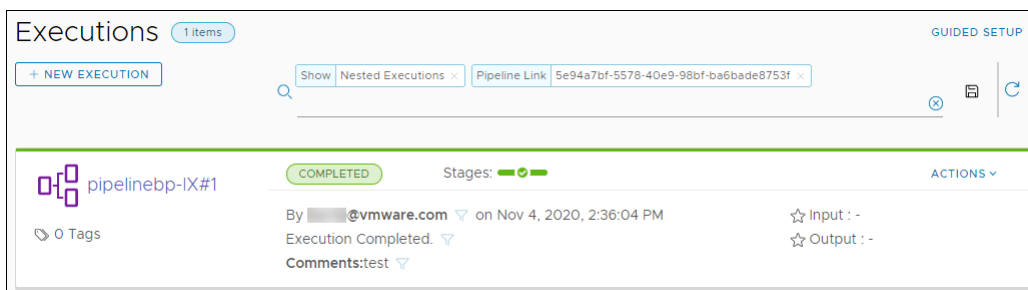
- Найдите сведения о развертывании и свойства ресурса для задачи облачного шаблона в конвейере, который был запущен и успешно выполнен.
- IP-адрес облачного компьютера находится в разделе «Ресурсы» сведений о развертывании.
- В конвейере добавьте задачу REST после задачи облачного шаблона.
- Привяжите задачу облачного шаблона к задаче REST, указав IP-адрес облачного компьютера в URL-адресе задачи REST.
- Запустите конвейер и проконтролируйте привязку задачи облачного шаблона к задаче REST.

Необходимые условия

- Убедитесь, что у вас есть работающий облачный шаблон VMware, для которого настроено управление версиями.
- Убедитесь, что облачный шаблон VMware развернут в Cloud Assembly.
- Убедитесь, что у вас есть конвейер, который включает в себя задачу облачного шаблона с этим шаблоном VMware.
- Убедитесь, что конвейер запущен и выполнен.

Процедура

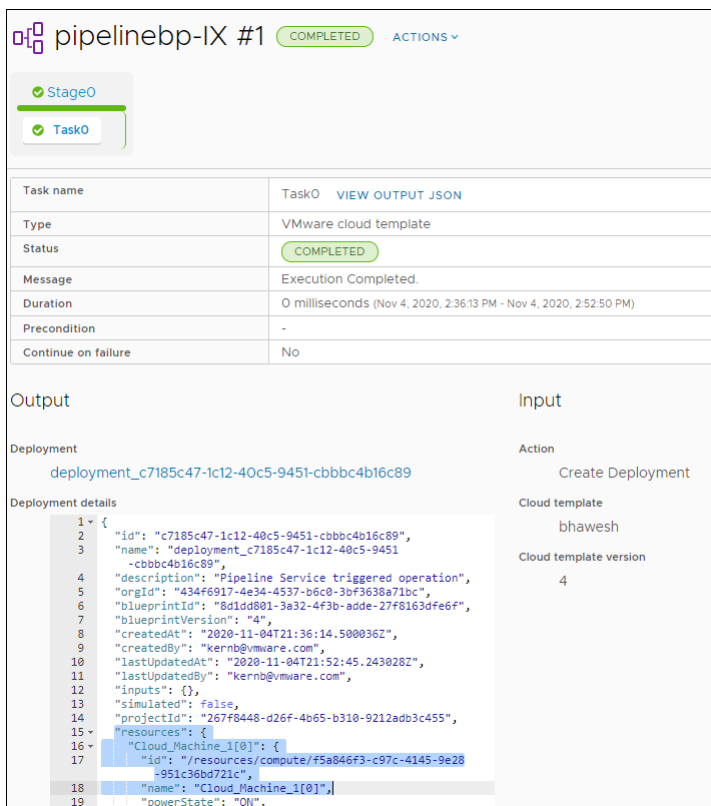
1. В конвейере найдите IP-адрес облачного компьютера. Для этого перейдите в раздел «Ресурсы» сведений о развертывании задачи облачного шаблона.
 - а) Щелкните **Действия > Просмотреть выполняемые элементы**.
 - б) При успешном выполнении конвейера щелкните ссылку на выполнение конвейера.



- в) Под именем конвейера щелкните ссылку на раздел **Задача**.

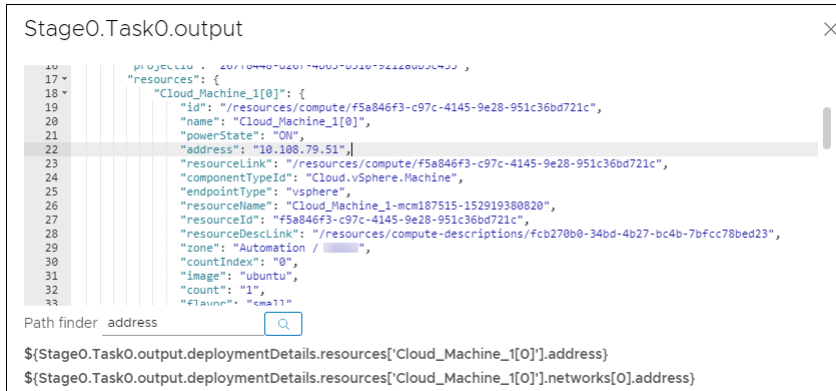


- г) В области «Выходные данные» выберите «Сведения о развертывании».



- д) В разделе сведений о разворачивании «Ресурсы» укажите имя облачного компьютера.
- В URL-адрес задачи REST будет добавлен синтаксис имени облачного компьютера.
- е) Чтобы найти выражение привязки для выходного свойства задачи облачного шаблона, щелкните **ПРОСМОТР ВЫХОДНЫХ ДАННЫХ JSON**, найдите свойство «Адрес». Там будет указан IP-адрес облачного компьютера.

Выражение привязки отображается под свойством и значком поиска в выходных данных JSON.



В поле свойства ресурса адреса отображается IP-адрес облачного компьютера. Например:

```

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

```

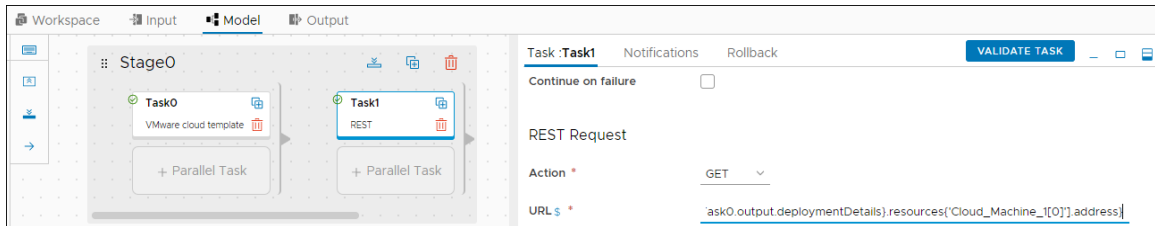
2. Вернитесь к модели конвейера и введите URL-адрес в задаче REST.

- Щелкните **Действия > Просмотреть конвейер**.
- Щелкните задачу REST.

- в) В поле URL-адреса запроса REST введите \$, выберите **Этап, Задача, Выходные данные, Сведения о развертывании** и введите **resources**.

Ввод с помощью автоматического заполнения доступен до поля ввода значения **resources**.

- г) Укажите оставшиеся ресурсы облачного компьютера в разделе сведений о развертывании следующим образом: `{ 'Cloud_Machine_1[0]' }.address`.



Для указания облачного компьютера нужно использовать нотацию в квадратных скобках, как показано выше.

Формат полного URL-адреса: \$

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

3. Запустите конвейер: задача REST использует облачный компьютер и IP-адрес из выходных данных задачи облачного шаблона в качестве URL-адреса для проверки.

Результаты

Поздравляем! Вы нашли имя и IP-адрес облачного компьютера в сведениях о развертывании и выходных данных JSON облачного шаблона и использовали их, чтобы привязать выходные данные задачи облачного шаблона к входным данным URL-адреса задачи REST в конвейере.

Следующие шаги

Попробуйте использовать переменные привязки из ресурсов в задаче с облачным шаблоном для других задач в конвейере.

Как использовать интерфейс REST API для интеграции Code Stream с другими приложениями

Code Stream содержит подключаемый модуль REST, позволяющий интегрировать Code Stream с другими приложениями, использующими интерфейс REST API. Это дает возможность непрерывно разрабатывать и предоставлять приложения, которым необходимо взаимодействовать друг с другом. Подключаемый модуль REST вызывает API-интерфейс, который осуществляет обмен данными между Code Stream и другим приложением.

Подключаемый модуль REST позволяет выполнять следующие действия.

- Интеграция внешних систем на базе интерфейса REST API в конвейер Code Stream.
- Интеграция конвейера Code Stream в процессы внешних систем.

Подключаемый модуль REST работает с любыми интерфейсами REST API и поддерживает методы GET, POST, PUT, PATCH и DELETE при обмене данными между Code Stream и другими приложениями.

Таблица 5-7. Подготовка конвейера для обмена данными по интерфейсу REST API

Действия	Результат
Добавьте задачу REST в конвейер.	Задача REST выполняет передачу данных между приложениями и может предоставлять сведения о состоянии для следующей задачи текущего этапа конвейера.
В задаче REST выберите действие REST и укажите URL-адрес.	В ходе работы конвейера задача конвейера вызывает URL-адрес. Для действий POST, PUT и PATCH необходимо добавить полезные данные. В полезных данных можно выполнить привязку свойств конвейера и задачи в ходе выполнения конвейера.
Рассмотрим пример.	Пример использования подключаемого модуля REST. Можно добавить задачу REST, которая будет создавать тег для сборки при отправке кода в Git, и настроить в параметрах задачи отправку запроса на получение идентификатора обновления кода из репозитория. Задача может отправить полезную нагрузку в репозиторий и создать тег для сборки, а репозиторий может отправить ответ, содержащий этот тег.

По аналогии с использованием подключаемого модуля REST для вызова API-интерфейса можно включить в конвейер задачу опроса, которая будет вызывать интерфейс REST API и опрашивать его, пока он не завершит работу и пока задача конвейера не будет удовлетворять выходным критериям.

Кроме того, можно использовать интерфейсы REST API для импорта и экспорта конвейера, а также применять образцы сценариев для запуска конвейера.

Эта процедура позволяет получить обычный URL-адрес.

Процедура

1. Чтобы создать конвейер, выберите **Конвейеры > Создать конвейер > Пустой холст**.
2. В текущем этапе конвейера щелкните **Новая последовательная задача**.
3. На панели задач добавьте задачу REST.

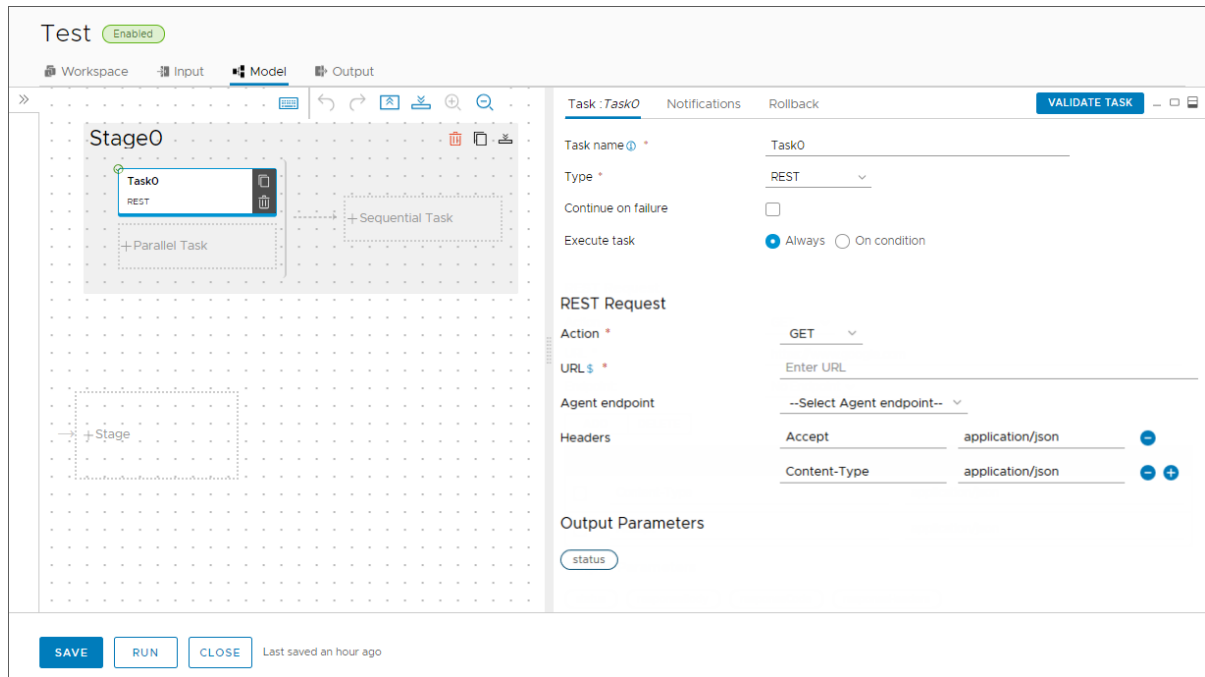
- а) Введите имя задачи.
- б) В раскрывающемся меню «Тип» выберите **REST**.
- в) В области «Запрос REST» выберите **GET**.

Чтобы получить данные запроса задачи REST из другого приложения, необходимо выбрать метод GET. Чтобы отправить данные в другое приложение, выберите метод POST.

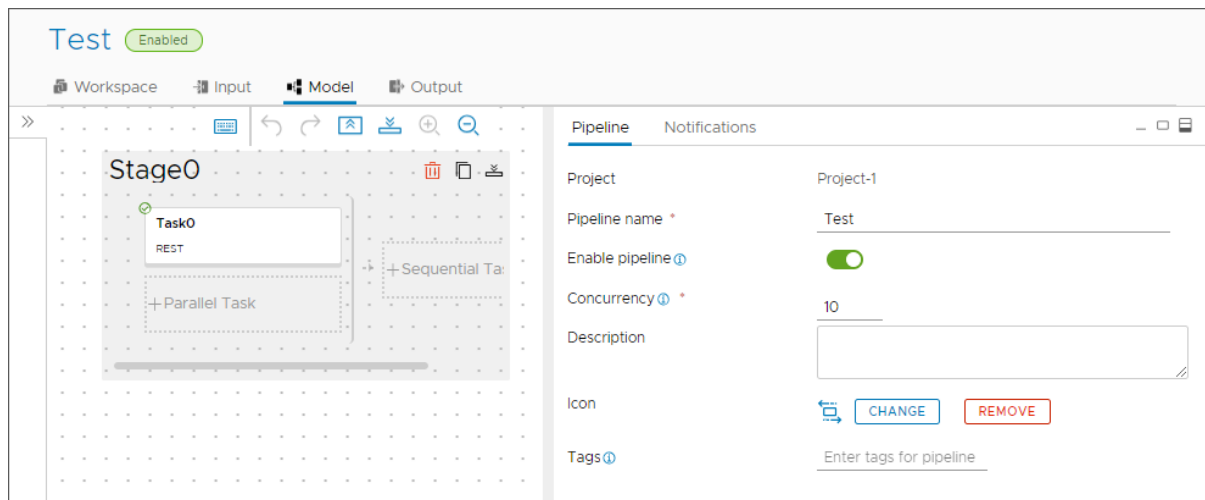
- г) Введите URL-адрес, определяющий конечную точку интерфейса REST API. Например, `https://www.google.com`.

Для того чтобы задача REST могла импортировать данные из другого приложения, можно включить в нее переменную полезной нагрузки. Например, для действия «импорт» можно ввести `${Stage0.export.responseBody}`. Если размер данных ответа превышает 5 МБ, задача REST может завершиться сбоем.

- д) Для авторизации задачи щелкните **Добавить заголовки**, введите ключ и значение заголовка.



4. Чтобы сохранить конвейер, щелкните **Сохранить**.
5. На вкладке «Конвейер» щелкните **Включить конвейер**.



6. Щелкните **Сохранить**, затем **Заккрыть**.
7. Щелкните **Запустить**.

8. Чтобы отслеживать работу конвейера, нажмите **Выполняемые элементы**.

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

Workflow	Status	Stages	Actions
Test#1	RUNNING	By system-user on 11/26/2018 3:11 PM	Input : n/a Output : n/a

9. Чтобы убедиться, что задача REST возвращает ожидаемые данные, проверьте выполнение конвейера и результаты задачи.

- После завершения работы конвейера убедитесь, что другое приложение вернуло запрошенные данные. Для этого щелкните ссылку на ход выполнения конвейера.
- Щелкните задачу REST в конвейере.
- В ходе выполнения конвейера щелкните задачу REST, просмотрите сведения и убедитесь, что по задаче получены ожидаемые результаты.

В сведениях о задаче отображается код ответа, текст, ключи заголовка и значения.

BACK

Test #2
COMPLETED
ACTIONS

Stage0
Task0

Task name
Task0
[VIEW OUTPUT JSON](#)

Type
REST

Status
COMPLETED Execution Completed.

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

Continue on failure
☐

Execute task
☒ Always ☐ On condition

Response

Code
200

Body

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

Headers

Header Key	Header Value
X-Frame-Options	SAMEORIGIN
Transfer-Encoding	chunked
Cache-Control	private, max-age=0
Server	gws
Alt-Svc	quic="443" max=350000; us="44,42,30,25"

10. Чтобы просмотреть выходные данные JSON, щелкните **Просмотреть выходные данные JSON**.

```

1  {
2    "responseHeaders": {
3      "X-Frame-Options": "SAMEORIGIN",
4      "Transfer-Encoding": "chunked",
5      "Cache-Control": "private, max-age=0",
6      "Server": "gws",
7      "Alt-Svc": "quic=\":443\"; ma=2592000; v=\"44,43,39,35\"",
8      "Set-Cookie": "NID=148
          =RTUkVjVhyg9KvAZR1S8yCCSEw8WosYfn9MMDfQ1N5fnd5DavrXUM5B3J8PyKMX1Z_zRNp3usxttMpd7YiqRU05fMkTC7cTERbd
          UmOnj3cTppHe3PHIXJPGHnTSZEweb3cxtjvIHv0LS85ezVXaTSRYFcG0B_XIHZ8kqB8uwl1aE; expires=Tue, 28-May-2019
          22:45:06 GMT; path=/; domain=.google.com; HttpOnly",
9      "Expires": "-1",
10     "P3P": "CP=\"This is not a P3P policy! See g.co/p3phelp for more info.\"\"",
11     "X-XSS-Protection": "1; mode=block",
12     "Date": "Mon, 26 Nov 2018 22:45:06 GMT",
13     "Content-Type": "text/html; charset=ISO-8859-1"
14   },
15   "responseBody": "<!doctype html><html itemscope=\"\" itemtype=\"http://schema.org/WebPage\" lang=\"en-IN\"
          ><head><meta content=\"text/html; charset=UTF-8\" http-equiv=\"Content-Type\"><meta content=\"/images
          /branding/google/1x/google_standard_color_128dp.png\" itemprop=\"image\"><title>Google</title><script
          nonce=\"anWw/ydugkGr9CHU6QQGz==\">(function(){window.google={kEI:'cnf8W6KpJIEVkwXx-aLoDA',kEXPI:'0
          ,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457
          ,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,6056,636,2239
          ,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284
          ,2,579,727,612,1820,58,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
          ,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8
          ,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483
          ,698,59,318,273,100,167,323,744,101,1119,38,363,557,438,135,145,155,497,2,718,383,978,487,47,1080,901
          ,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37
          ,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,247,109,1049,14

```

Результаты

Поздравляем! Вы настроили задачу REST, которая вызывает интерфейс REST API и осуществляет обмен данными между Code Stream и другим приложением с помощью подключаемого модуля REST.

Следующие шаги

Продолжайте использовать задачи REST в конвейерах для выполнения команд и интеграции Code Stream с другими приложениями в процессе разработки и предоставления ПО. Рассмотрите возможность использования задач для опроса API-интерфейса до того момента, пока он не завершит работу, а задачи конвейера не будут соответствовать критериям выхода.

Использование конвейера в виде кода в Code Stream

Администратору или разработчику DevOps может потребоваться создать конвейер в Code Streamне через пользовательский интерфейс, а с помощью кода YAML. При создании конвейеров в виде кода можно использовать любой редактор и вставлять комментарии в код конвейера.

В коде конвейера можно обращаться к внешним конфигурациям, таким как переменные среды и учетные данные для безопасного доступа. Переменные, используемые в коде конвейера, можно обновлять, не обновляя код конвейера.

Код YAML конвейера можно использовать в качестве шаблона для клонирования и создания других конвейеров, а также для предоставления общего доступа к шаблонам.

Шаблоны кода конвейеров можно хранить в репозитории системы управления версиями, который назначает версии и отслеживает обновления. Используя систему управления версиями, можно легко создать резервную копию кода конвейера и при необходимости восстановить его.

Необходимые условия

- Убедитесь в наличии редактора кода.
- Если планируется хранить код конвейера в репозитории системы управления версиями, убедитесь в наличии доступа к рабочему экземпляру.

Процедура

1. Создайте файл в редакторе кода.
2. Скопируйте и вставьте образец кода конвейера и измените его с учетом требований конкретного конвейера.
3. Чтобы добавить конечную точку в код конвейера, скопируйте и вставьте пример кода конечной точки и измените его с учетом особенностей конечной точки.

При использовании конечной точки API-интерфейса Kubernetes в рабочей области конвейера Code Stream создаст необходимые для выполнения задачи непрерывной интеграции или настраиваемой задачи ресурсы Kubernetes, такие как ConfigMap, Secret и Pod. Code Stream обменивается данными с контейнером с помощью порта узла.

Рабочая область конвейера Code Stream поддерживает Docker и Kubernetes для выполнения задач непрерывной интеграции и настраиваемых задач.

Дополнительные сведения о настройке рабочей области см. в разделе [Настройка рабочей области конвейера](#).

4. Сохраните код.
5. Чтобы сохранить код конвейера с назначением версии, обновите код в репозитории системы управления версиями.
6. При создании конвейера непрерывной интеграции и предоставления необходимо импортировать файл Kubernetes YAML.

Чтобы импортировать файл Kubernetes YAML, выберите его в области непрерывного предоставления смарт-шаблона конвейера и щелкните **Обработать**. Или используйте API-интерфейс.

Результаты

С помощью примеров кода создан код YAML, который представляет собой конвейер и конечные точки.

Пример. Пример кода YAML для конвейера и конечных точек

Этот пример кода YAML включает в себя разделы, представляющие рабочую область для сборки с помощью встроенных инструментов, этапов, задач, уведомлений и других элементов Code Stream в конвейере.

Примеры кода для поддерживаемых подключаемых модулей см. в разделе [Глава 6 Подключение Code Stream к конечным точкам](#)

```
---
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}
---
```

Этот код YAML представляет собой пример конечной точки 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
---
```

Этот код YAML представляет собой пример конечной точки Kubernetes.

```

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

Следующие шаги

Запустите конвейер и внесите необходимые изменения. См. раздел [Запуск конвейера и просмотр результатов](#).

Подключение Code Stream к конечным точкам

6

Службу Code Stream можно интегрировать с инструментами разработки, используя подключаемые модули. В число поддерживаемых подключаемых модулей входят Jenkins, Bamboo, vRealize Operations, Bugzilla, Team Foundation Server, Git и многие другие.

Можно также создавать собственные подключаемые модули, чтобы интегрировать Code Stream с другими приложениями для разработки.

Для интеграции Code Stream с Jira внешний подключаемый модуль не требуется, так как Code Stream позволяет создавать обращения Jira в виде специальных уведомлений. Чтобы создать обращение Jira, касающееся состояния работы конвейера, необходимо добавить конечную точку Jira.

В эту главу входят следующие разделы:

- [Что такое конечные точки в Code Stream](#)
- [Интеграция Jenkins со службой Code Stream](#)
- [Интеграция Git со службой Code Stream](#)
- [Интеграция Gerrit со службой Code Stream](#)
- [Интеграция службы Code Stream с решением vRealize Orchestrator](#)

Что такое конечные точки в Code Stream

Конечная точка — это экземпляр приложения DevOps, который подключается к Code Stream и предоставляет данные, необходимые для работы конвейеров, например источник данных, репозиторий или система уведомлений.

Роль в Code Stream определяет способ использования конечных точек.

- Администраторы и разработчики могут создавать, обновлять, удалять и просматривать конечные точки.
- Администраторы могут пометить конечную точку как запрещенную и запускать конвейеры, которые используют запрещенные конечные точки.
- Пользователи, имеющие роль обозревателя, могут просматривать конечные точки, но не могут создавать, обновлять или удалять их.

Дополнительные сведения см. в разделе [Управление пользовательским доступом и утверждениями в службе Code Stream](#).

Чтобы подключить Code Stream к конечной точке, выполните следующие действия.

1. Добавьте задачу в конвейер.
2. Настройте задачу таким образом, чтобы она обменивалась данными с конечной точкой.
3. Убедитесь, что Code Stream может подключиться к конечной точке, нажав кнопку **Проверить**.
4. Затем при запуске конвейера задача подключается к конечной точке, чтобы она могла выполнить необходимые действия.

Сведения о типах задач, в которых используются эти конечные точки, см. в статье [Типы задач, доступные в Code Stream](#).

Таблица 6-1. Конечные точки, которые поддерживает Code Stream

Конечная точка	Функции конечной точки	Поддерживаемые версии	Требования
Bamboo	Создает планы сборки.	6.9.*	
Docker	Сборки с использованием встроенных средств могут использовать узлы Docker для развертывания.		Если конвейер включает в себя образ из Docker Hub, перед запуском конвейера необходимо убедиться, что в этом образе есть встроенная функция curl или wget. При выполнении конвейера Code Stream загружает двоичный файл, в котором для выполнения команд используется функция curl или wget.
Регистр Docker	Регистрирует образы контейнеров, чтобы узел сборки Docker мог получать образы.	2.7.1	
Gerrit	Подключается к серверу Gerrit для проверок и запуска	2.14.*	
Git	Запускает конвейеры, когда разработчики обновляют код и сохраняют его в репозитории.	Git Hub Enterprise 2.1.8 Git Lab Enterprise 11.9.12-ee	
Jenkins	Создание артефактов приложения.	1.6.* и 2.*	
Jira	Создает запрос Jira при неудачном выполнении задачи конвейера.	8.3.*	

Таблица 6-1. Конечные точки, которые поддерживает Code Stream (продолжение)

Конечная точка	Функции конечной точки	Поддерживаемые версии	Требования
Kubernetes	Автоматизирует шаги, необходимые для развертывания, масштабирования и управления контейнерными приложениями.	Все поддерживаемые версии для Cloud Assembly 8.4 и более поздней версии 1.18 для Cloud Assembly 8.3 и более ранней версии	При использовании конечной точки API-интерфейса Kubernetes в рабочей области конвейера Code Stream создает необходимые для выполнения задачи непрерывной интеграции или настраиваемой задачи ресурсы Kubernetes, такие как ConfigMap, Secret и Pod. Code Stream обменивается данными с контейнером с помощью порта узла. Дополнительные сведения о настройке рабочей области см. в разделе Настройка рабочей области конвейера .
PowerShell	Создает задачи, которые запускают сценарии PowerShell на компьютерах с Windows или Linux.	4 и 5	
SSH	Создает задачи, которые запускают сценарии SSH на компьютерах с Windows или Linux.	7.0	
TFS (Team Foundation Server)	Управляет исходным кодом, автоматизированными сборками, тестированием и соответствующими действиями.	2015 и 2017	
vRealize Orchestrator	Упорядочивает и автоматизирует рабочие процессы во время сборки.	7.* и 8.*	

Пример кода YAML для конечной точки GitHub

В этом примере кода YAML определяется конечная точка GitHub, которую можно указать в задаче 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
---
```

Интеграция Jenkins со службой Code Stream

Служба Code Stream включает в себя подключаемый модуль Jenkins, который запускает задания Jenkins, позволяющие выполнять сборку и тестирование исходного кода. Подключаемый модуль Jenkins позволяет выполнять различные варианты тестирования и поддерживает использование настраиваемых сценариев.

Чтобы запустить задание Jenkins в конвейере, воспользуйтесь сервером Jenkins и добавьте конечную точку Jenkins в службу Code Stream. Затем создайте конвейер и добавьте в него задачу Jenkins.

При использовании задачи Jenkins и конечной точки Jenkins в Code Stream можно создать конвейер, поддерживающий задания с несколькими ветвями в Jenkins. Задание с несколькими ветвями включает в себя отдельные задания в каждой ветви репозитория Git. При создании конвейеров в Code Stream, которые поддерживают задания с несколькими ветвями:

- Задача Jenkins может запускать задания Jenkins, которые находятся в разных папках на сервере Jenkins.
- В конфигурации задачи Jenkins можно переопределить путь к папке по умолчанию, указанный в конечной точке Jenkins в Code Stream.
- Конвейеры с несколькими ветвями в Code Stream обнаруживают файлы заданий Jenkins типа `.groovy` в репозитории Git или репозитории GitHub и начинают создавать задания для каждой ветви, которую они сканируют в репозитории.
- Путь по умолчанию, указанный в конечной точке Jenkins можно заменить на путь, указанный в конфигурации задачи Jenkins, чтобы запускать задание и конвейер, связанные с любой ветвью в основном задании Jenkins.

Необходимые условия

- Настройте сервер Jenkins 1.561 или более поздней версии.
- Проверьте, являетесь ли вы участником проекта в Code Stream. Если это не так, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).
- Убедитесь в наличии задания на сервере Jenkins (в противном случае задача конвейера не сможет его выполнить).

Процедура

1. Добавьте и проверьте конечную точку Jenkins.
 - а) Выберите пункт **Конечные точки > Новая конечная точка**.
 - б) Выберите проект и укажите **Jenkins** в качестве типа конечной точки. Затем введите имя и описание.
 - в) Если эта конечная точка — критически важный для бизнеса компонент инфраструктуры, установите флажок **С ограниченным доступом**.
 - г) Введите URL-адрес сервера Jenkins.

- д) Введите имя пользователя и пароль для входа на сервер Jenkins. Затем введите прочие сведения.

Таблица 6-2. Прочие сведения для конечной точки Jenkins

Запись о конечной точке	Описание
Путь к папке	<p>Путь к папке, в которой собираются задания. Jenkins может запустить все задания в папке. Можно также создавать вложенные папки. Например:</p> <ul style="list-style-type: none"> ■ <code>folder_1</code> может содержать <code>job_1</code> ■ <code>folder_1</code> может содержать <code>folder_2</code>, в которой может быть <code>job_2</code> <p>При создании конечной точки для <code>folder_1</code> в качестве пути к папке используется <code>job/folder_1</code> и в списке заданий конечной точки будет только <code>job_1</code>.</p> <p>Чтобы получить список заданий в дочерней папке с именем <code>folder_2</code>, необходимо создать другую конечную точку, где в качестве пути к папке будет использоваться <code>/job/folder_1/job/folder_2/</code>.</p>
Путь к папке для заданий Jenkins с несколькими ветвями	<p>Для поддержки заданий Jenkins с несколькими ветвями в задаче Jenkins необходимо ввести полный путь, который включает в себя URL-адрес сервера Jenkins и полный путь к заданию. При добавлении пути к папке в задаче Jenkins этот путь переопределит путь, который отображается в конечной точке Jenkins. При наличии настраиваемого пути к папке в задаче Jenkins служба Code Stream отображает только те задания, которые имеются в этой папке.</p> <ul style="list-style-type: none"> ■ Например, <code>https://server.yourcompany.com/job/project</code> ■ Если конвейер должен также запускать основное задание Jenkins, используйте путь <code>https://server.yourcompany.com/job/project/job/main</code>
URL-адрес	<p>URL-адрес узла для сервера Jenkins. Введите URL-адрес в формате <code>protocol://host:port</code>. Например, <code>http://192.10.121.13:8080</code></p>
Интервал между опросами	<p>Продолжительность интервала между проверками сервера Jenkins на наличие обновлений, выполняемыми службой Code Stream.</p>

Таблица 6-2. Прочие сведения для конечной точки Jenkins (продолжение)

Запись о конечной точке	Описание
Количество повторных попыток запроса	Количество повторных попыток отправить запрос запланированной сборки на сервер Jenkins.
Время ожидания между повторными попытками	Количество секунд между повторными попытками отправить запрос сборки на сервер Jenkins.

- е) Нажмите **Проверить** и убедитесь, что конечная точка подключена к службе Code Stream. Если подключение отсутствует, устраните проблему, мешающую подключению, и нажмите **Сохранить**.

Edit Endpoint

Project

test1

Type

Jenkins

Name *

aa

Description

Mark restricted

☐ non-restricted

URL *


http(s)://<server_url>:<port>

Username

username

Password

Enter password



CREATE VARIABLE

Folder Path

/job/DevFolder/

Poll Interval (sec) *

15

Request Retries *

5

Retry Wait Time (sec) *

60

SAVE

VALIDATE

CANCEL

2. Чтобы выполнить сборку кода, создайте конвейер и добавьте задачу, использующую конечную точку Jenkins.
 - а) Выберите пункт **Конвейеры > Создать конвейер > Пустой холст**.
 - б) Щелкните этап по умолчанию.
 - в) В области «Задача» (Task) введите имя задачи.
 - г) В качестве типа задачи укажите **Jenkins**.
 - д) Выберите созданную конечную точку Jenkins.
 - е) В раскрывающемся меню выберите задание на сервере Jenkins, которое будет выполнено конвейером.
 - ж) Введите параметр для задания.
 - з) Введите маркер проверки подлинности для задания Jenkins.

The screenshot displays the 'Build and Deploy' stage configuration in the vRealize Automation Code Stream interface. The stage is titled 'Stage0' and is currently 'Enabled'. The main workspace shows a visual pipeline with a 'Build' task (Jenkins) and a 'Test' task (Jenkins) connected sequentially. Below them is a 'Parallel Task' placeholder and a '+ Stage' button.

On the right, the 'Task : Build' configuration panel is open, showing the following settings:

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

At the bottom of the interface, there are buttons for 'SAVE', 'RUN', and 'CLOSE', along with a status indicator 'Last saved a month ago' and a chat icon.

3. Включите и запустите конвейер и проследите, как выполняется его работа.

[< BACK](#)

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

Stage0

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

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

4. Просмотр подробностей по выполнению и состоянию конвейера на панели управления конвейером.

Здесь можно увидеть ошибки в работе конвейера и определить их причины. Здесь также можно отслеживать тенденции, касающиеся продолжительности работы конвейера, успешных случаев его выполнения и сбоев.

The screenshot shows the 'Build and Deploy' interface. At the top, there are tabs for '1D', '7D', and '14D'. Below this is a 'Recent Executions' section with a list of execution numbers (#20 to #29) and a visual progress bar for 'Stage0'. A legend indicates: green for 'Completed', red for 'Failed', blue for 'Running', and yellow for 'Waiting'.

Below the recent executions is the 'Execution Details' section, which contains a table with the following data:

Execution#	Status	Status Message	Duration	Updated On
#29	FAILED	Execution failed on task 'Stage0.Deployment'. namespaces "prod1" already exists	1m 32s	08/19 10:49PM
#28	COMPLETED	Execution Completed.	3m 42s	08/06 12:30AM
#27	COMPLETED	Execution Completed.	1m 45s	08/06 12:24AM
#26	FAILED	Execution failed on task 'Stage0.Deployment'. Conflict	1m 8s	08/06 12:19AM
#25	COMPLETED	Execution Completed.	2m 11s	08/06 12:07AM
#24	COMPLETED	Execution Completed.	58s	08/05 11:59PM
#23	FAILED	Execution failed on task 'Stage0.Approval for Deployment'. User Operation request has been	4m 55s	08/06 12:03AM

Результаты

Поздравляем! Вы интегрировали Jenkins со службой Code Stream, добавили конечную точку, создали конвейер и настроили задачу Jenkins, выполняющую сборку кода.

Пример. Пример кода YAML для задачи сборки Jenkins

Для задач по сборке Jenkins, аналогичных данному примеру, код YAML будет выглядеть приблизительно следующим образом (с включенными уведомлениями):

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

Следующие шаги

Дополнительные сведения см. в других разделах. См. раздел [Глава 6 Подключение Code Stream к конечным точкам](#).

Интеграция Git со службой Code Stream

Служба Code Stream позволяет настроить запуск конвейера при внесении изменений в код, хранящийся в репозитории GitHub, GitLab или Bitbucket. Триггер Git использует конечную точку Git ветви репозитория, состояние которого требуется отслеживать. Для подключения службы Code Stream к конечной точке Git используется веб-перехватчик.

Чтобы определить конечную точку Git в Code Stream, выберите проект и введите ветвь репозитория Git, в которой находится конечная точка. В проекте конвейер группируется с конечной точкой и другими связанными объектами. При выборе проекта в определении веб-перехватчика необходимо указать конечную точку и конвейер, который будет запускаться.

Примечание Если веб-перехватчик определен по конечной точке, а затем конечная точка изменена, сведения об этой конечной точке в веб-перехватчике изменить нельзя. Чтобы изменить сведения о конечной точке, необходимо удалить веб-перехватчик и заново задать конечную точку в определении веб-перехватчика. См. раздел [Как использовать триггер Git в Code Stream для запуска конвейера](#).

Можно создать несколько веб-перехватчиков для различных ветвей с помощью одной конечной точки Git, указав разные значения для имени ветви на странице конфигурации веб-перехватчика. Чтобы создать другой веб-перехватчик для другой ветви в том же репозитории Git, не нужно несколько раз клонировать конечную точку Git для нескольких ветвей. Просто укажите имя ветви в веб-перехватчике, чтобы использовать конечную точку Git еще раз. Если ветвь веб-перехватчика Git совпадает с ветвью в конечной точке, не нужно указывать имя ветви на странице веб-перехватчика Git.

Необходимые условия

- Убедитесь в наличии доступа к репозиторию GitHub, GitLab или Bitbucket, подключение к которому планируется установить.
- Проверьте, являетесь ли вы участником проекта в Code Stream. Если нет, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).

Процедура

1. Определите конечную точку Git.
 - а) Выберите пункт **Конечные точки > Новая конечная точка**.
 - б) Выберите проект и укажите **Git** в качестве типа конечной точки. Затем введите имя и описание.

- в) Если эта конечная точка — критически важный для бизнеса компонент инфраструктуры, установите флажок **С ограниченным доступом**.

Если в конвейере используется конечная точка с ограниченным доступом, администратор может запустить конвейер и должен одобрить его выполнение. Если стоит отметка об ограничении доступа к конечной точке или переменной, а пользователь, не являющийся администратором, запускает конвейер, эта задача приостанавливается и ожидает возобновления администратором.

Администратор проекта может запускать конвейер, который включает в себя конечные точки или переменные с ограниченным доступом, если эти ресурсы относятся к проекту, администратором которого является пользователь.

Если пользователь, который не является администратором, пытается запустить конвейер, в котором есть ресурс с ограниченным доступом, конвейер останавливается на задаче, в которой используется такой ресурс. После этого администратор должен возобновить работу конвейера.

Для получения дополнительных сведений о ресурсах с ограниченным доступом и настраиваемых ролях, которые включают разрешение **Управление конвейерами с ограниченным доступом**, см. следующий раздел.

- [Управление пользовательским доступом и утверждениями в службе Code Stream](#)
- [Глава 2 Настройка Code Stream для моделирования процесса выпуска](#)

- г) Выберите один из поддерживаемых типов серверов Git.
- д) В пути введите URL-адрес репозитория со шлюзом API-интерфейса для сервера. Например:
- Для GitHub введите `https://api.github.com/vmware-example/repo-example`
- Для BitBucket введите `https://api.bitbucket.org/{user}/{repo name}` или `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`
- е) Введите ветвь репозитория, в которой находится конечная точка.
- ж) Выберите тип проверки подлинности и введите имя пользователя GitHub, GitLab или BitBucket. Затем введите закрытый маркер, который используется вместе с именем пользователя.

- Пароль. Чтобы позже создать веб-перехватчик, необходимо ввести закрытый маркер для пароля. Веб-перехватчики для Git не поддерживают конечные точки, созданные с помощью базовой проверки подлинности.

Используйте секретные переменные для скрытия и шифрования конфиденциальной информации. Используйте переменную с ограниченным доступом для строк, паролей и URL-адресов, которые должны быть скрыты и зашифрованы, а также для ограничения возможностей использования элементов в циклах выполнения. Например, для пароля или URL-адреса. Секретные переменные и переменные с ограниченным доступом можно использовать в любом типе задач конвейера.

- Закрытый маркер. Этот маркер используется исключительно в Git и позволяет получить доступ к конкретному действию. См. раздел https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html. Кроме того, можно создать переменную для закрытого маркера.

2. Нажмите **Проверить** и убедитесь, что конечная точка подключена к службе Code Stream.

Если подключение отсутствует, устраните проблему, мешающую подключению, и нажмите **Создать**.

New endpoint

Project *

test

Type *

GIT

Name *

DemoApp-Git

Description

Git example branch

Mark restricted

☐ non-restricted

Git Server Type *

GitHub

Repo URL ⓘ *

https://api.github.com/vmware-example/repo-example

ACCEPT CERTIFICATE

Branch *

master

Authentication Type *

Password

Username *

ExampleUser

Password *

.....

✖

CREATE VARIABLE

CREATE

VALIDATE

CANCEL

Следующие шаги

Дополнительные сведения см. в других разделах. См. раздел [Как использовать триггер Git в Code Stream для запуска конвейера](#).

Интеграция Gerrit со службой Code Stream

Служба Code Stream позволяет настроить запуск конвейера при внесении изменений в код, хранящийся в проекте Gerrit. Определение триггера для Gerrit включает в себя проект Gerrit и конвейеры, которые должны выполняться для разных типов событий.

В триггере Gerrit используется прослушиватель Gerrit на отслеживаемом сервере Gerrit. Чтобы определить конечную точку Gerrit в Code Stream, выберите проект и введите URL-адрес сервера Gerrit. Затем при создании прослушивателя Gerrit на этом сервере необходимо создать конечную точку.

Если сервер Gerrit используется в качестве конечной точки Code Stream в экземпляре vRealize Automation с включенным шифрованием FIPS, необходимо убедиться, что в файле конфигурации Gerrit содержатся правильные ключи проверки подлинности сообщения. Если правильные ключи проверки подлинности сообщения отсутствуют, сервер не сможет запуститься правильно и выдаст сообщение:

```
PrivateKey/PassPhrase is incorrect
```

Необходимые условия

- Убедитесь в наличии доступа к серверу Gerrit, подключение к которому планируется установить.
- Проверьте, являетесь ли вы участником проекта в Code Stream. Если это не так, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).

Процедура

1. Определите конечную точку Gerrit.

- а) Щелкните **Настроить > Конечные точки** и затем **Создать конечную точку**.
- б) Выберите проект и укажите **Gerrit** в качестве типа конечной точки. Затем введите имя и описание.
- в) Если эта конечная точка — критически важный для бизнеса компонент инфраструктуры, установите флажок **С ограниченным доступом**.
- г) Введите URL-адрес сервера Gerrit.

Чтобы использовать порт по умолчанию, укажите номер порта с URL-адресом или оставьте значение пустым.

- д) Введите имя пользователя и пароль для доступа к серверу Gerrit.

Если необходимо зашифровать пароль, щелкните **Создать переменную** и выберите один из следующих типов переменных.

- Секретная. Пароль обрабатывается, если конвейер запускается пользователем с любой ролью.
- С ограниченным доступом. Пароль обрабатывается, если конвейер запускает пользователь с ролью администратора.

В качестве значения введите защищенный пароль, например пароль сервера Jenkins.

- е) В качестве закрытого ключа введите SSH-ключ, используемый для безопасного доступа к серверу Gerrit.

Он представляет собой закрытый ключ RSA, расположенный в каталоге `.ssh`.

- ж) (дополнительно) Если закрытый ключ сопровождается парольной фразой, введите парольную фразу.

Чтобы зашифровать парольную фразу, нажмите **Создать переменную** и выберите один из следующих типов переменной.

- Секретная. Пароль обрабатывается, если конвейер запускается пользователем с любой ролью.
- С ограниченным доступом. Пароль обрабатывается, если конвейер запускает пользователь с ролью администратора.

В качестве значения введите защищенную парольную фразу, например парольную фразу для сервера SSH.

2. Щелкните **Проверить** и убедитесь, что конечная точка Gerrit в Code Stream подключается к серверу Gerrit.

Если подключение отсутствует, устраните причины проблемы и нажмите **Проверить**.

New endpoint

Project	test
Type	Gerrit
Name *	Gerrit-Demo-Endpoint
Description	
Mark restricted	<input type="checkbox"/> non-restricted
URL *	http://example-gerrit.mycompany.com:8080
Username *	CS_user
Password *	<div> <div>.....</div> <div>✖</div> <div>CREATE VARIABLE</div> </div>
Private Key *	<div> <div>-----BEGIN RSA PRIVATE KEY-----</div> <div>Proc-Type: 4,ENCRYPTED</div> <div>DEK-Info: AES-128-CBC,F00CE0B6526AF67DC77ADCD0962DBF92</div> </div>
Pass Phrase ⓘ	<div> <div>.....</div> <div>✖</div> <div>CREATE VARIABLE</div> </div>
<div> <div>CREATE</div> <div>VALIDATE</div> <div>CANCEL</div> </div>	

3. Щелкните **Создать**.

4. Убедитесь, что в среде vRealize Automation включено шифрование FIPS, или создайте среду с шифрованием FIPS с помощью задания Jenkins, используя URL-адрес Jenkins.
 - а) Чтобы выполнить команду из командной строки, подключите свое устройство vRealize Automation 8.x по протоколу SSH и войдите в систему как пользователь root.
Можно подключиться к URL-адресу полного доменного имени, например `https://cava-1-234-567.yourcompanyFQDN.com`, используя порт 22, 5480 или 443.
 - б) Чтобы проверить, включено ли шифрование FIPS в vRealize Automation, выполните команду `vracli security fips`.
 - в) Убедитесь, что команда возвращает `FIPS mode: strict`.
5. Если сервер Gerrit является конечной точкой в экземпляре vRealize Automation, где включено шифрование FIPS, убедитесь, что в файле конфигурации Gerrit содержатся правильные ключи проверки подлинности сообщения (MAC).
 - а) Откройте Gerrit и создайте пару ключей SSH.
 - б) Найдите файл конфигурации сервера Gerrit `'$site_path'/etc/gerrit.config`.
 - в) Убедитесь, что в этом файле имеется один или несколько ключей кода проверки подлинности сообщения (MAC), за исключением `hmac-MD5`.

Примечание В режиме FIPS алгоритм MAC `hmac-MD5` не поддерживается. Чтобы сервер Gerrit запускался правильно, этот алгоритм должен быть исключен в файле конфигурации сервера Gerrit. Если сервер Gerrit не запустится правильно, он выдаст сообщение: `PrivateKey/PassPhrase is incorrect`

Активируются поддерживаемые имена ключей для кода проверки подлинности сообщений (MAC), которые начинаются со знака плюс (+). Имена ключей MAC, которые начинаются с дефиса (-), удаляются из списка ключей MAC по умолчанию. По умолчанию в Code Stream для сервера Gerrit доступны следующие поддерживаемые коды MAC:

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

Следующие шаги

Дополнительные сведения см. в других разделах. См. раздел [Как использовать триггер Gerrit в Code Stream для запуска конвейера](#).

Интеграция службы Code Stream с решением vRealize Orchestrator

Службу Code Stream можно интегрировать с vRealize Orchestrator (vRO), чтобы дополнить ее функции возможностью выполнять рабочие процессы vRO. vRealize Orchestrator содержит множество предварительно определенных рабочих процессов, которые можно интегрировать со сторонними инструментами. Эти рабочие процессы позволяют автоматизировать процессы DevOps и управлять ими, автоматизировать массовые операции и многое другое.

Например, рабочий процесс в задаче vRO конвейера можно использовать для активации или удаления пользователя, перемещения виртуальных машин, интеграции со средствами тестирования для проверки кода в ходе выполнения конвейера и т. д. Примеры кода для рабочих процессов vRealize Orchestrator можно найти на сайте code.vmware.com.

Рабочий процесс vRealize Orchestrator позволяет конвейеру совершать различные действия в ходе выполнения сборки, тестирования и развертывания приложения. В конвейер можно включить предварительно настроенные рабочие процессы. Можно также создать и использовать настраиваемые рабочие процессы. Каждый рабочий процесс состоит из входных данных, задач и выходных данных.

Чтобы рабочий процесс vRO можно было запустить в рамках конвейера, он должен отображаться в списке доступных рабочих процессов в задаче vRO, включенной в конвейер.

Прежде чем рабочий процесс появится в задаче vRO конвейера, администратору нужно выполнить следующие действия в vRealize Orchestrator:

1. Примените тег CODESTREAM к рабочему процессу vRO.
2. Пометьте рабочий процесс vRO как глобальный.

Необходимые условия

- Убедитесь в наличии доступа к локальному экземпляру vRealize Orchestrator с правами администратора. Для получения справки обратитесь к своему администратору либо см. [документацию по vRealize Orchestrator](#).
- Проверьте, являетесь ли вы участником проекта в Code Stream. Если нет, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).
- Создайте конвейер в Code Stream и добавьте в него новый этап.

Процедура

1. Администратору необходимо подготовить рабочий процесс vRealize Orchestrator для выполнения конвейера.
 - а) В vRealize Orchestrator найдите рабочий процесс, который требуется использовать в конвейере, например рабочий процесс для активации пользователя.

Если нужный рабочий процесс отсутствует, его можно создать.
 - б) В строке поиска введите **Рабочий процесс с тегом**, чтобы найти рабочий процесс с именем Рабочий процесс с тегом.
 - в) На карточке Рабочий процесс с тегом выберите **Запустить**. Отобразится область настройки.
 - г) В текстовом поле Рабочий процесс с тегом введите имя рабочего процесса, который будет использоваться в конвейере Code Stream, а затем выберите его из списка.
 - д) В текстовых областях Тег и Значение укажите CODESTREAM заглавными буквами.
 - е) Установите флажок **Глобальный тег**.
 - ж) Щелкните **Запустить**. Тег CODESTREAM будет присвоен рабочему процессу, который необходимо выбрать в конвейере Code Stream.
 - з) В панели навигации выберите **Рабочие процессы** и убедитесь, что тег с именем CODESTREAM отображается в карточке рабочего процесса, который будет запускать конвейер.

После входа в Code Stream и добавления в конвейер задачи vRO в списке рабочих процессов появится рабочий процесс с тегом.
2. Создайте в службе Code Stream конечную точку для экземпляра vRealize Orchestrator.
 - а) Выберите пункт **Конечные точки > Новая конечная точка**.
 - б) Выберите проект.
 - в) Введите соответствующее имя.

- г) Введите URL-адрес конечной точки vRealize Orchestrator.

Используйте следующий формат: **https://vro-appliance.yourdomain.local:8281**

Не используйте следующий формат: **https://vro-appliance.yourdomain.local:8281/vco/api**

URL-адрес экземпляра vRealize Orchestrator, встроенного в устройство vRealize Automation, является полным доменным именем для устройства без порта. Например: **https://vro-appliance.yourdomain.local/vco**

Для внешних устройств vRealize Orchestrator, начиная с версии vRealize Automation 8.x, полным доменным именем для устройства является **https://vro-appliance.yourdomain.local**

Для внешних устройств vRealize Orchestrator, добавленных с версией vRealize Automation 7.x, полным доменным именем для устройства является **https://vro-appliance.yourdomain.local:8281/vco**

Если при добавлении конечной точки возникает проблема, возможно, будет нужно импортировать конфигурацию YAML с контрольной суммой сертификата SHA-256, где удалены двоеточия.

Например, **B0:01:A2:72 . . .** становится **B001A272 . . .**. Пример кода YAML имеет приблизительно следующий вид:

```

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

- д) Щелкните **Принять сертификат**, если для введенного URL-адреса требуется сертификат.
- е) Введите имя пользователя и пароль для доступа к серверу vRealize Orchestrator.

Если для проверки подлинности используется нелокальный пользователь, не вводите доменную часть имени пользователя. Например, для проверки подлинности с **svc_vro@yourdomain.local** необходимо ввести **svc_vro** в текстовой области **Имя пользователя**.

3. Подготовьте конвейер к выполнению задачи vRO.

- а) Добавьте задачу vRO в этап конвейера.
- б) Введите соответствующее имя.
- в) В области «Свойства рабочего процесса» выберите конечную точку vRealize Orchestrator.

- г) Выберите рабочий процесс с тегом CODESTREAM в vRealize Orchestrator.

При выборе настраиваемого рабочего процесса может понадобиться ввести значения входных параметров.

- д) Для параметра **Выполнение задачи** задайте значение **При условии**.

The screenshot shows the configuration window for a task named 'vRO workflow'. The interface includes tabs for 'Task : vRO workflow', 'Notifications', and 'Rollback', with a 'VALIDATE TASK' button in the top right. The configuration fields are as follows:

- Task name**: vRO workflow
- Type**: vRO (dropdown menu)
- Duration**: NaNps (-)
- Continue on failure**: ☐
- Execute task**: ☐ Always ☒ On condition
- Condition**: An empty text box with a dollar sign (\$) and an information icon (i).

Below these fields are the **Workflow Properties**:

- Endpoint**: vROEP (dropdown menu)
- Workflow**: Test (dropdown menu)
- Greeting**: Hello! (text field)

At the bottom, there are **Output Parameters** with buttons for 'status' and 'properties'.

- е) Введите условия, которые применяются в ходе работы конвейера.

Когда запускать конвейер...	Выберите условия...
При условии	<p>Задача конвейера выполняется только в том случае, если заданное условие соблюдено (имеет значение «истина»). Если условие не соблюдено (имеет значение «ложь»), задача пропускается.</p> <p>В задачу vRO можно добавлять логические выражения, в которых используются перечисленные ниже операнды и операторы.</p> <ul style="list-style-type: none"> ■ Переменные конвейера, такие как <code>\${pipeline.variableName}</code>. Фигурные скобки используются только при вводе переменных. ■ Переменные выходных данных задачи, такие как <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code>. ■ Переменные привязки конвейеров по умолчанию, такие как <code>\${releasePipelineName}</code>. ■ Логические значения без учета регистра, такие как <code>true</code>, <code>false</code>, <code>'true'</code> и <code>'false'</code>. ■ Целочисленные или десятичные значения без кавычек. ■ Строковые значения с одинарными или двойными кавычками, такие как <code>"test"</code> и <code>'test'</code>. ■ Строковые и числовые типы значений, такие как <code>== Equals</code> и <code>!= Not Equals</code>. ■ Операторы отношения, такие как <code>></code>, <code>>=</code>, <code><</code> и <code><=</code>. ■ Логические операторы, такие как <code>&&</code> и <code> </code>. ■ Арифметические операторы, такие как <code>+</code>, <code>-</code>, <code>*</code> и <code>/</code>. ■ Вложенные выражения с фигурными скобками. ■ Строки, содержащие буквенное значение ABCD, считаются ложными, и задача пропускается. ■ Унарные операторы не поддерживаются. <p>Пример условия: <code>\${Stage1.task1.output} == "Passed" \${pipeline.variableName} == 39</code></p>
Всегда	Если выбрать параметр Всегда , конвейер выполняет задачу без учета условий.

- ж) Введите приветственное сообщение.
- з) Щелкните **Проверить задачу** и исправьте возникшие ошибки.

4. Сохраните, включите и запустите конвейер.

5. Проверьте результаты работы конвейера.

- а) Щелкните **Выполняемые элементы**.
- б) Щелкните нужный конвейер.
- в) Щелкните нужную задачу.
- г) Проверьте результаты, входное значение и свойства.

Здесь можно просмотреть идентификатор выполнения рабочего процесса, имя пользователя, отреагировавшего на задачу, время ответа и все оставленные им комментарии.

Результаты

Поздравляем! Вы назначили тег рабочему процессу vRealize Orchestrator, который требовалось использовать в службе Code Stream, и добавили задачу vRO в конвейер Code Stream для выполнения рабочего процесса, автоматизирующего действие в среде DevOps.

Пример. Формат выходных данных задачи vRO

Формат выходных данных задачи vRO аналогичен данному примеру.

```
[{
    "name": "result",
    "type": "STRING",
    "description": "Result of workflow run.",
    "value": ""
},
{
    "name": "message",
    "type": "STRING",
    "description": "Message",
    "value": ""
}]
```

Следующие шаги

Добавьте в конвейеры дополнительные задачи, включающие рабочие процессы vRO, чтобы автоматизировать выполнение задач в средах разработки, тестирования и производства.

Запуск конвейеров в Code Stream

7

При необходимости Code Stream может запускать конвейер при возникновении определенных событий.

Например:

- Триггер Docker может запускать конвейер после создания нового артефакта или обновления существующего.
- Триггер для Git может запускать конвейер, когда разработчики обновляют код.
- Триггер для Gerrit может запускать конвейер, когда разработчики проверяют код.

В эту главу входят следующие разделы:

- [Как использовать триггер Docker в Code Stream для запуска конвейера непрерывного предоставления](#)
- [Как использовать триггер Git в Code Stream для запуска конвейера](#)
- [Как использовать триггер Gerrit в Code Stream для запуска конвейера](#)

Как использовать триггер Docker в Code Stream для запуска конвейера непрерывного предоставления

Администратор или разработчик Code Stream может использовать триггер Docker в Code Stream. Триггер Docker запускает автономный конвейер непрерывного предоставления (CD) при создании или обновлении артефакта сборки. Триггер Docker запускает конвейер CD, который передает новый или обновленный артефакт в виде образа контейнера в репозиторий центра Docker. Конвейер CD может выполняться в рамках автоматизированной сборки.

Например, для непрерывного развертывания обновленного образа контейнера с помощью конвейера CD используйте триггер Docker. При проверке образа контейнера в реестре Docker веб-перехватчик в центре Docker отправляет уведомление об изменении образа в службу Code Stream. Это уведомление инициирует запуск конвейера непрерывного предоставления с обновленным образом контейнера и отправку образа в репозиторий центра Docker.

Для использования триггера Docker необходимо выполнить несколько шагов в Code Stream.

Таблица 7-1. Как использовать триггер Docker

Действия...	Сведения о действии...
Создайте конечную точку реестра Docker.	Чтобы решение Code Stream могло запускать конвейер, необходима конечная точка реестра Docker. Если конечная точка не существует, можно выбрать параметр, который создаст ее при добавлении веб-перехватчика для триггера Docker. Конечная точка реестра Docker включает в себя URL-адрес репозитория центра Docker.
Добавьте в конвейер входные параметры, которые автоматически добавляют параметры Docker во время работы конвейера.	Возможна вставка параметров Docker в конвейер. Параметры могут включать в себя имя владельца события Docker, образ, имя репозитория, пространство имен репозитория и тег. В конвейер CD следует добавить входные параметры, которые веб-перехватчик Docker передает в конвейер до его запуска.
Создайте веб-перехватчик Docker.	При создании веб-перехватчика Docker в Code Stream создается соответствующий веб-перехватчик в центре Docker. Веб-перехватчик Docker в Code Stream подключается к центру Docker по URL-адресу, который указывается в веб-перехватчике. Веб-перехватчики обмениваются данными друг с другом и запускают конвейер при создании или обновлении артефакта в центре Docker. При обновлении или удалении веб-перехватчика Docker в Code Stream также обновляется или удаляется веб-перехватчик в центре Docker.
Добавьте и настройте в конвейере задачу Kubernetes.	При создании или обновлении артефакта в репозитории центра Docker запускается конвейер. Затем он развертывает артефакт через конвейер на узле Docker в кластере Kubernetes.
Добавьте в задачу локальное определение YAML.	Определение YAML, применяемое к задаче развертывания, содержит образ контейнера Docker. Если необходимо загрузить образ из частного репозитория, файл YAML должен содержать раздел с секретом конфигурации Docker. См. подраздел о непрерывном предоставлении в разделе Планирование собственной сборки по модели непрерывной интеграции и непрерывной доставки в Code Stream перед использованием смарт-шаблона конвейера .

При создании или обновлении артефакта в репозитории центра Docker веб-перехватчик в центре Docker отправляет уведомление веб-перехватчику в Code Stream, который запускает конвейер. Выполняются следующие действия:

1. Центр Docker отправляет запрос POST на URL-адрес, указанный в веб-перехватчике.
2. Code Stream запускает триггер Docker.
3. Триггер Docker запускает конвейер CD.
4. Конвейер Docker отправляет артефакт в репозиторий центра Docker.
5. Code Stream запускает собственный веб-перехватчик Docker, который запускает конвейер CD, развертывающий артефакт на узле Docker.

В этом примере создаются конечная точка Docker и веб-перехватчик Docker в Code Stream, который развертывает приложение в кластере разработки Kubernetes. Эти шаги включают в себя пример кода для полезных данных, который Docker отправляет по URL-адресу в веб-перехватчике, используемый код API-интерфейса и код проверки подлинности с защищенным маркером.

Необходимые условия

- Убедитесь, что в экземпляре Code Stream существует конвейер непрерывного предоставления (CD). Кроме того, убедитесь в наличии одной или нескольких задач Kubernetes, которые развертывают приложение. См. раздел [Глава 4 Планирование сборки, интеграции и предоставления кода встроенными средствами в Code Stream](#).
- Убедитесь, что у вас есть доступ к существующему кластеру Kubernetes, в котором конвейер CD может развернуть приложение для разработки.
- Проверьте, являетесь ли вы участником проекта в Code Stream. Если нет, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).

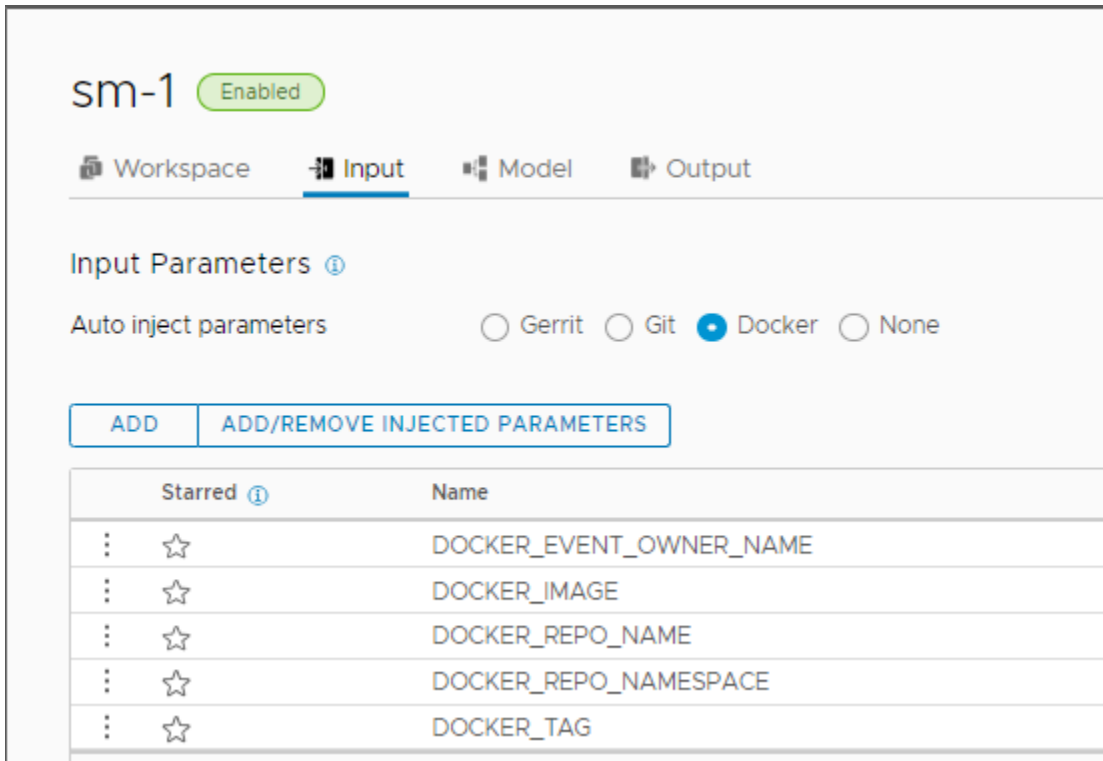
Процедура

1. Создайте конечную точку реестра Docker.
 - а) Щелкните **Конечные точки**.
 - б) Щелкните **Создать конечную точку**.
 - в) Начните вводить имя существующего проекта.
 - г) Выберите тип **Реестр Docker**.
 - д) Введите соответствующее имя.
 - е) Выберите тип сервера **Центр Docker**.
 - ж) Введите URL-адрес репозитория центра Docker.
 - з) Введите имя и пароль для доступа к репозиторию.

New endpoint

Project *	<input type="text" value="AWS_PGProj"/>
Type *	<input type="text" value="Docker Registry"/>
Name *	<input type="text" value="dockerhub-endpoint"/>
Description	<input type="text"/>
Mark restricted	<input type="checkbox"/> non-restricted
Server type *	<input type="text" value="DockerHub"/>
Repo URL *	<input type="text" value="https://hub.docker.com/repository/docker/automation/cs-builder"/> <input type="button" value="ACCEPT CERTIFICATE"/>
Username *	<input type="text" value="admin"/>
Password *	<input type="password" value="....."/> <input type="button" value="CREATE VARIABLE"/>

2. В конвейере CD настройте входные свойства для автоматической вставки параметров Docker во время работы конвейера.



3. Создайте веб-перехватчик Docker.
 - а) Нажмите **Триггеры > Docker**.
 - б) Щелкните **Новый веб-перехватчик для Docker**.
 - в) Выберите проект.
 - г) Введите соответствующее имя.
 - д) Выберите конечную точку реестра Docker.

Если конечная точка еще не существует, щелкните **Создать конечную точку** и создайте ее.

- е) Выберите конвейер со вставленными параметрами Docker для запуска веб-перехватчика. См. раздел [Шаг 2](#).

Если конфигурация конвейера включает в себя настраиваемые входные параметры, в списке «Входные параметры» отображаются параметры и значения. Можно ввести значения для входных параметров, которые будут переданы в конвейер с событием триггера. Кроме того, можно оставить значения пустыми или использовать значения по умолчанию, если они заданы.

Дополнительные сведения о параметрах на вкладке входных данных см. в разделе [Планирование собственной сборки по модели непрерывной интеграции и предоставления в Code Stream перед добавлением задач вручную](#).

ж) Введите маркер API-интерфейса.

Маркер API-интерфейса CSP выполняет проверку подлинности для внешних соединений API-интерфейса с Code Stream. Чтобы получить маркер API-интерфейса, выполните следующие действия.

1. Выберите **Создать маркер**.
2. Введите адрес электронной почты, связанный с именем пользователя и паролем, а затем нажмите **Создать**.

Созданный маркер действителен в течение шести месяцев. Он также называется маркером обновления.

- Чтобы сохранить маркер в виде переменной для использования в будущем, щелкните **Создать переменную**, введите имя переменной и нажмите кнопку **Сохранить**.
- Чтобы сохранить маркер как текстовое значение для использования в будущем, нажмите кнопку **Копировать** и вставьте маркер в текстовый файл для сохранения в локальной системе.

Можно создать переменную и одновременно сохранить маркер в текстовом файле для дальнейшего использования.

3. Нажмите кнопку **Заккрыть**.

з) Введите образ сборки.

и) Введите тег.

Docker

Activity **Webhooks for Docker**

Webhook URL [?] `https://[redacted]m/codestream/api/registry-webhook-listeners/54bd030d-`

Project `test`

Name * `sm-1-Docker-WH`

Description `Docker webhook trigger for sm-1`

Docker Registry `Docker-Register-Endpoint`

Pipeline * `sm-1` [?]

API token * `[redacted]` [?] CREATE VARIABLE GENERATE TOKEN

Image [?] `Image`

Tag [?] `Tags`

SAVE CANCEL

к) Нажмите **Сохранить**.

Открывается карточка веб-перехватчика с включенным веб-перехватчиком Docker. Если требуется выполнить пробную отправку в репозиторий центра Docker без запуска веб-перехватчика Docker и выполнения конвейера, щелкните **Отключить**.

4. В конвейере CD настройте задачу развертывания Kubernetes.

- а) В свойствах задачи Kubernetes выберите кластер разработки Kubernetes.
- б) Выберите действие **Создать**.

- в) Выберите **Локальное определение** для источника полезных данных.
- г) Затем выберите локальный файл YAML.

Например, центр Docker может отправить это локальное определение YAML в виде полезных данных на URL-адрес в веб-перехватчике:

```
{
  "callback_url": "https://registry.hub.docker.com/u/svendowideit/testhook/hook/2141b5bi5i5b02bec211i4eeih0242eg11000a/",
  "push_data": {
    "images": [
      "27d47432a69bca5f2700e4dff7de0388ed65f9d3fb1ec645e2bc24c223dc1cc3",
      "51a9c7c1f8bb2fa19bcd09789a34e63f35abb80044bc10196e304f6634cc582c",
      "...",
    ],
    "pushed_at": 1.417566161e+09,
    "pusher": "trustedbuilder",
    "tag": "latest"
  },
  "repository": {
    "comment_count": 0,
    "date_created": 1.417494799e+09,
    "description": "",
    "dockerfile": "#\n# BUILD\u0009\u0009docker build -t svendowideit/apt-cacher .\n# RUN\u0009\u0009docker run -d -p 3142:3142 -name apt-cacher-run apt-cacher\n#\n# and then you can run containers with:\n#\n\u0009\u0009docker run -t -i -rm -e http_proxy http://192.168.1.2:3142/debian bash\n#\nFROM\u0009\u0009ubuntu\n#\nVOLUME\u0009\u0009[/var/cache/apt-cacher-ng]\nRUN\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n#\nEXPOSE\n\u0009\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"
  }
}
```

В API-интерфейсе, который создает веб-перехватчик в центре Docker, используется следующая форма: https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/

Тело кода JSON выглядит следующим образом:

```
{
  "name": "demo_webhook",
}
```

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

Чтобы получать события с сервера центра Docker, в схеме проверки подлинности для веб-перехватчика Docker, созданного в Code Stream, используется механизм проверки подлинности с фильтрацией по разрешенному списку со случайным строковым маркером для веб-перехватчика. Он фильтрует события на основе защищенного маркера, который можно присоединить к `hook_url`.

Code Stream может проверять любые запросы с сервера центра Docker, используя настроенный защищенный маркер. Например, `hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""`.

5. Создайте артефакт Docker в репозитории центра Docker. Или обновите существующий артефакт.
6. Убедитесь, что триггер сработал, и просмотрите действие, инициируемое веб-перехватчиком Docker, щелкнув **Триггеры > Docker > Действие**.

Docker

GUIDED SETUP

Activity
Webhooks for Docker

Commit Time	Webhook	Image	Tag	Owner	Repository	Pipeline	Execution Status
01/09/2019 10:59 AM	dt11-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	fvxd-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	test-do-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	sm-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	t-token-Docker-WH	admin/repo:s1	s1	admin	repo		FAILED
01/09/2019 10:57 AM	dt11-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	sm-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	test-do-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	fvxd-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED

7. Щелкнув **Выполняемые элементы**, можно наблюдать за работой конвейера.

Executions

417 items
GUIDED SETUP

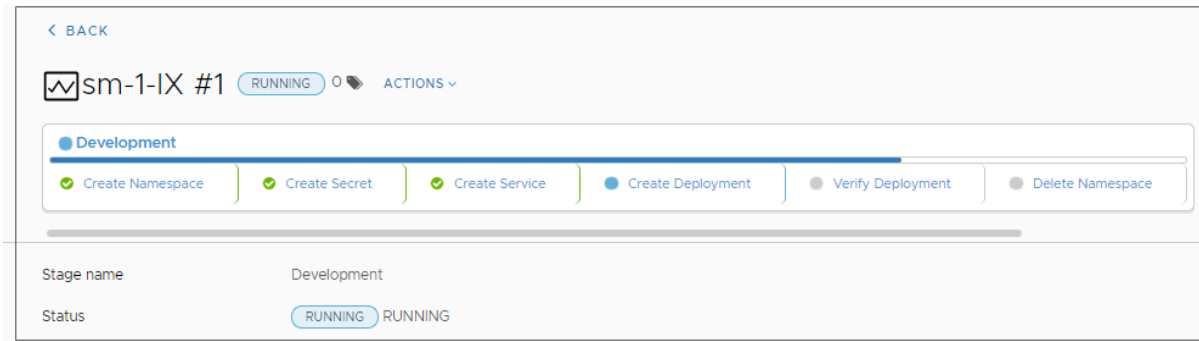
+ NEW EXECUTION
🔍
📄
🔍
🔄

sm-1-IX#1

RUNNING
Stages:
ACTIONS ▾

By k on 01/09/2019 2:41 PM
☆ Input : n/a
☆ Output : n/a

8. Щелкните выполняющийся этап и просмотрите задачи в ходе работы конвейера.



Результаты

Поздравляем! Настроен триггер Docker для непрерывного запуска конвейера непрерывного предоставления. Теперь конвейер может отправлять новые и обновленные артефакты Docker в репозиторий центра Docker.

Следующие шаги

Убедитесь, что новый или обновленный артефакт развернут на узле Docker в кластере разработки Kubernetes.

Как использовать триггер Git в Code Stream для запуска конвейера

Администратор или разработчик Code Stream может интегрировать Code Stream с процессами жизненного цикла Git с помощью триггера Git. При изменении кода в GitHub, GitLab или Bitbucket Enterprise событие обменивается данными с Code Stream через веб-перехватчик и запускает конвейер. Веб-перехватчик работает с локальными корпоративными версиями GitLab, GitHub и Bitbucket, когда Cloud Assembly и корпоративная версия доступны в одной сети.

При добавлении веб-перехватчика для Git в Code Stream также создается веб-перехватчик в репозитории GitHub, GitLab или Bitbucket. При последующем обновлении или удалении веб-перехватчика это действие также приведет к обновлению или удалению веб-перехватчика в GitHub, GitLab или Bitbucket.

В определении веб-перехватчика должна быть указана конечная точка Git в ветви репозитория, которая будет отслеживаться. Для создания веб-перехватчика в Code Stream используется конечная точка Git. Если конечная точка не существует, ее можно создать при добавлении веб-перехватчика. В этом примере предполагается, что в GitHub есть предварительно заданная конечная точка Git.

Примечание Конечная точка Git, используемая для создания веб-перехватчика, должна использовать закрытый маркер для проверки подлинности, но не может использовать пароль.

Можно создать несколько веб-перехватчиков для различных ветвей с помощью одной конечной точки Git, указав разные значения для имени ветви на странице конфигурации веб-перехватчика. Чтобы создать другой веб-перехватчик для другой ветви в том же репозитории Git, не нужно несколько раз клонировать конечную точку Git для нескольких ветвей. Просто укажите имя ветви в веб-перехватчике, чтобы использовать конечную точку Git еще раз. Если ветвь веб-перехватчика Git совпадает с ветвью в конечной точке, не нужно указывать имя ветви на странице веб-перехватчика Git.

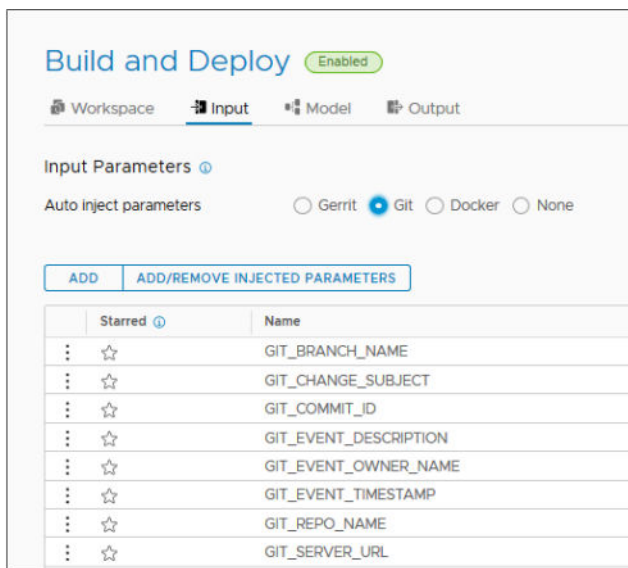
В этом примере показано, как использовать триггер Git в репозитории GitHub. При этом в предварительных условиях указано, какие подготовительные действия требуются при использовании других типов сервера Git.

Необходимые условия

- Проверьте, являетесь ли вы участником проекта в Code Stream. Если нет, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).
- Убедитесь, что существует конечная точка Git в ветви GitHub, которую необходимо отслеживать. См. раздел [Интеграция Git со службой Code Stream](#).
- Убедитесь в наличии прав на создание веб-перехватчика в репозитории Git.
- При настройке веб-перехватчика в GitLab измените параметры сети по умолчанию в GitLab Enterprise, чтобы включить исходящие запросы и разрешить создание локальных веб-перехватчиков.

Примечание Это изменение требуется только для GitLab Enterprise. Эти параметры не применяются к GitHub или Bitbucket.

- а) Войдите в экземпляр GitLab Enterprise как администратор.
- б) Перейдите к настройкам сети, используя URL-адрес, например, `http://{gitlab-server}/admin/application_settings/network`.
- в) Разверните раздел **Исходящие запросы** и выберите:
 - разрешить запросы к локальной сети от веб-перехватчиков и служб;
 - разрешить запросы к локальной сети от системного перехватчика.
- Относительно конвейеров, которые требуется запустить: убедитесь, что настроены входные свойства для вставки параметров Git во время работы конвейера.



Дополнительные сведения о входных параметрах см. в разделе [Планирование собственной сборки по модели непрерывной интеграции и предоставления в Code Stream перед добавлением задач вручную](#).

Процедура

1. В Code Stream выберите **Триггеры > Git**.
2. Перейдите на вкладку **Веб-перехватчики для Git**, а затем щелкните **Создать веб-перехватчик для Git**.
 - а) Выберите проект.
 - б) Введите понятное имя и описание для веб-перехватчика.

- в) Выберите конечную точку Git, настроенную для ветви, которую требуется отслеживать.

При создании веб-перехватчика его определение включает в себя сведения о текущей конечной точке.

- Если тип Git, тип сервера Git или URL-адрес репозитория Git в конечной точке позже будут изменены, веб-перехватчик не сможет запустить конвейер, так как будет пытаться получить доступ к репозиторию Git, используя сведения исходной конечной точки. Необходимо удалить веб-перехватчик и снова создать его, указав конечную точку.
- Если позднее в конечной точке будет изменен тип проверки подлинности, имя пользователя или закрытый маркер, веб-перехватчик продолжит работать.
- При использовании репозитория BitBucket его URL-адрес должен быть в формате `https://api.bitbucket.org/{user}/{repo name}` или `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`.

Примечание Если веб-перехватчик, использующий конечную точку Git, где для базовой проверки подлинности применяется пароль, уже был создан раньше, необходимо удалить веб-перехватчик и переопределить его с конечной точкой Git, использующей закрытый маркер для проверки подлинности.

См. раздел [Интеграция Git со службой Code Stream](#).

- г) (дополнительно) Укажите ветвь, которую требуется отслеживать с помощью веб-перехватчика.

Если ветвь не указана, веб-перехватчик отслеживает ветвь, настроенную для конечной точки Git.

- д) (дополнительно) Создайте секретный маркер для веб-перехватчика.

При использовании секретного маркера Code Stream создает случайный строковый маркер для веб-перехватчика. Затем, когда веб-перехватчик получает данные о событии Git, он отправляет эти данные с секретным маркером. Code Stream использует информацию, чтобы определить, поступают ли вызовы из предполагаемого источника, например из настроенного экземпляра GitHub, репозитория и ветви. Секретный маркер обеспечивает дополнительный уровень безопасности, который позволяет убедиться, что данные о событии Git поступают из верного источника.

е) (дополнительно) В качестве условий триггера укажите условия включения или исключения файлов.

- Включения файлов. В случае совпадения файлов в отправленной партии с файлами, указанными в путях включения или в регулярном выражении, запускаются конвейеры. Если задано регулярное выражение, Code Stream запускает только конвейеры с именами файлов в наборе изменений, которые совпадают с заданным выражением. Фильтр регулярных выражений полезен при настройке триггера для нескольких конвейеров в одном репозитории.
- Исключения файлов. Если все файлы в отправленной партии совпадают с файлами, указанными в путях исключения или регулярном выражении, конвейеры не запускаются.
- Указание приоритета исключения. Если функция «Установка приоритета для исключения» включена, конвейеры не будут запускаться даже в том случае, если какой-либо из файлов во время фиксации будет совпадать с файлами в путях исключения или регулярном выражении. Значение по умолчанию — «Выкл.»

Если условия соответствуют как условиям включения файла, так и условиям исключения файла, конвейеры не запускаются.

В следующем примере для триггера используются условия включения и исключения файлов.

File ⓘ			
Inclusions	PLAIN	runtime/src/main/a.java	-
	REGEX	([a-z A-Z]+[/][a-z A-Z])+	- +
Exclusions	PLAIN	runtime/pom.xml	-
	PLAIN	runtime/demo.yaml	- +
Prioritize Exclusion		<input type="checkbox"/>	

- Включение файлов: при отправке кода с любым изменением в `runtime/src/main/a.java` или любом файле Java будут запускаться конвейеры, настроенные в конфигурации событий.
- Для исключений файлов: при отправке кода с изменениями только в обоих файлах конвейеры, настроенные в конфигурациях событий, не будут запускаться.

ж) Для события Git выберите запрос **Отправить** или **Получить**.

з) Введите маркер API-интерфейса.

Маркер API-интерфейса CSP выполняет проверку подлинности для внешних соединений API-интерфейса с Code Stream. Чтобы получить маркер API-интерфейса, выполните следующие действия.

1. Выберите **Создать маркер**.
2. Введите адрес электронной почты, связанный с именем пользователя и паролем, а затем нажмите **Создать**.

Созданный маркер действителен в течение шести месяцев. Он также называется маркером обновления.

- Чтобы сохранить маркер в виде переменной для использования в будущем, щелкните **Создать переменную**, введите имя переменной и нажмите кнопку **Сохранить**.
- Чтобы сохранить маркер как текстовое значение для использования в будущем, нажмите кнопку **Копировать** и вставьте маркер в текстовый файл для сохранения в локальной системе.

Можно создать переменную и одновременно сохранить маркер в текстовом файле для дальнейшего использования.

3. Нажмите кнопку **Заккрыть**.

и) Выберите конвейер, который должен запускаться веб-перехватчиком.

Если конвейер включает в себя добавленные настраиваемые входные параметры, в списке «Входные параметры» отображаются параметры и значения. Можно ввести значения для входных параметров, которые передаются в конвейер с событием триггера. Кроме того, можно оставить значения пустыми или использовать установленные по умолчанию значения, если они заданы.

Дополнительные сведения о настройках автоматической вставки входных параметров для триггеров Git см. в разделе [Предварительные условия](#).

к) Щелкните **Создать**.

Веб-перехватчик отобразится в виде новой карточки.

3. Щелкните карточку веб-перехватчика.

Когда форма данных веб-перехватчика появится снова, в верхней ее части будет отображаться добавленный URL-адрес веб-перехватчика. Веб-перехватчик Git подключается к репозиторию GitHub с помощью URL-адреса веб-перехватчика.

Git

Activity

Webhooks for Git

Webhook URL ⓘ

https://ca811b68-1b3c-4309-9912-bf6178217b3e/codestream/api/git-webhook-listeners/963b2287-527f-4e9b-b00e-400000000000

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

⌵

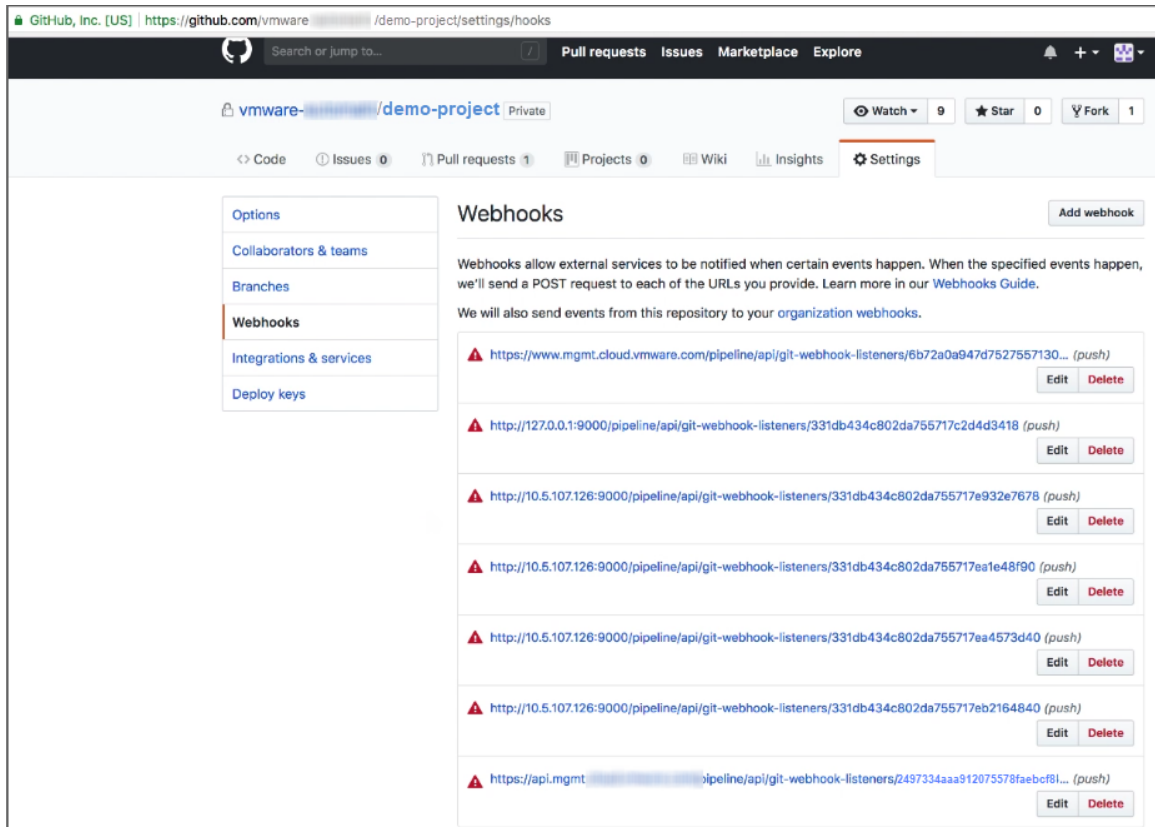
SAVE

CANCEL

4. В новом окне браузера откройте репозиторий GitHub, подключенный с помощью веб-перехватчика.

- а) Для отображения веб-перехватчика, добавленного в Code Stream, перейдите на вкладку **Параметры** и выберите **Веб-перехватчики**.

В нижней части списка веб-перехватчиков появится тот же самый URL-адрес веб-перехватчика.



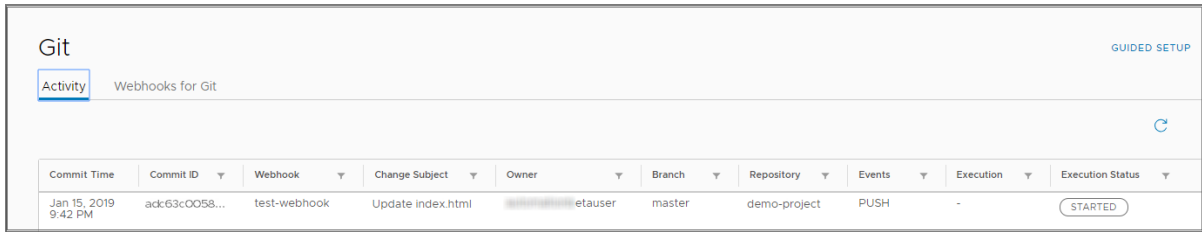
- б) Чтобы внести изменения в код, перейдите на вкладку **Код** и выберите файл в ветви. После внесения изменений в файл зафиксируйте их.
- в) Чтобы убедиться, что URL-адрес веб-перехватчика работает, перейдите на вкладку **Настройки** и снова выберите **Веб-перехватчики**.

В нижней части списка веб-перехватчиков рядом с URL-адресом веб-перехватчика появится зеленая галочка.



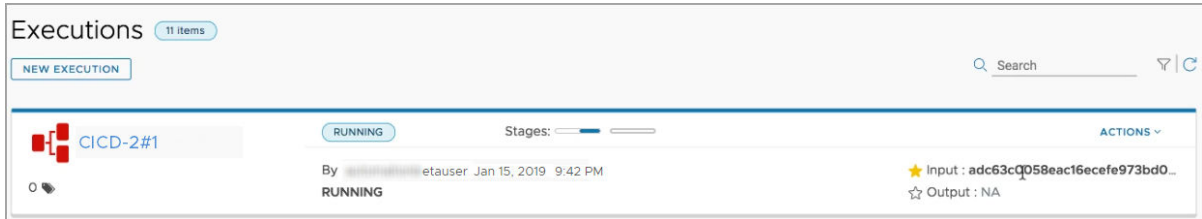
5. Чтобы просмотреть действие, инициируемое веб-перехватчиком Git, вернитесь в Code Stream. Щелкните **Триггеры > Git > Действие**.

В разделе «Состояние выполнения» убедитесь, что конвейер запущен.



6. Щелкните **Выполняемые элементы** и проследите за выполнением конвейера.

Для наблюдения за выполнением конвейера нажмите «Обновить».



Результаты

Поздравляем! Вы успешно использовали триггер для Git.

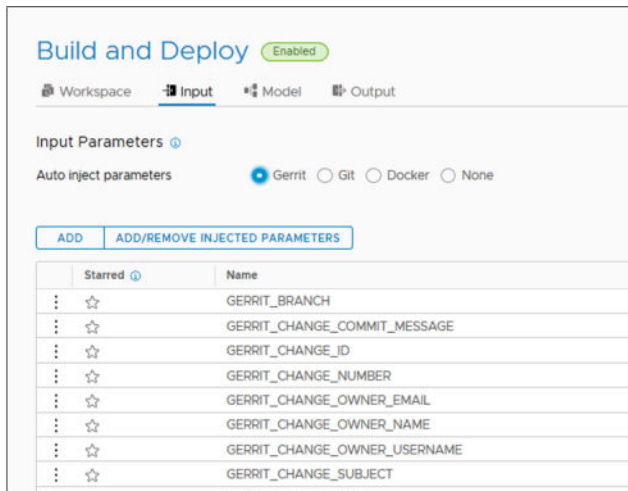
Как использовать триггер Gerrit в Code Stream для запуска конвейера

Администратор Code Stream или разработчик может интегрировать Code Stream с процессом проверки кода Gerrit на протяжении всего жизненного цикла с помощью триггера Gerrit. Событие запускает конвейер при создании набора исправлений, публикации черновиков, объединении изменений кода в проекте Gerrit или прямой отправке изменений в ветвь Git.

При добавлении триггера Gerrit нужно выбрать прослушиватель Gerrit, проект Gerrit на сервере Gerrit и настроить события Gerrit. В этом примере сначала настраивается прослушиватель Gerrit, который затем используется в триггере Gerrit с двумя событиями для трех разных конвейеров.

Необходимые условия

- Проверьте, являетесь ли вы участником проекта в Code Stream. Если нет, попросите администратора Code Stream добавить вас в проект в качестве участника. См. раздел [Добавление проекта в Code Stream](#).
- Убедитесь, что существует конечная точка Gerrit, настроенная в Code Stream. См. раздел [Интеграция Gerrit со службой Code Stream](#).
- Убедитесь, что вы знаете версию выпуска Gerrit.
- Чтобы обеспечить запуск конвейеров, убедитесь, что входные свойства конвейера заданы как **Gerrit**. Это позволит конвейеру получать параметры Gerrit в качестве входных данных во время работы конвейера.



Дополнительные сведения о входных параметрах см. в разделе [Планирование собственной сборки по модели непрерывной интеграции и предоставления в Code Stream](#) перед добавлением задач вручную.

Процедура

1. В Code Stream щелкните **Триггеры > Gerrit**.
2. (дополнительно) Перейдите на вкладку **Прослушиватели** и щелкните **Создать прослушиватель**.

Примечание Пропустите этот шаг, если прослушиватель Gerrit, который планируется использовать для триггера Gerrit, уже определен.

- а) Выберите проект.
- б) Введите имя прослушивателя Gerrit.
- в) Выберите конечную точку Gerrit.

- г) Введите маркер API-интерфейса.

Маркер API-интерфейса CSP выполняет проверку подлинности для внешних соединений API-интерфейса с Code Stream. Чтобы получить маркер API-интерфейса, выполните следующие действия.

1. Выберите **Создать маркер**.
2. Введите адрес электронной почты, связанный с именем пользователя и паролем, а затем нажмите **Создать**.

Созданный маркер действителен в течение шести месяцев. Он также называется маркером обновления.

- Чтобы сохранить маркер в виде переменной для использования в будущем, щелкните **Создать переменную**, введите имя переменной и нажмите кнопку **Сохранить**.
- Чтобы сохранить маркер как текстовое значение для использования в будущем, нажмите кнопку **Копировать** и вставьте маркер в текстовый файл для сохранения в локальной системе.

Можно создать переменную и одновременно сохранить маркер в текстовом файле для дальнейшего использования.

3. Нажмите кнопку **Заккрыть**.

Если была создана переменная, в маркере API-интерфейса отображается введенное имя переменной с помощью привязки в виде значка доллара. Если был скопирован маркер, в маркере API-интерфейса отображается маскированный маркер.

The screenshot shows the 'Gerrit' configuration interface with the 'Listeners' tab selected. The form contains the following fields and controls:

- Project**: A text input field containing 'test1' with a clear button (X).
- Name**: A text input field containing 'Gerrit-Demo-Listener'.
- Endpoint**: A dropdown menu showing 'corporate-gerrit'.
- API token**: A text input field containing '\$var.CSuser API Token' with a green checkmark icon to its right.
- Buttons**: At the bottom, there are three buttons: 'CREATE' (blue), 'VALIDATE' (light blue), and 'CANCEL' (light blue). To the right of the API token field, there are two more buttons: 'CREATE VARIABLE' (light blue) and 'GENERATE TOKEN' (light blue).

- д) Чтобы проверить сведения о маркере и конечной точке, щелкните **Проверить**.

Срок действия маркера истекает через 90 дн.

- е) Щелкните **Создать**.
- ж) На карточке прослушивателя щелкните **Подключить**.

Прослушиватель начинает мониторинг всех действий на сервере Gerrit и прослушивает все включенные триггеры на этом сервере. Чтобы остановить прослушивание триггера на этом сервере, данный триггер необходимо деактивировать.

Примечание Чтобы обновить конечную точку Gerrit, подключенную к прослушивателю, сначала необходимо отключить прослушиватель.

- Выберите **Настроить > Триггеры > Gerrit**.
 - Перейдите на вкладку **Прослушиватели**.
 - Нажмите кнопку **Отключить** для прослушивателя, подключенного к конечной точке, которую необходимо обновить.
-

3. Перейдите на вкладку **Триггеры** и щелкните **Создать триггер**.

4. Выберите проект на сервере Gerrit.

5. Введите имя.

Имя триггера должно быть уникальным.

6. Выберите настроенный прослушиватель Gerrit.

Благодаря использованию прослушивателя Gerrit Code Stream предоставляет список проектов Gerrit, доступных на сервере.

7. Выберите проект на сервере Gerrit.

8. Укажите ветвь в репозитории, которую будет отслеживать прослушиватель Gerrit.

9. (дополнительно) В качестве условий триггера укажите условия включения или исключения файлов.

- Укажите условия включения файлов, которые будут запускать конвейеры. В случае совпадения какого-либо из файлов в отправленной партии с файлами, указанными в путях включения или в регулярном выражении, выполняется запуск конвейеров. Если задано регулярное выражение, Code Stream запускает только конвейеры с именами файлов в наборе изменений, которые совпадают с заданным выражением. Фильтр регулярных выражений полезен при настройке триггера для нескольких конвейеров в одном репозитории.
- Чтобы предотвратить запуск конвейеров, необходимо указать исключения файлов. Если все файлы в отправленной партии совпадают с файлами, указанными в путях исключения или регулярном выражении, конвейеры не запускаются.
- Если параметр **Указание приоритета исключения** включен, это гарантирует, что конвейеры не будут запускаться. Конвейеры не будут запускаться даже в том случае, если какой-либо из файлов в отправленной партии совпадает с файлами, указанными в путях исключения или регулярном выражении. По умолчанию параметр **Указание приоритета исключения** выключен.

Если условия соответствуют как условию включения файла, так и условию исключения файла, конвейеры не запускаются.

В следующем примере для триггера используются условия включения и исключения файлов.

File ⓘ			
Inclusions	PLAIN	▼	runtime/src/main/a.java -
	REGEX	▼	([a-z A-Z]+[/])[a-z A-Z])+ - +
Exclusions	PLAIN	▼	runtime/pom.xml -
	PLAIN	▼	runtime/demo.yaml - +
Prioritize Exclusion		<input type="checkbox"/>	

- Включение файлов: при отправке кода с любым изменением в `runtime/src/main/a.java` или любом файле Java будут запускаться конвейеры, настроенные в конфигурации событий.
- Исключения файлов: при отправке кода с изменениями только в обоих файлах конвейеры, настроенные в конфигурации событий, не будут запускаться.

10. Щелкните **Создать конфигурацию**.

- а) Для события Gerrit выберите тип **Набор исправлений создан**, **Черновик опубликован** или **Изменение объединено**. Или для прямой отправки в Git, минуя Gerrit, выберите **Прямая отправка в Git**.

Примечание Начиная с версии 2.15 выпуска Gerrit, черновики изменений и наборы черновиков изменений больше не поддерживаются. Поэтому если вы используете выпуск Gerrit версии 2.15 или более поздней, событие **Опубликованный черновик** не поддерживается.

- б) Выберите конвейер, который будет запускаться.

Если конвейер включает в себя добавленные настраиваемые входные параметры, в списке «Входные параметры» отображаются параметры и значения. Можно ввести значения для входных параметров, которые требуется передать в конвейер с событием триггера. Кроме того, можно оставить значения пустыми или использовать значения по умолчанию.

Примечание Если значения по умолчанию определены:

- Любые вводимые значения входных параметров будут переопределять значения по умолчанию, заданные в модели конвейера.
- Значения по умолчанию в конфигурации триггера не будут изменяться, если будут изменены значения параметров в модели конвейера.

Дополнительные сведения о настройках автоматической вставки входных параметров для триггеров Gerrit см. в разделе [Предварительные условия](#).

- в) Для типов **Набор исправлений создан**, **Черновик опубликован** и **Изменение объединено**, некоторые действия по умолчанию появляются с метками. Можно изменить метку или добавить комментарии. При запуске конвейера метка или комментарий отображаются на вкладке **Действия** данного конвейера как **Выполненное действие**.

Конфигурация события Gerrit позволяет вводить комментарии с помощью переменной для комментария «Успешно» или «Сбой». Например: `${var.success}` и `${var.failure}`.

- г) Нажмите **Сохранить**.

Чтобы добавить несколько событий триггеров в нескольких конвейерах, снова щелкните **Создать конфигурацию**.

В следующем примере показаны события для трех конвейеров:

- Если в проекте Gerrit происходит событие **Изменение объединено**, запускается конвейер с именем **Конвейер Gerrit**.
- Если в проекте Gerrit происходит событие **Набор исправлений создан**, запускаются **Конвейер триггера Gerrit** и **Демоконвейер Gerrit**.

Gerrit

GUIDED SETUP

Activity
Triggers
Listeners

Project *

test1 ×

Name *

Gerrit-Demo-Trigger

Gerrit Listener *

Gerrit-Demo-Listener ▼

Gerrit project *

Gerrit-Demo-Project ▼

Branch *

master

File ⓘ

Inclusions

-- Select Type -- ▼

value

+

Exclusions

+

Prioritize Exclusion

+ NEW CONFIGURATION

	Event Type	Pipeline	Label
⋮	Change Merged	Gerrit-Pipeline	Verified
⋮	Patchset Created	Gerrit-Trigger-Pipeline	Verified
⋮	Patchset Created	Gerrit-Demo-Pipeline	Verified
3 configurations			

11. Щелкните **Создать**.

Триггер Gerrit отображается в виде новой карточки на вкладке **Триггеры**, и ему по умолчанию назначается значение **Отключен**.

12. На карточке триггера щелкните **Включить**.

После включения триггера он может использовать прослушиватель Gerrit, который запускает мониторинг событий, происходящих в ветви проекта Gerrit.

Чтобы создать триггер, который имеет такие же условия включения или исключения файлов, но с другим репозиторием, который отличается от репозитория, добавленного при создании триггера, на карточке триггера нажмите **Действия > Клонировать**. Затем в клонированном триггере нажмите **Открыть** и измените параметры.

Результаты

Поздравляем! Вы успешно настроили триггер Gerrit с двумя событиями в трех разных конвейерах.

Следующие шаги

После фиксации изменения кода в проекте Gerrit проверьте на вкладке **Действие** наличие события Gerrit в Code Stream. Убедитесь, что список действий включает в себя записи, соответствующие каждому циклу выполнения конвейера в конфигурации триггера.

При возникновении события могут запускаться только те конвейеры в триггере Gerrit, которые относятся к конкретному типу события. В этом примере, если создан набор исправлений, будут выполняться только **Конвейер триггера Gerrit** и **Демо-конвейер Gerrit**.

Сведения в столбцах на вкладке **Действие** описывают каждое событие триггера Gerrit. Чтобы выбрать столбцы для отображения, щелкните значок столбца под таблицей.

- Столбцы **Изменение темы** и **Выполнение** являются пустыми, если в качестве триггера используется «Прямая отправка в Git».
- В столбце **Триггер Gerrit** отображается триггер, создавший событие.
- По умолчанию флажок столбца **Прослушиватель** снят. Если этот флажок установлен, в данном столбце отображается прослушиватель Gerrit, получивший событие. Один прослушиватель может отображаться как связанный с несколькими триггерами.
- По умолчанию флажок столбца **Тип триггера** снят. Если этот флажок установлен, в столбце будет отображаться тип триггера «АВТОМАТИЧЕСКИЙ» или «ВРУЧНУЮ».
- Другие столбцы: **Время фиксации**, **Ном. изменения**, **Состояние**, **Сообщение**, **Действия**, **Пользователь**, **Проект Gerrit**, **Ветвь** и **Событие**.

Gerrit GUIDED SETUP

Activity Triggers Listeners

TRIGGER MANUALLY ⓘ

	Commit Time	Change#	Change Subject	Execution	Status	Message	Action taken	User	Gerrit project	Gerrit Trigger	Branch	Event
⋮	Nov 12, 2019, 12:47:53 PM	19570 /4	111Dummy	Gerrit-Pipeline #1	COMPLETED	Execution Completed.	Verified +1	Drivank dpryanin@gm	test1	Gerrit-Demo-Trigger	master	Change Merged
⋮	Nov 12, 2019, 12:50:04 PM	19570 /6	11111Dummy	Gerrit-Pipeline #2	WAITING	Stage0.Task0: Execution Waiting for User Action.		Drivank dpryanin@gm	test1	Gerrit-Demo-Trigger	master	Change Merged
			11111Dummy	Gerrit-Demo-Pipeline #1	FAILED	Stage0.Task0: User Operation request has been rejected by Fritz.	Verified -1	Drivank dpryanin@gm	test1	Gerrit-Demo-Trigger	master	Patchset created
			11111Dummy	Gerrit-Trigger-Pipeline #1	WAITING	Stage0.Task0: Execution Waiting for User Action.		Drivank dpryanin@gm	test1	Gerrit-Demo-Trigger	master	Patchset created

Show columns

- ☐ Change
- ☒ Change Subject
- ☒ Execution
- ☒ Status
- ☒ Message
- ☒ Action taken
- ☒ User
- ☒ Gerrit project
- ☒ Gerrit Trigger
- ☐ Listener
- ☒ Branch
- ☒ Event
- ☐ Trigger Type

SELECT ALL

Items per page: 20 1 - 4 of 4 items

Для контроля успешного или неудачного цикла выполнения конвейера щелкните три точки слева от любой записи на экране «Действия».

- Если конвейер не выполняется из-за ошибки в модели конвейера или из-за другой проблемы, исправьте соответствующую ошибку и нажмите **Выполнить заново**, чтобы запустить конвейер снова.
- Если конвейер не выполняется из-за сбоя подключения к сети или из-за другой проблемы, нажмите **Возобновить**, чтобы перезапустить тот же цикл выполнения конвейера и сократить время выполнения.
- Нажмите **Просмотреть выполнение**, чтобы открыть представление выполнения конвейера. См. раздел [Запуск конвейера и просмотр результатов](#).
- Для удаления записи на экране «Действия» нажмите **Удалить**.

Если событие Gerrit не запускает конвейер, нажмите **Запуск вручную**, а затем выберите триггер Gerrit, введите «ИД изменения» и нажмите **Запустить**.

Мониторинг конвейеров в Code Stream



Разработчикам и администраторам Code Stream требуются данные о производительности конвейеров в Code Stream. Им необходимо оценивать эффективность конвейеров по выпуску кода на всех стадиях: разработки, тестирования и переноса в производственную среду.

Чтобы получить подробные сведения, можно использовать панели управления Code Stream для отслеживания тенденций и результатов выполнения конвейера. Можно использовать панели управления по умолчанию для отслеживания отдельного конвейера или создать настраиваемые панели управления для отслеживания нескольких конвейеров.

- Показатели конвейера включают в себя статистику, например среднее время выполнения операций, отображаемое на панели управления конвейера.
- Для отображения показателей разных конвейеров используйте настраиваемые панели управления.

В эту главу входят следующие разделы:

- [Элементы, отображаемые на панели управления конвейера в Code Stream](#)
- [Отслеживание ключевых показателей эффективности конвейера в Code Stream с помощью настраиваемых панелей управления](#)

Элементы, отображаемые на панели управления конвейера в Code Stream

На панели управления конвейера отображаются результаты для конкретного конвейера, например тенденции, основные ошибки и успешно внесенные изменения. При создании конвейера Code Stream создает панель управления данным конвейером.

Панель управления содержит мини-приложения, в которых отображаются результаты выполнения конвейера.

Мини-приложение «Количество циклов выполнения конвейера и их состояния»

В нем отображается общее количество циклов выполнения конвейера за период времени, по состоянию: «завершено», «сбой» или «отменено». Чтобы увидеть, как изменялось состояние цикла выполнения конвейера за более длительный или короткий периоды времени, измените длительность на дисплее.

Мини-приложение «Статистика выполнения конвейера»

Статистика выполнения конвейера включает в себя среднее время восстановления, доставки или сбоя конвейера за период времени.

К циклам выполнения конвейера применяются следующие состояния:

- Завершено
- Сбой
- Ожидание
- Выполняется
- Отменено
- В очереди
- Не запущено
- Выполняется откат
- Откат завершен
- Сбой при выполнении отката
- Приостановлено

Таблица 8-1. Измерение среднего времени

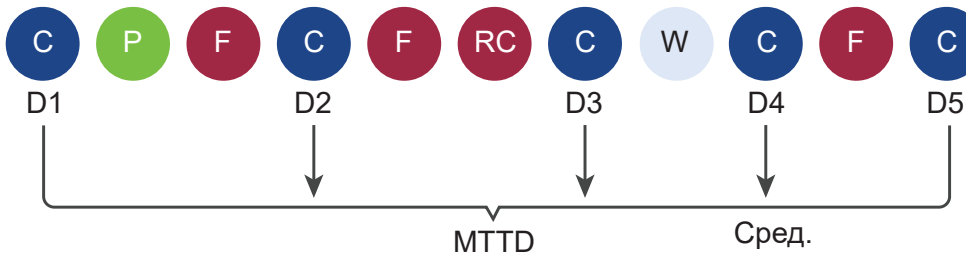
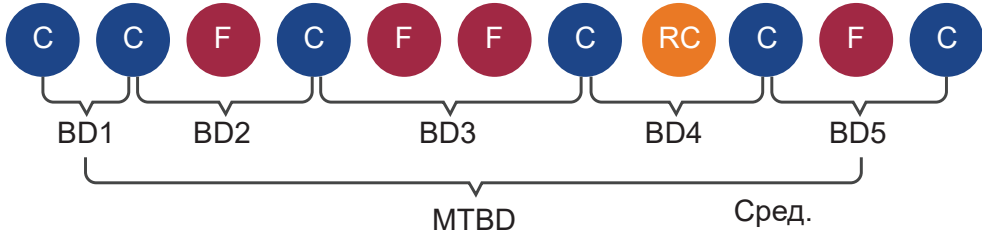
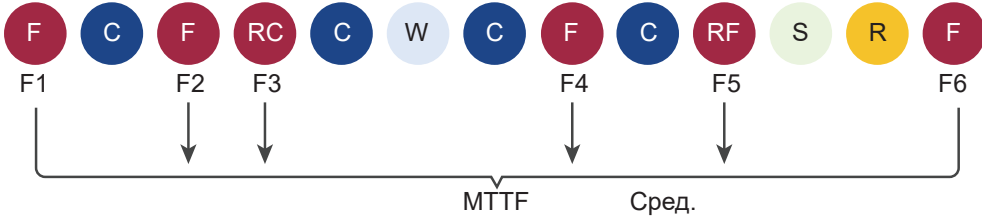
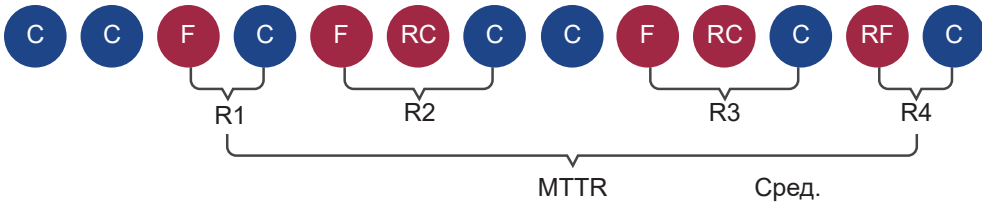
Что измеряется...	Что это означает...
Среднее значение CI	Среднее время выполнения этапа непрерывной интеграции (CI), измеряемое по времени, указанному для типа задачи CI.
Среднее время доставки (MTTD)	<p>Средняя длительность всех циклов выполнения в состоянии ЗАВЕРШЕНО за период времени. D1, D2 и так далее — это время, необходимое для доставки каждого ЗАВЕРШЕННОГО цикла выполнения.</p> 

Таблица 8-1. Измерение среднего времени (продолжение)

Что измеряется...	Что это означает...
Среднее время между доставками (MTBD)	<p>Среднее время между успешными доставками за период времени. Время между двумя последовательными ЗАВЕРШЕННЫМИ циклами выполнения — это время между успешными доставками, например BD1, BD2 и так далее. MTBD указывает на частоту обновления производственной среды.</p> 
Среднее время до отказа (MTTF)	<p>Средняя длительность циклов выполнения, которые заканчиваются в состоянии СБОЙ, ОТКАТ ЗАВЕРШЕН, СБОЙ ПРИ ВЫПОЛНЕНИИ ОТКАТА за период времени. F1, F2 и так далее — время, в течение которого цикл выполнения закончится в состоянии ОШИБКА, ОТКАТ ЗАВЕРШЕН или СБОЙ ПРИ ВЫПОЛНЕНИИ ОТКАТА.</p> 
Среднее время восстановления (MTTR)	<p>Среднее время восстановления после отказа за период времени. Время восстановления после отказа — это время между циклом выполнения с итоговым состоянием СБОЙ, ОТКАТ ЗАВЕРШЕН или СБОЙ ПРИ ВЫПОЛНЕНИИ ОТКАТА и следующим ближайшим успешным циклом выполнения с состоянием ЗАВЕРШЕНО. R1, R2 и так далее — это время восстановления после каждого цикла выполнения в состоянии СБОЙ или СБОЙ ПРИ ВЫПОЛНЕНИИ ОТКАТА.</p> 

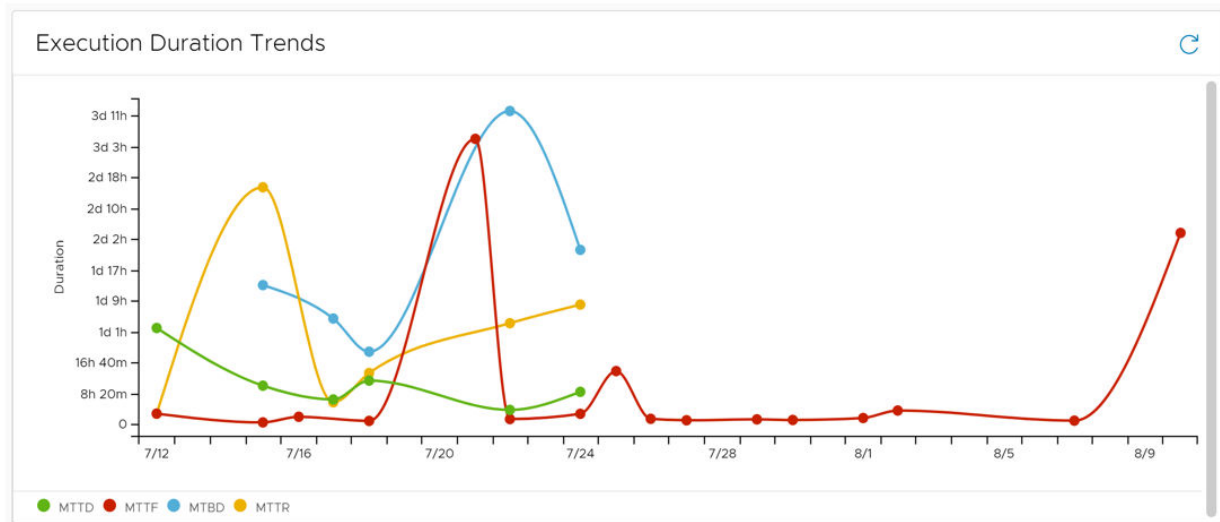
Мини-приложения «Этапы с наиболее высокими показателями невыполнения» и «Задачи с наиболее высокими показателями невыполнения»

В этих двух мини-приложениях отображаются этапы и задачи с наиболее высокими показателями невыполнения в конвейере. Каждый показатель дает сведения о количестве и проценте сбоев в средах разработки и последующей обработки для каждого конвейера и проекта, усредненные за неделю или месяц. Чтобы устранить проблемы, возникающие в процессе автоматизации выпуска, можно просмотреть наиболее серьезные ошибки.

Можно настроить отображение за определенный период, например за последнюю неделю, и отметить основные задачи со сбоем за этот период. После внесения изменений в среду или конвейер и повторного запуска конвейера проверьте основные задачи со сбоем за более длительный период времени, например за последние 14 дней. Список таких задач может измениться. Благодаря этому результату станет понятно, что изменение процесса автоматизации выпуска повысило уровень успешного выполнения конвейера.

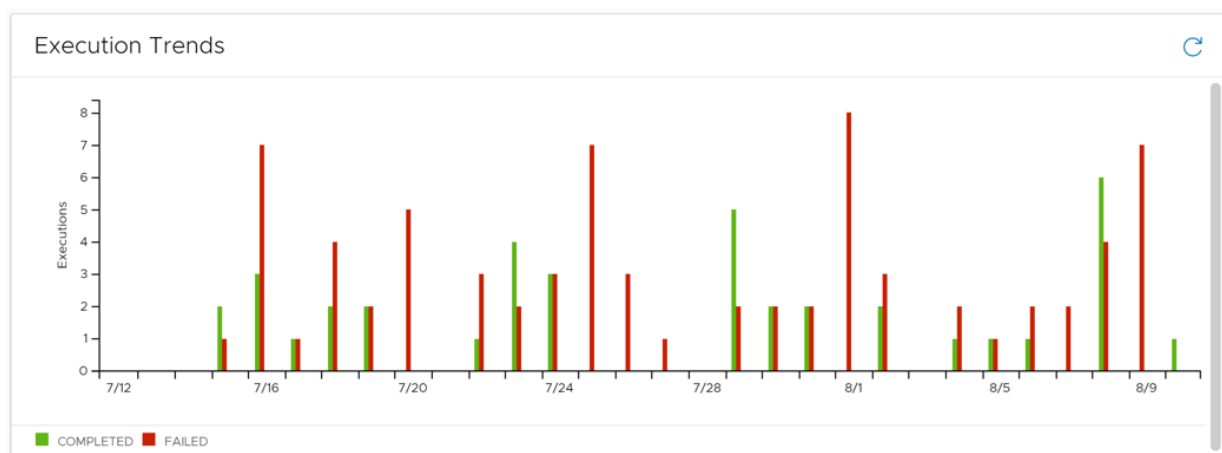
Мини-приложение «Тенденции длительности выполнения конвейера»

В этом мини-приложении отображаются значения MTDD, MTTF, MTBD и MTTR за период времени.



Мини-приложение «Тенденции выполнения конвейера»

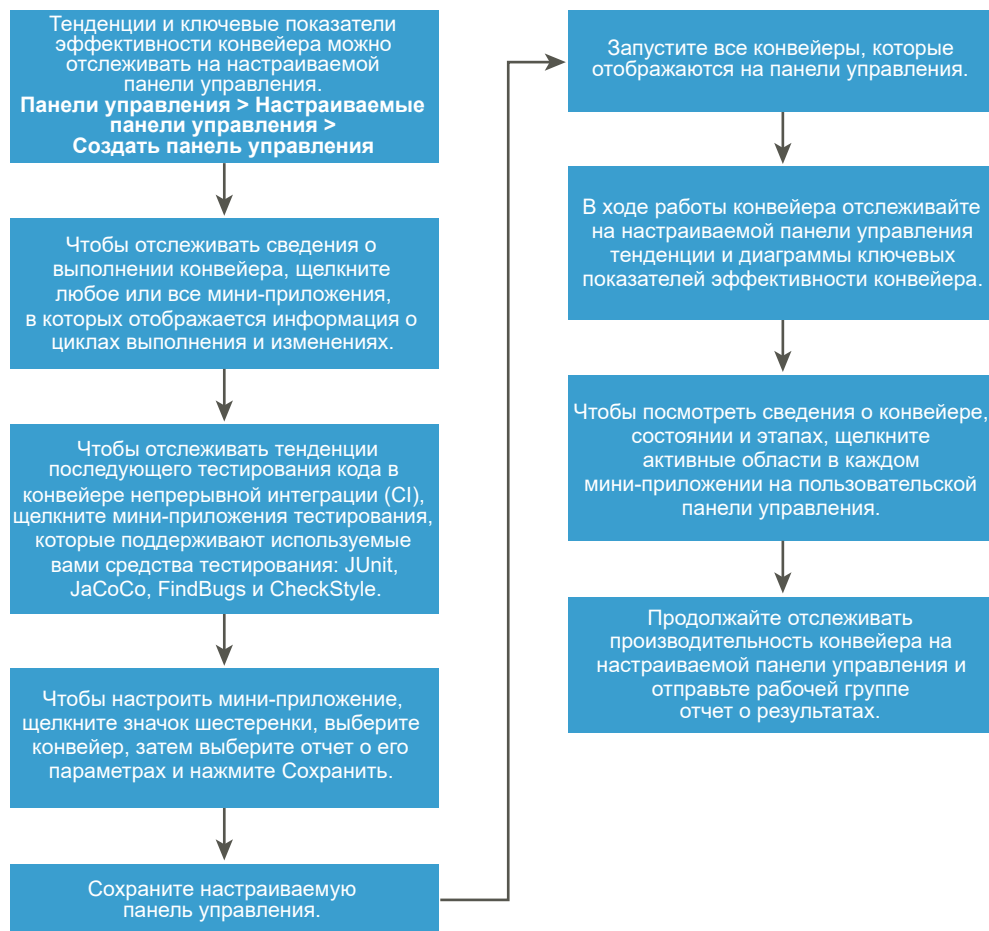
В этом мини-приложении отображается общее количество ежедневных циклов выполнения конвейера, по состоянию за период времени. За исключением текущего дня, большинство сводных количественных значений за день показывают только циклы выполнения в состоянии ЗАВЕРШЕНО и СБОЙ.



Отслеживание ключевых показателей эффективности конвейера в Code Stream с помощью настраиваемых панелей управления

Администратор Code Stream или разработчик может создать настраиваемую панель управления для отображения необходимых результатов для одного или нескольких работающих конвейеров. Например, можно создать панель управления для всего проекта с ключевыми показателями эффективности и показателями, получаемыми из нескольких конвейеров. При возникновении ошибки или предупреждения о сбое конвейера панель управления можно использовать для устранения проблемы.

Для отслеживания тенденций и ключевых показателей эффективности конвейеров с помощью настраиваемой панели управления необходимо добавить мини-приложения на эту панель и настроить в них создание отчетов о работе конвейеров.



Необходимые условия

- Убедитесь, что существует один или несколько конвейеров. В пользовательском интерфейсе щелкните **Конвейеры**.
- Убедитесь, что конвейеры, которые планируется отслеживать, работают успешно. Щелкните **Выполняемые элементы**.

Процедура

1. Чтобы создать настраиваемую панель управления, щелкните **Панели управления > Настраиваемые панели управления > Создать панель управления**.
2. Чтобы настроить на панели управления создание отчетов о конкретных тенденциях и ключевых показателях эффективности конвейера, щелкните мини-приложение.

Например, для отображения сведений о состоянии конвейера, его этапах, задачах, времени работы и пользователях, которые его запускали, щелкните мини-приложение **Сведения о выполнении**. Помимо этого, для конвейера непрерывной интеграции (CI) можно отслеживать тенденции последующей обработки с помощью мини-приложений для JUnit, JaCoCo, FindBugs и CheckStyle.

IX KPIs								
EDIT DELETE CLONE BACK								
Execution Details								
	Execution ID	Execution#	Status	Status Message	Stages	Tasks	Task0 (Stage0)	Duration
⋮	178f62eef...	#2	WAITING	Stage0.Task0 Execution Waiting for User Action.	●	⊗	⊗	15s
⋮	5503c1e51...	#1	COMPLETED	Execution Completed.	●	✓	✓	1h 28m 7s

3. Настройте каждое добавляемое мини-приложение.
 - а) В мини-приложении щелкните значок шестеренки.
 - б) Выберите конвейер, задайте доступные параметры и выберите столбцы для отображения.
 - в) Чтобы сохранить конфигурацию мини-приложения, щелкните **Сохранить**.
 - г) Чтобы сохранить настраиваемую панель управления, щелкните **Сохранить**, а затем **Заккрыть**.
4. Чтобы отобразить дополнительные сведения о конвейере, щелкните активные области в мини-приложениях.

Например, в мини-приложении **Сведения о выполнении** щелкните запись в столбце «Состояние», чтобы отобразить дополнительные сведения о выполнении конвейера. Или в мини-приложении **Последнее успешное изменение** для отображения сведений об этапе и задаче конвейера щелкните активную ссылку.

Результаты

Поздравляем! Создана настраиваемая панель управления, которая позволяет отслеживать тенденции и ключевые показатели эффективности для конвейеров.

Следующие шаги

Продолжайте отслеживать производительность конвейеров в Code Stream, а также поделитесь полученными результатами с менеджером и группами, чтобы продолжить улучшение выпуска приложений.

Дополнительные сведения о Code Stream

9

Разработчики и администраторы Code Stream могут использовать различные способы, чтобы получить дополнительные сведения о Code Stream и его возможностях.

Эту документацию можно использовать для получения дополнительных сведений о конвейерах и их выполнении, о добавлении конечных точек и проектов и многом другом.

Ознакомьтесь с разрешениями, которые предоставляют различные роли. Узнайте, как использовать ресурсы с ограниченным доступом и настраивать обязательное утверждение в конвейерах. См. раздел [Управление пользовательским доступом и утверждениями в службе Code Stream](#).

Ознакомьтесь с возможностями поиска, позволяющими определять расположения заданий или компонентов в конвейерах, циклах выполнения или конечных точках.

В эту главу входят следующие разделы:

- [Что такое поиск в Code Stream](#)
- [Дополнительные ресурсы для администраторов и разработчиков Code Stream](#)

Что такое поиск в Code Stream

Используйте функцию поиска, чтобы найти расположение конкретных элементов или других компонентов. Например, может потребоваться найти активированные или деактивированные конвейеры, так как деактивированный конвейер не может выполняться.

Что можно искать

Можно искать в следующих областях:

- Проекты
- Конечные точки
- Конвейеры
- Выполняемые элементы
- Панели управления конвейера, настраиваемые панели управления
- Триггеры Gerrit и серверы
- Веб-перехватчики Git

- Веб-перехватчики Docker

Можно выполнять поиск по столбцам с помощью фильтра для следующих категорий.

- Пользовательские операции
- Переменные
- Действия триггеров для Gerrit, Git и Docker

На странице **Действия** можно выполнять поиск с помощью фильтра на основе сетки для каждого триггера.

Как работает функция поиска

Критерии поиска зависят от страницы, на которой вы находитесь. Для каждой страницы можно задать разные критерии поиска.

Место поиска	Критерии, используемые для поиска
Панели управления конвейера	Проект, имя, описание, теги, ссылка
Настраиваемые панели управления	Проект, имя, описание, ссылка (UUID элемента на панели управления)
Выполняемые элементы	Имя, комментарии, причина, теги, индекс, состояние, проект, показать, кем выполнено, выполнено мной, ссылка (UUID выполнения), входные параметры, выходные параметры или сообщение о состоянии в следующем формате: <key>:<value>
Конвейеры	Имя, описание, состояние, теги, кем создано, создано мной, кем обновлено, обновлено мной, проект
Проекты	Имя, описание
Конечные точки	Имя, описание, тип, кем обновлено, проект
Триггеры Gerrit	Имя, состояние, проект
Серверы Gerrit	Имя, URL-адрес сервера, проект
Веб-перехватчики Git	Имя, тип сервера, репозиторий, ветвь, проект

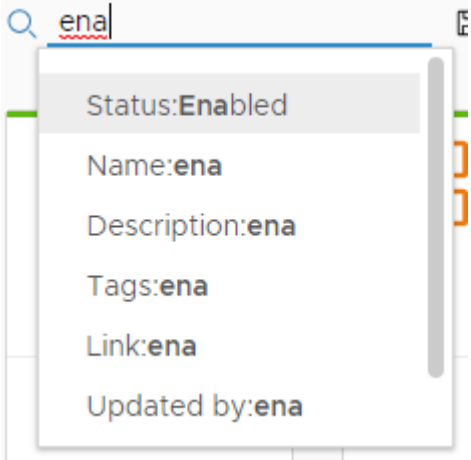
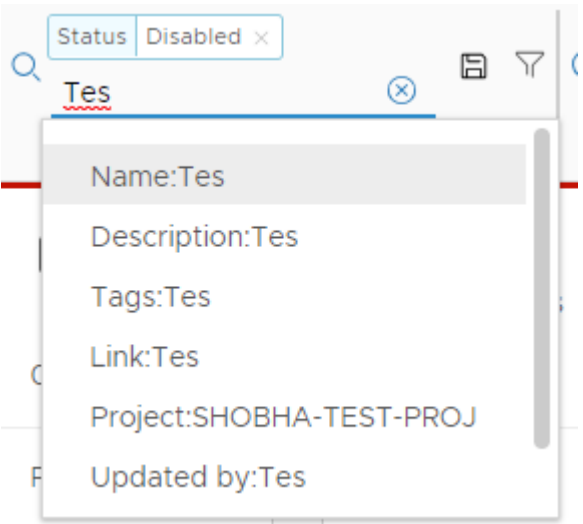
Где:

- Ссылка — это идентификатор UUID конвейера, выполнения или мини-приложения на панели управления.
- Обозначения и примеры входных параметров, выходных параметров и сообщений о состоянии:
 - Нотация: `input.<inputKey>:<inputValue>`
Пример: `input.GERRIT_CHANGE_OWNER_EMAIL:joe_user`
 - Нотация: `output.<outputKey>:<outputValue>`
Пример: `output.BuildNo:29`
 - Нотация: `statusMessage:<value>`

Пример: **statusMessage:Execution failed**

- Состояние зависит от страницы поиска.
 - Для выполнений можно использовать следующие значения: завершено, сбой, сбой_отката или отменено.
 - Для конвейеров можно использовать следующие значения состояния: включено, отключено или выпущено.
 - Для триггеров возможны следующие значения состояния: включено или отключено.
- Выполнено, создано или обновлено мною, относится ко мне, пользователь, вошедший в систему.

Поиск отображается в верхнем правом углу каждой допустимой страницы. Если начать ввод данных в пустое поле поиска, Code Stream отслеживает контекст страницы и предлагает параметры для поиска.

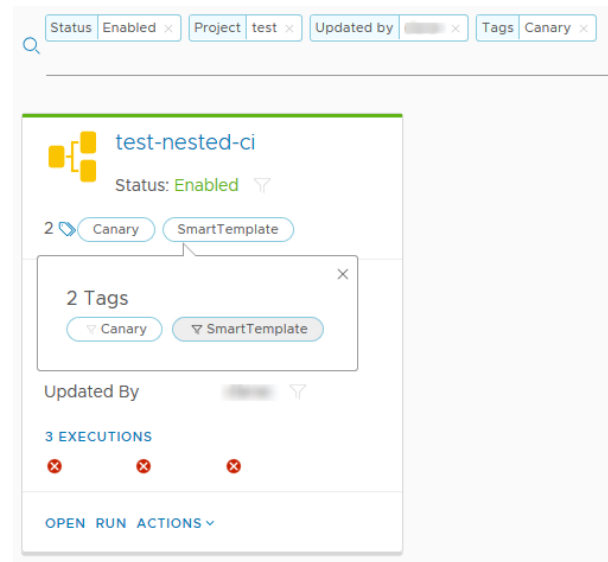
Методы, которые можно использовать для поиска	Как ввести данные
<p>Введите часть параметра поиска.</p> <p>Например, чтобы добавить фильтр состояния, позволяющий получить список всех включенных конвейеров, введите ena.</p>	
<p>Чтобы уменьшить количество найденных элементов, добавьте фильтр.</p> <p>Например, введите Tes, чтобы добавить фильтр имени. Фильтр работает как оператор И с существующим фильтром Status:disabled и показывает только деактивированные конвейеры с буквами Tes в имени.</p> <p>При добавлении другого фильтра отображаются другие возможные варианты: Имя, Описание, Теги, Связь, Проект и Кем обновлено.</p>	

Методы, которые можно использовать для поиска

Чтобы уменьшить количество отображаемых элементов, щелкните значок фильтра для свойств конвейера или его выполнения.

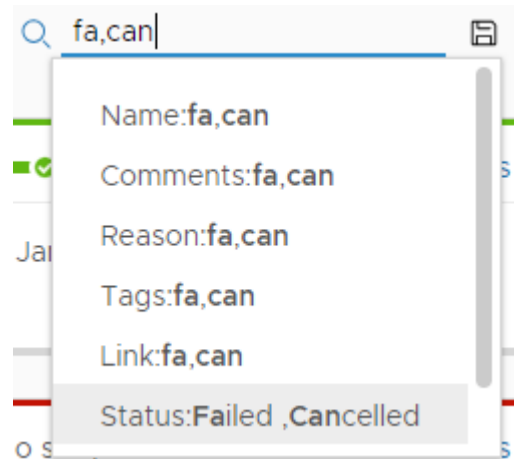
- Для конвейеров свои значки фильтра есть у свойств **Состояние**, **Теги**, **Проект** и **Кем обновлено**.
- Для выполняемых элементов свои значки фильтра есть у свойств **Теги**, **Кем выполнено** и **Сообщение о состоянии**.

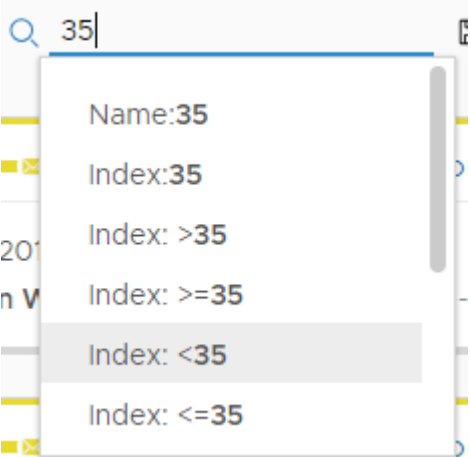
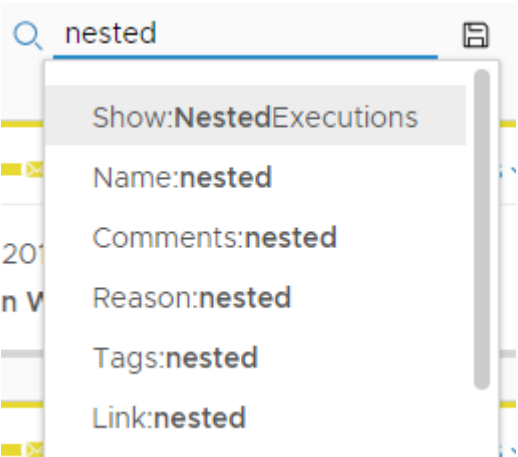
Например, на карточке конвейера щелкните значок, чтобы добавить фильтр для тега **SmartTemplate** к существующим фильтрам: **Состояние: включено**, **Проект: тест**, **Кем обновлено: пользователь** и **Теги: Canary**.

Как ввести данные

Используйте разделение запятой, чтобы включить все элементы в двух состояниях выполнения.

Например, введите **fa,can**, чтобы создать фильтр состояния, который работает как оператор **ИЛИ** для перечисления всех несостоявшихся или отмененных выполнений.

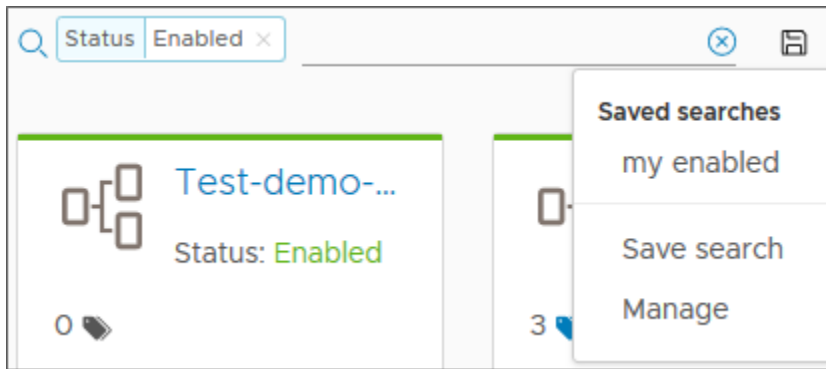


Методы, которые можно использовать для поиска	Как ввести данные
<p>Введите число, чтобы включить все элементы в диапазон индекса.</p> <p>Например, введите 35 и выберите <, чтобы получить список всех выполнений с номером индекса, который меньше 35.</p>	
<p>Конвейеры, смоделированные как задачи, становятся вложенными выполнениями и не перечисляются по умолчанию со всеми выполнениями.</p> <p>Чтобы показать вложенные выполнения, введите вложенный и выберите фильтр Показать.</p>	

Как сохранить избранный поиск

Для сохранения избранных поисковых запросов и использования их на каждой странице щелкните значок диска рядом с областью поиска.

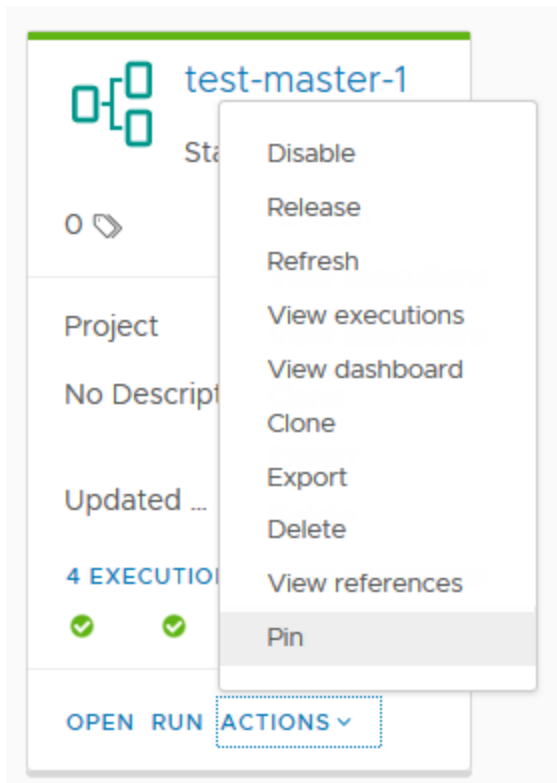
- Для сохранения поискового запроса введите параметры поиска и щелкните значок, чтобы задать для поиска имя, например, **включены мною**.
- После сохранения поискового запроса щелкните значок, чтобы получить доступ к поиску. Кроме того, можно выбрать **Управление**, чтобы переименовать, удалить или переместить поисковый запрос в список сохраненных поисковых запросов.



Поисковые запросы привязываются к имени пользователя и отображаются только на тех страницах, к которым применяется операция поиска. Например, если сохранить поисковый запрос с именем **включено мною** для **Status:enabled** на странице конвейеров, поисковый запрос **включено** будет недоступен на странице «Триггеры Gerrit», несмотря на то что состояние **Status:enabled** является допустимым поисковым запросом для триггера.

Можно ли сохранить избранный конвейер

Если у вас есть избранный конвейер или панель управления, можно закрепить этот элемент, чтобы он всегда отображался в верху списка контейнеров или страницы панелей управления. В карточке конвейера выберите пункт Действия > Закрепить.



Дополнительные ресурсы для администраторов и разработчиков Code Stream

Разработчики или администраторы Code Stream могут получить дополнительные сведения о Code Stream.

Таблица 9-1. Дополнительные ресурсы для администраторов

Для получения сведений о...	См. эти ресурсы...
<p>Другие способы использования Code Stream администраторами</p> <ul style="list-style-type: none"> ■ Настройка конвейеров для автоматизации тестирования и выпуска собственных облачных приложений. ■ Автоматизация и тестирование исходного кода разработчика в тестовой и производственной среде. ■ Настройка конвейеров для разработчиков для тестирования изменений перед их отправкой в основную ветвь. ■ Отслеживание ключевых показателей конвейера. 	<p>Code Stream</p> <ul style="list-style-type: none"> ■ Документация по продукту vRealize Automation ■ Страница продукта vRealize Automation <p>Прикладные материалы VMware</p> <ul style="list-style-type: none"> ■ Посетите сообщество vRealize Automation. ■ Посетите учебную зону VMware. ■ Почитайте блоги VMware. ■ Выполните практические занятия VMware.

Таблица 9-2. Дополнительные ресурсы для разработчиков

Для получения сведений о...	См. эти ресурсы...
<p>Другие способы использования Code Stream разработчиками</p> <ul style="list-style-type: none"> ■ Используйте общедоступные и частные образы реестра для создания сред для новых приложений или служб. ■ Настройте среду разработки таким образом, чтобы было можно создавать ветви на основе последней стабильной сборки. ■ Обновляйте среду разработки, используя новейшие изменения кода и артефакты. ■ Тестируйте незафиксированные изменения кода, используя новейшие стабильные сборки других зависимых служб. ■ Получайте уведомления, когда изменение, отправленное в основной конвейер непрерывной интеграции и непрерывного предоставления (CICD), нарушает работу других служб. 	<p>Code Stream</p> <ul style="list-style-type: none"> ■ Документация по продукту vRealize Automation ■ Страница продукта vRealize Automation <p>Прикладные материалы VMware</p> <ul style="list-style-type: none"> ■ Посетите сообщество vRealize Automation. ■ Посетите учебную зону VMware. ■ Почитайте блоги VMware. ■ Выполните практические занятия VMware.