

Практикум

Vorlon.JS: как мы внедряли DevOps



Содержание

Введение.....	3
Подготовительные шаги.....	3
Регистрация в GitHub.....	3
Создание форка Vorlon.JS.....	4
Создание учетной записи Visual Studio Team Services	6
Активация Azure pass.....	7
Обзор Visual Studio Team Services.....	8
Непрерывная интеграция с Visual Studio Team Services	8
Введение.....	8
Создание нового определения сборки.....	9
Подключение вашей учетной записи GitHub	11
Настройка триггера сборки	12
Определение процесса сборки	13
Обзор	13
Добавление шагов в процесс сборки.....	14
Настройка шагов сборки	15
Сохранение определения сборки и запуск новой	17
Просмотр отчета после сборки.....	18
Управление релизами с помощью Visual Studio Team Services	20
Введение.....	20



Создание нового определения релиза	20
Определение сред.....	21
Ссылка на описание релиза к определению сборки.....	21
Настройка триггера	22
Определение процесса релиза для каждой среды	22
Подключение подписки Azure	24
Добавление Azure classic endpoint	24
Добавление Azure Resource Manager endpoint	25
Настройка задач в среде Development	25
Создание условий развертывания для среды Development	27
Настройка задач в среде PreProduction	28
Создание условий развертывания для среды Preproduction.....	29
Сохранение определения релиза и запуск нового релиза	30



Введение

В этой практической лабораторной работе вы узнаете, как использовать Visual Studio Team Services для реализации DevOps практик, таких как непрерывная интеграция, непрерывное развертывание и управление релизами (выпусками).

Вы будете работать над реальным проектом, Vorlon.JS, который представляет собой инструмент, позволяющий удаленно отлаживать веб-приложения. Если вы уже использовали инструмент F12 в любом браузере для отладки веб-приложений, вы можете считать, что Vorlon.JS делает то же самое, но удаленно, так что вы можете сделать это на любом устройстве, например, на мобильных устройствах Windows, Android, iPhone или Xbox One.

Vorlon.JS является проектом с открытым исходным кодом, и его исходный код размещен на GitHub: <https://github.com/microsoftdx/vorlonjs~~HEAD=pobj>. Команда использует Visual Studio Team Services, чтобы обеспечивать процессы непрерывной интеграции, развертывания и управления релизами. Вы можете найти подробные материалы, описывающие их внедрение в проект Vorlon.JS [здесь](#).

В этой практической лабораторной работе вам предстоит сначала создать новый форк (fork) проекта Vorlon.JS, а затем определить процесс сборки с использованием непрерывной интеграции и создать определение для процесса управления релизами в Visual Studio Team Services, повторяя путь команды Vorlon.JS. Так как Vorlon.JS разрабатывается в Node.js, вы также узнаете, как использовать открытые веб-технологии, такие как, NPM, Gulp, Mocha.JS (и другие...) с VSTS.

Но сначала давайте проведем необходимые приготовления, чтобы вам комфортно было начать выполнение заданий!

Подготовительные шаги

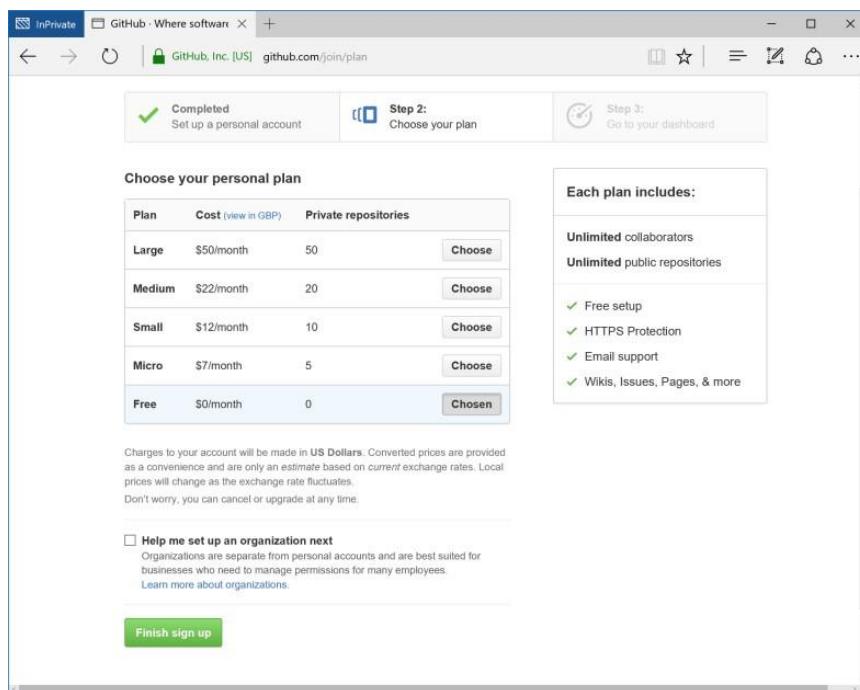
Регистрация в GitHub

Для этой лабораторной работы вам понадобится зарегистрироваться в GitHub (это бесплатно!). **Если у вас уже есть GitHub аккаунт, вы можете пропустить этот шаг.** Если у вас нет учетной записи GitHub, перейдите на <http://github.com> и введите имя пользователя, адрес электронной почты и пароль, который вы хотите использовать:





Затем кликните на **Sign up for GitHub**. На следующем экране выберите **Free**:



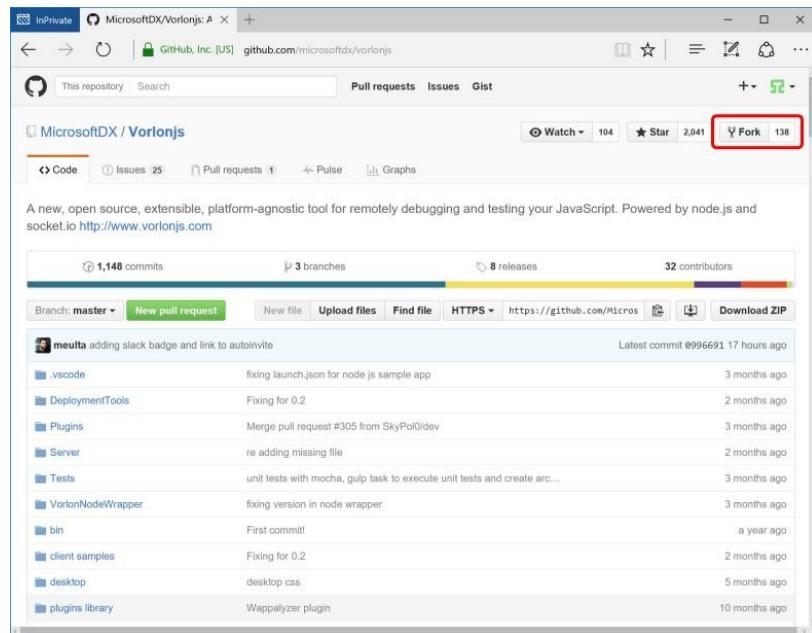
Нажмите на кнопку **Finish sign up**. Вы получите подтверждение по электронной почте, поэтому проверьте ваши входящие и кликните на кнопку **Verify email address** в письме, чтобы подтвердить свою учетную запись (вам нужно произвести это действие перед переходом к следующему шагу).

Создание форка Vorlon.JS

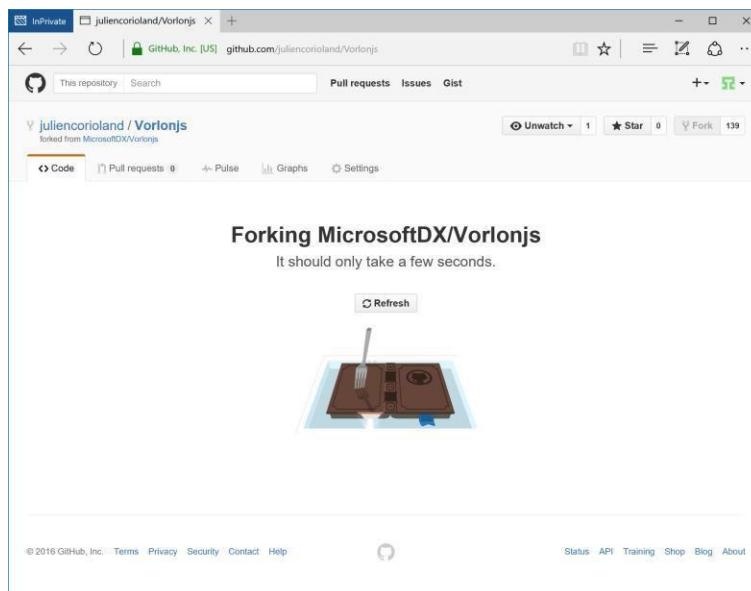
Теперь, когда у вас есть учетная запись GitHub, вы можете создать форк проекта Vorlon.JS. Создание форка продублировать проект в вашем аккаунте GitHub. Вы сможете делать все, что вы хотите с этим проектом, даже запросы на включение кода, если вы заинтересованы в участии в проекте 😊



Перейдите по ссылке <https://github.com/microsoftdx/vorlonjs> и нажмите кнопку **Fork** в верхнем правом углу окна:

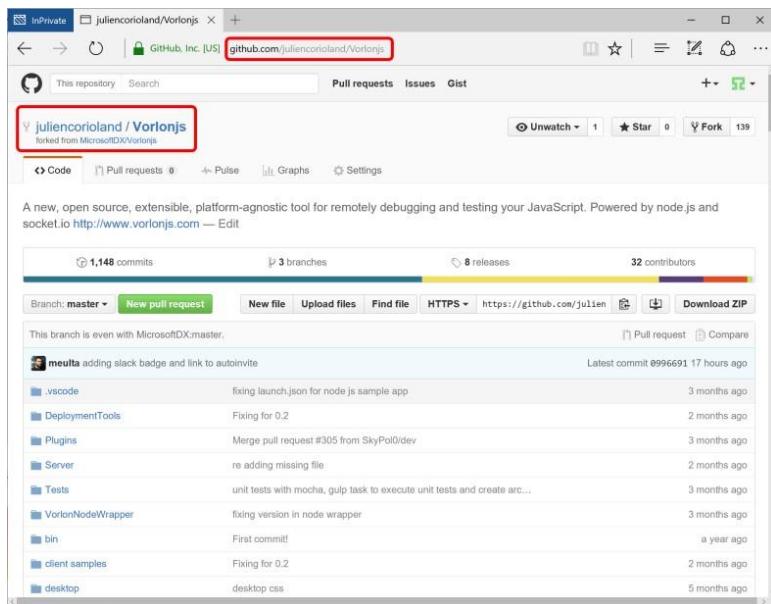


Затем подождите минуту, пока завершится создание форка:



После этого вы будете автоматически перенаправлены в репозиторий Vorlon.JS в вашем аккаунте GitHub:





Поздравляем! Вы создали форк проекта Vorlon.JS 😊

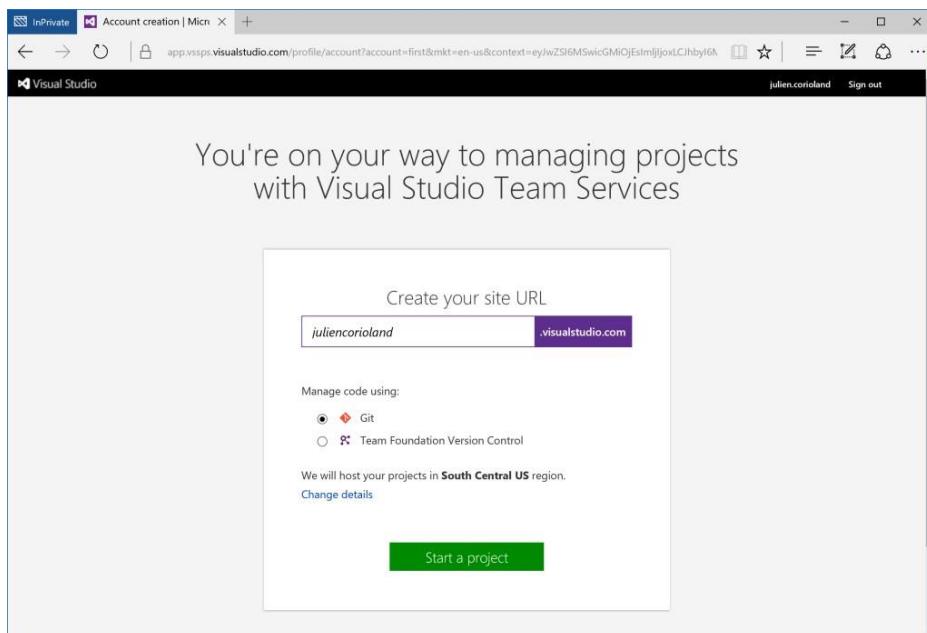
Создание учетной записи Visual Studio Team Services

В этой части вы будете создавать учетную запись Visual Studio Team Services (VSTS).

Если у вас уже есть аккаунт в VSTS, вы можете пропустить этот шаг.

Чтобы создать учетную запись VSTS, вам потребуется учетная запись Microsoft. Если у вас ее нет, перейдите по ссылке <http://signup.live.com> и создайте новую учетную запись.

Затем [войдите в Visual Studio Team Services](#) с помощью учетной записи Microsoft. Введите имя вашей учетной записи:

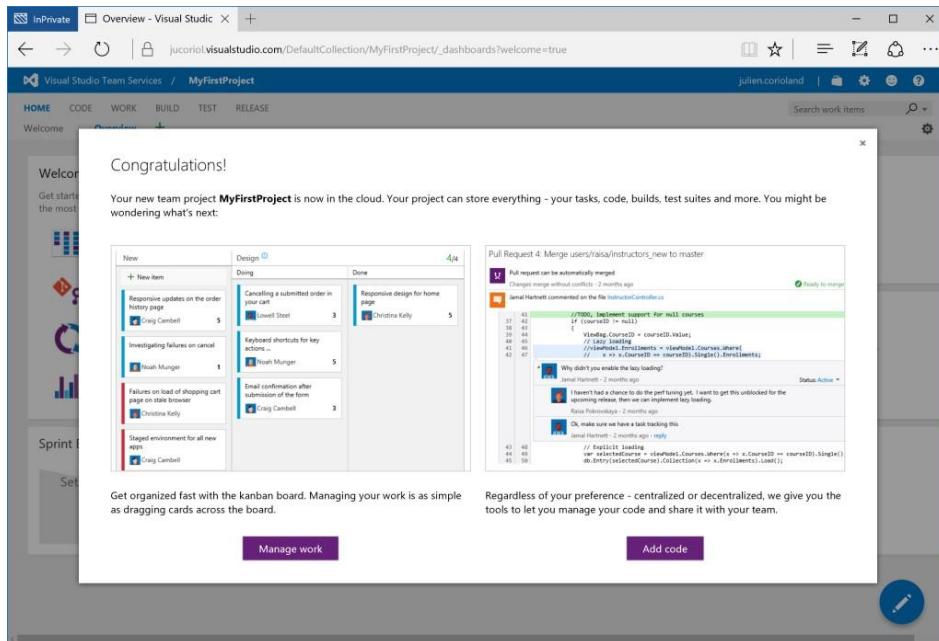


Вы можете выбрать Git, или Team Foundation Version Control, в зависимости от ваших предпочтений. В случае этой лабораторной работы, вы будете использовать GitHub.



Кликните **Start a project** и подождите минуту, пока будет создан ваш аккаунт.

После этого вы будете перенаправлены на следующую страницу:

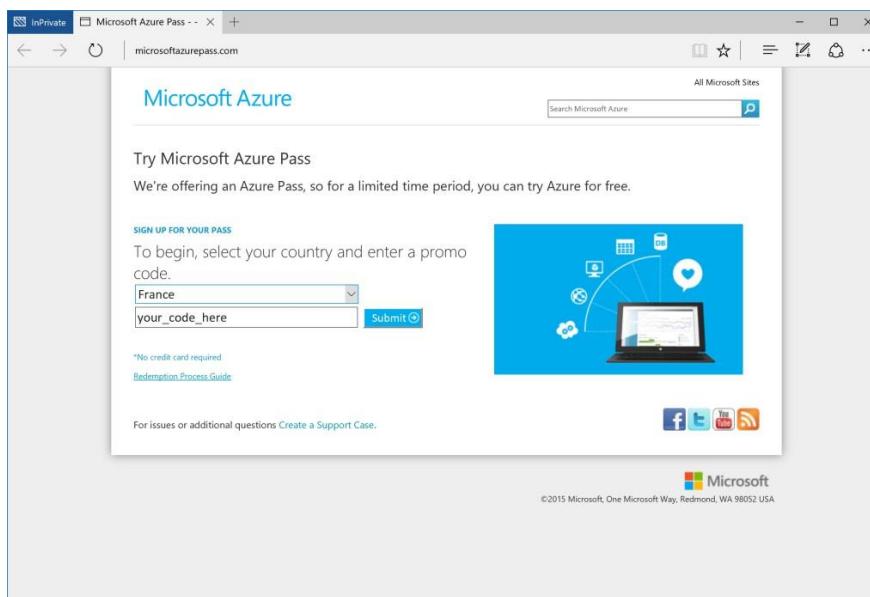


Сейчас вы можете просто закрыть всплывающее окно **Congratulations** и перейти к следующему шагу.

Активация Azure pass

Если у вас есть активная подписка Microsoft Azure, вы можете пропустить этот шаг.

Для этой лабораторной работы вам необходимо получить у организаторов код Azure Pass, который на некоторое время предоставит вам \$100 на использование сервисов в Azure. Чтобы активировать учетную запись Azure, перейдите по ссылке <http://www.microsoftazurerepass.com>, выберите вашу страну, введите код и нажмите кнопку **Submit**:



Вам будет предложено войти в систему при помощи учетной записи Microsoft. Вы можете использовать тот же аккаунт, который вы использовали для создания учетной записи VSTS на предыдущем шаге.



Подождите, пока будет создан ваш аккаунт Azure. После этого вы будете перенаправлены на портал управления Azure.

Теперь вы готовы к внедрению DevOps практик в проекте Vorlon.JS!

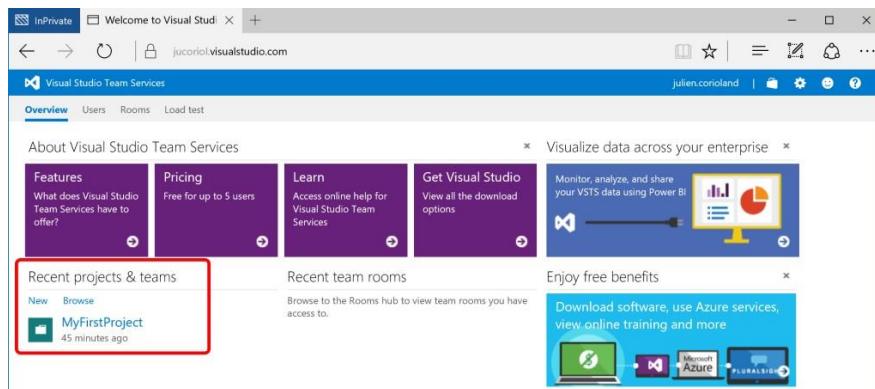
Обзор Visual Studio Team Services

В этой части представлен быстрый обзор возможностей Visual Studio Team Services. **Если вы знакомы с этим инструментом, вы можете перейти к следующей части.**

Перейдите к учетной записи VSTS, созданной ранее (используйте <http://youraccount.visualstudio.com>) и войдите в систему, используя данные учетной записи Microsoft.

После создания учетной записи также создается командный проект под названием **MyFirstProject**.

Нажмите на ссылку MyFirstProject на главной странице сайта, чтобы войти в командный проект:



Командный проект является логическим сущностью, которая будет группировать всю информацию по разработке проекта. Visual Studio Team Services - это мощный инструмент, который предоставляет систему управления версиями, контрольную панель, чтобы следить за проектом, управление рабочими элементами (work items) (user stories, task, bugs...) и панель задач, управление спринтами, инструменты сборки и непрерывной интеграции, тесты, управление релизами...

Для этой лабораторной работы вы можете остаться в командном проекте **MyFirstProject**.

Если у вас есть другие проекты разработки, и вы хотите использовать VSTS при работе с этими проектами, просто нажмите на ссылку **New** на домашней странице и создайте новый проект.

Вы можете получить доступ к полной документации по Visual Studio Team Services на этой странице: <https://www.visualstudio.com/get-started/overview-of-get-started-tasks-vs>

Непрерывная интеграция с Visual Studio Team Services

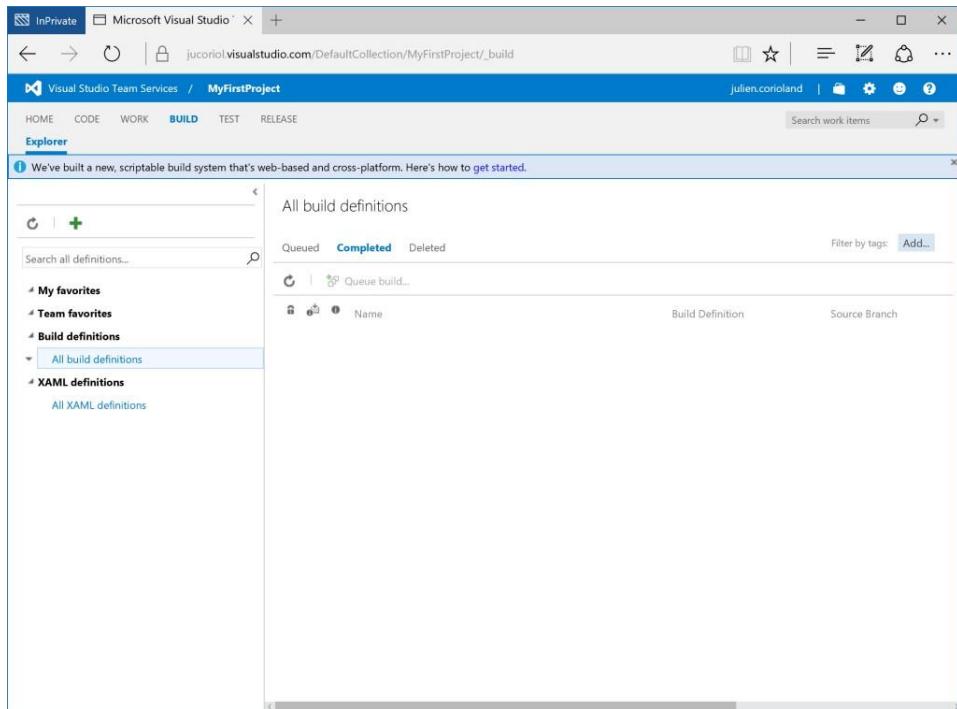
Введение

Непрерывная интеграция нужна для того, чтобы убедиться, что все изменения, которые сделаны разработчиками в исходном коде, не нарушают работу приложения, а также обнаружить некоторые регрессии/ошибки (баги) в проекте.



Для того чтобы сделать это, вы можете настроить один или несколько процессов сборки, которые будут срабатывать каждый раз, когда разработчик производит коммит в пакет\сборку приложения, затем выполнить модульное тестирование, анализа кода...

Перейдите на вкладку BUILD вашего командного проекта:



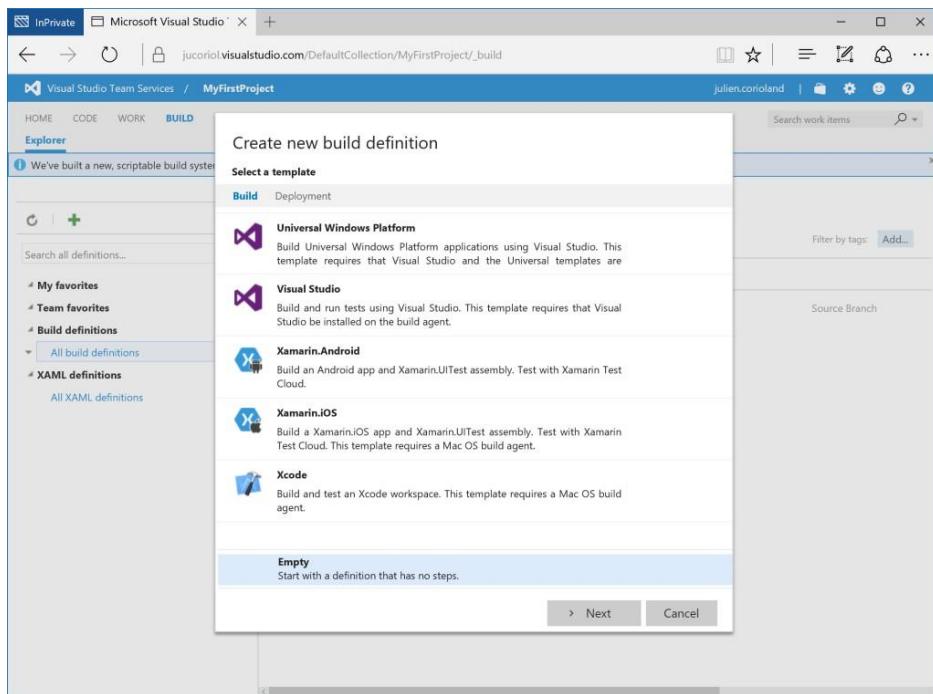
Это секция, где вы можете обрабатывать все определения сборки вашего проекта.

Создание нового определения сборки

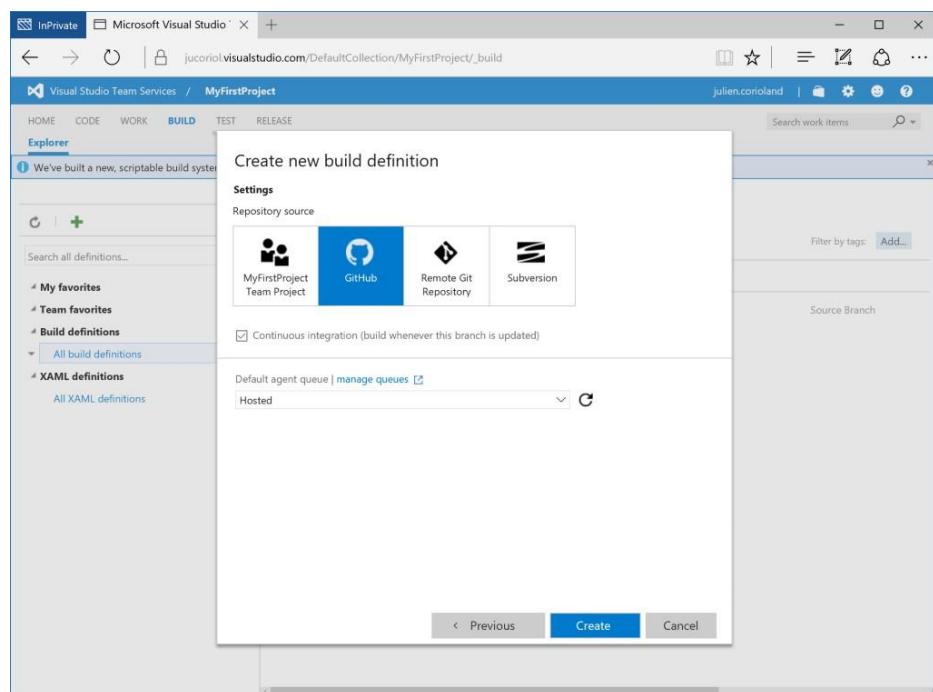
Чтобы добавить определение сборки, нажмите на кнопку в левой панели. Как вы видите, вы можете выбрать существующий шаблон, например, Universal Windows Platform или Visual Studio.

Выберите **Empty**, чтобы начать с пустого определения, и нажмите кнопку Next:





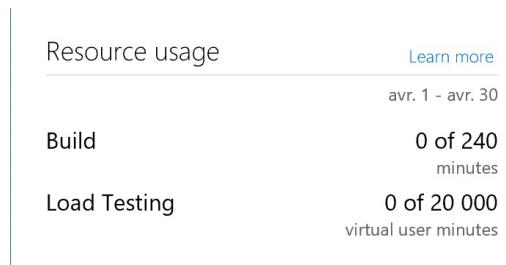
На следующем этапе вы должны выбрать репозиторий, который вы будете использовать, чтобы получить исходный код для сборки. Выберите GitHub и отметьте **Continuous integration** (производить сборку всякий раз, когда эта ветка обновляется), чтобы убедиться, что сборка будет запускаться каждый раз, когда осуществляется коммит в GitHub. Нажмите кнопку **Create**.



Для этой лабораторной работы, оставьте Hosted agent по умолчанию (как вы можете видеть на скриншоте выше).

Примечание: В VSTS сборка запускается на агенте сборки (в принципе, на любой машине, где установлен агент). Это может быть Windows, Linux или Mac OS, виртуальная или реальная машина. По умолчанию, VSTS предоставляет hosted Windows build agent, который вы можете использовать бесплатно в течение 240 минут в месяц (данные ниже доступны на домашней странице):



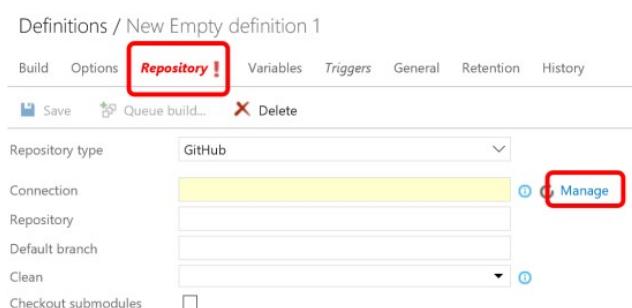


Если у вас сложный технологический процесс сборки, или вы должны использовать Linux, вы можете предоставить свой собственный агент. Это означает, что вы можете производить сборку самого разного ПО, используя Visual Studio Team Services. Если Microsoft не поддерживают вашу технологию по умолчанию, просто используйте свой собственный агент 😊 Чтобы получить более подробную информацию о Linux и Mac агентах, перейдите по ссылке: <https://github.com/Microsoft/vso-agent>.

Если вы знакомы с Chef, вы можете проверить представленный ниже репозиторий GitHub, который содержит руководство по начальной загрузке агентов для Windows и Linux VSTS с помощью Chef в Azure: <https://github.com/Microsoft/vsts-build-agent-cookbook/wiki>.

Подключение вашей учетной записи GitHub

Сейчас вы на этапе и в настройках для определения первого выпуска в VSTS. Первое, что нужно настроить, это подключение к GitHub. Перейдите на вкладку **Repository**. Убедитесь, что тип репозитория – GitHub, и нажмите на ссылку Manage, чтобы настроить соединение:



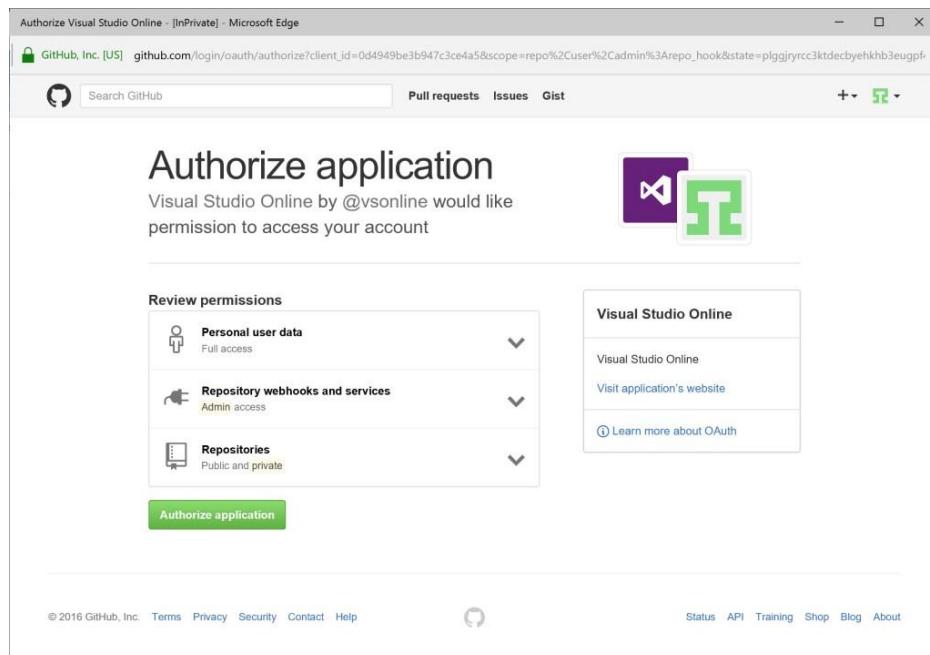
Откроются параметры Services Endpoint для проекта команды. К Visual Studio Team Services могут быть подключены большое количество внешних сервисов, таких как, например, Chef, Jenkins, Microsoft Azure или GitHub.

Нажмите на кнопку **New Service Endpoint** в левой панели и выберите GitHub. У вас есть два варианта для импорта вашего GitHub аккаунт:

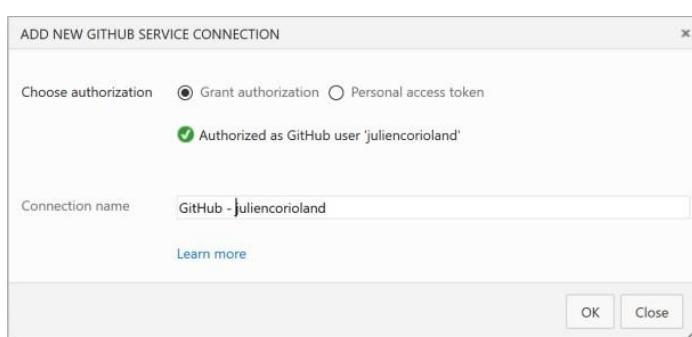
- Нажмите на кнопку авторизации, она перенаправит вас на GitHub, так что вы сможете авторизоваться в VSTS для работы с вашей учетной записью.
- Токен персонального доступа: укажите персональный токен доступа в ваших параметрах безопасности аккаунта GitHub, а затем предоставьте этот токен в VSTS.

Первый способ будет значительно проще. Нажмите на кнопку **Authorize** (возможно, потребуется изменить разрешения для всплывающих окон в вашем браузере), введите учетные данные аккаунта GitHub и авторизуйте приложение:

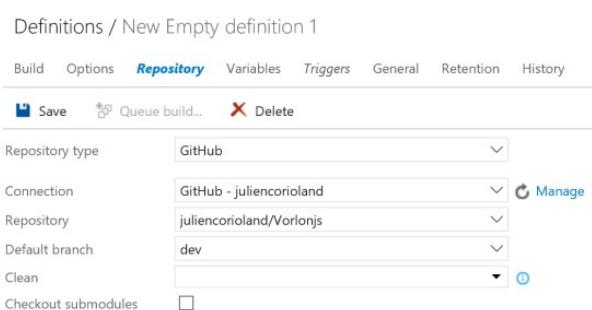




Затем кликните OK, чтобы установить связь в Visual Studio Team Services:



Вы можете вернуться в определении сборки и нажать на кнопку обновить, чтобы перезагрузить соединения GitHub. Соединение, которое вы только что настроили, должно появиться. Убедитесь, что выбран форк Vorlon.JS и выберите **dev** в качестве ветки Default Branch:



Настройка триггера сборки

Перейдите на вкладку Triggers и убедитесь, что была отмечена настройка для непрерывной интеграции:



The screenshot shows the 'Triggers' tab in the 'Definitions / New Empty definition 1' interface. At the top, there are tabs for Build, Options, Repository, Variables, Triggers (which is highlighted in blue), General, Retention, and History. Below the tabs are buttons for Save, Queue build..., and Delete. The 'Continuous integration (CI)' section is selected, with the sub-instruction 'Build each check-in.' displayed. Underneath, 'Batch changes' is checked. A 'Filters' section shows a dropdown set to 'Include' with the value 'dev' and a 'Add new filter' button. The 'Scheduled' section is also present but not selected.

Примечание: как вы можете видеть, также можно настроить сборку по расписанию. Например, вы можете запланировать сборку каждую ночь на вашем проекте, чтобы производить ночные сборки, которые будут использоваться пользователями-бета-тестерами.

Теперь вы можете перейти на вкладку Build, где вы будете определять различные шаги, из которых будет состоять процесс сборки.

Определение процесса сборки

Обзор

Как уже объяснялось во введении, Vorlon.JS разработан с использованием Node.js. Определение сборки будет использовать **npm** (node package manager), для того чтобы установить все зависимости, а затем выполнит две **Gulp** задачи:

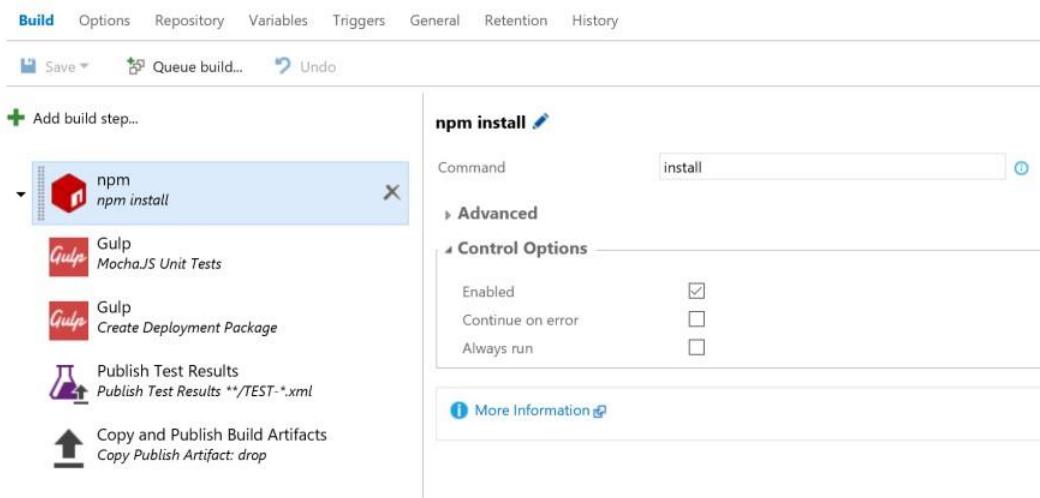
- Первая из них – для выполнения модульного тестирования, на основе фреймворка Mocha.js. gulpfile находится в каталоге Tests в проекте. Если вы увидите определение этого файла, то заметите, что задача выполняет тесты, а затем использует репортер для отображения результатов тестирования в виде XML-файла (используя формат JUnit), чтобы иметь возможность опубликовать результаты в VSTS.
- Вторая задача создает Zip-архив с приложением Vorlon.JS и всеми его зависимостями (этот архив будет использоваться для развертывания Vorlon.JS в Azure Web App в шаге управления релизами). Эта задача использует основной gulpfile приложения, расположенный в корневом каталоге.

После того, как задача Gulp выполнена, следующим шагом сборки выполниться задача **Publish Test Results**, которая будет принимать файл XML, полученный с помощью Mocha.js и импортировать результаты в Visual Studio Team Services, чтобы сделать их видимыми в протоколе сборки.

Последней задачей рабочего процесса является **Copy and Publish Build Artifacts**. Эта задача собирает содержимое каталога **DeploymentTools** (см на GitHub), а именно архив приложения, созданного во время выполнения второй Gulp задачи и Azure Resource Manager, PowerShell скрипт, Dockerfile и т.д., которые находятся в этом каталоге в системе управления кодом. Когда вы определите release pipeline с помощью Visual Studio Team Services Management Release (в следующих заданиях), вы будете повторно использовать эти артефакты, чтобы создать другую среду и развернуть приложение.

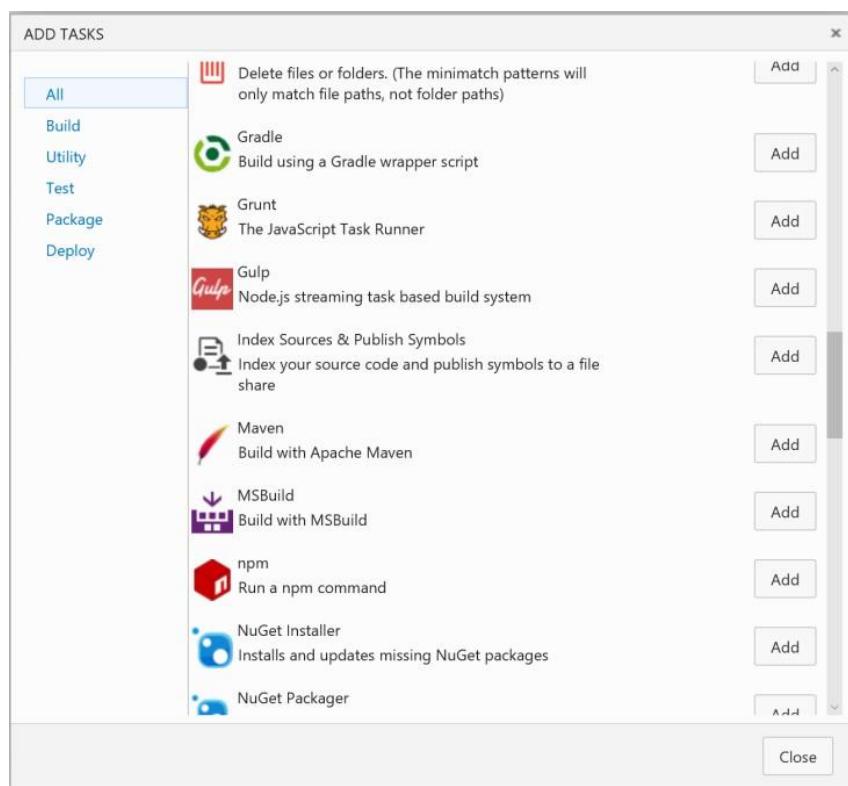
Окончательный процесс сборки будет выглядеть следующим образом:





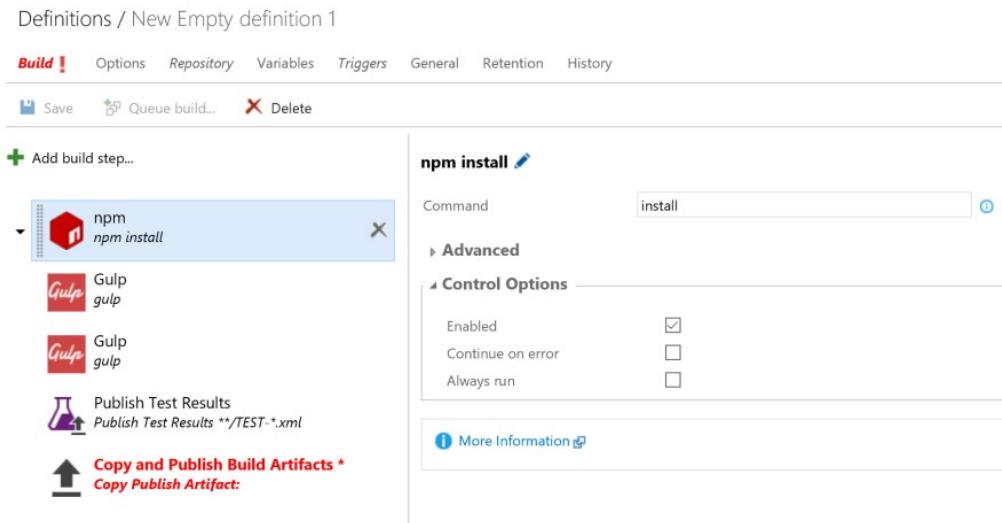
Добавление шагов в процесс сборки

Чтобы добавить новый шаг в рабочий процесс, нажмите на кнопку **Add build step...**. Вы увидите всплывающее окно со всеми возможными определениями задач, которые поддерживаются в Visual Studio Team Services (вы можете потратить минуту, чтобы просмотреть все эти задачи, а также посмотреть, как VSTS может интегрироваться с множеством других инструментов, таких как Chef, Android, XCode, Xamarin, Azure, Bash, Ant, Maven, Gulp, Grunt, npm...):



Нажмите на кнопку **Add** напротив **npm**, а затем два раза нажмите на кнопку **Add** напротив **Gulp**, после добавьте шаг **Publish Test Results** и **Copy and Publish Build Artifacts**. Теперь процесс сборки выглядит следующим образом:





Настройка шагов сборки

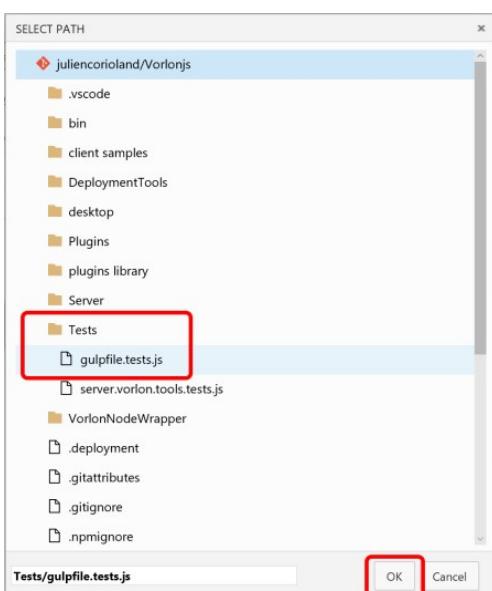
В этой части вы будете настраивать параметры каждой задачи, которая представляет из себя отдельный шаг и составляет определение сборки.

Npm задача

Вам ничего не нужно настраивать для выполнения этой задачи, она будет просто выполнять прм установки в корневой папке приложения.

Gulp задача # 1

Эта задача выполнит модульные тесты. Выберите задачу в списке в определении процесса сборки. В правой панели щелкните ... для просмотра репозитория и выберите файл **gulpfile.tests.js** в каталоге Tests. Нажмите кнопку **OK**, чтобы подтвердить:



Затем введите "tests" в поле задач(и) Gulp. Это имя задачи, определенной в Gulp файле. Вы также можете нажать на иконку около названия задачи, чтобы указать более подробное описание (например, " Gulp - Unit tests").



The screenshot shows the VSTS build pipeline editor. On the left, there's a tree view of build steps: 'npm install', 'Gulp - Unit tests' (selected), 'Gulp', 'Publish Test Results', and 'Copy and Publish Build Artifacts'. The 'Gulp - Unit tests' step has its configuration panel open on the right. It includes fields for 'Gulp File Path' (set to 'Tests/gulpfile.tests.js') and 'Gulp Task(s)' (set to 'tests'). Under 'Control Options', 'Enabled' is checked, while 'Continue on error' and 'Always run' are unchecked. A 'More Information' link is also present.

Gulp задача #2

Эта задача будет производить Zip-архив с приложением и всеми его зависимостями. Задача использует файл gulpfile.js по умолчанию, расположенный в корневом каталоге и выполняющий задачу под названием "zip":

The screenshot shows the VSTS build pipeline editor. The 'Create Deployment Archive' task is selected in the tree view on the left. Its configuration panel is open on the right, showing 'Gulp File Path' set to 'gulpfile.js' and 'Gulp Task(s)' set to 'zip'. Under 'Control Options', 'Enabled' is checked, while 'Continue on error' and 'Always run' are unchecked. A 'More Information' link is also present.

Задача Publish Tests Results

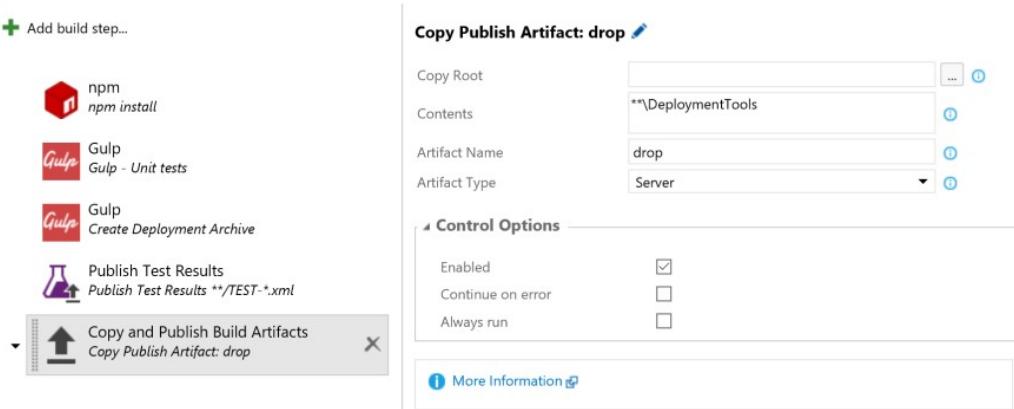
Эта задача будет обрабатывать файл XML, полученный с помощью задачи тестирования Gulp и будет импортировать результаты тестирования в отчет о сборке. Просто убедитесь, что **JUnit** выбран в качестве параметра Test Result Format:

The screenshot shows the VSTS build pipeline editor. The 'Publish Test Results' task is selected in the tree view on the left. Its configuration panel is open on the right, showing 'Test Result Format' set to 'JUnit', 'Test Results Files' set to '**/TEST-*.xml', and 'Merge Test Results' and 'Test Run Title' both unchecked. Under 'Control Options', 'Enabled' is checked, while 'Continue on error' and 'Always run' are unchecked. A 'More Information' link is also present.

Задача Copy and Publish Build Artifacts

Эта задача будет копировать артефакты, произведенные сборки, и публиковать их, чтобы они могли быть повторно использованы в следующих процессах, таких как управление релизами. В этом случае мы хотим скопировать папку DeploymentTools в папку **drop**, на сервере сборки (агента). Введите ****\DeploymentTools** в поле Contents, **drop** в поле Artifacts Name, и выберите **Server** в качестве Artifact Type.

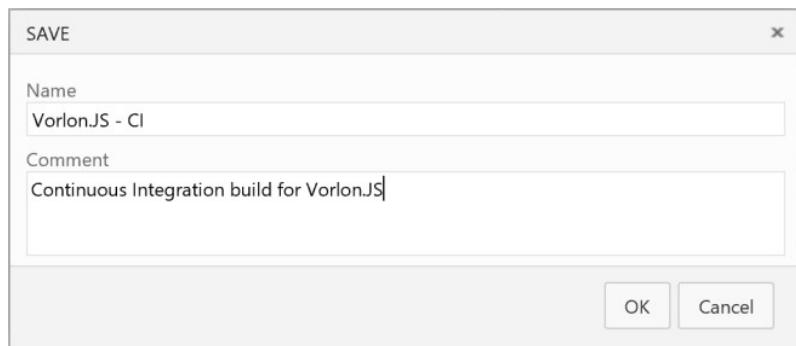




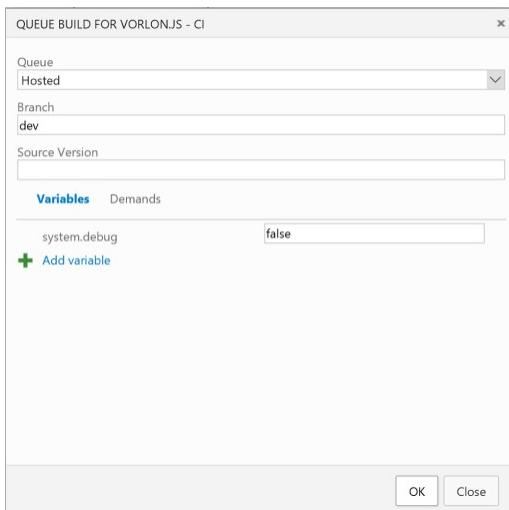
Сохранение определения сборки и запуск новой

Вы готовы к запуску процесса сборки! 😊

Нажмите на кнопку **Save** на панели инструментов и введите имя для этого определения сборки (например: Vorlon.JS - CI). Нажмите кнопку **OK**, чтобы сохранить:

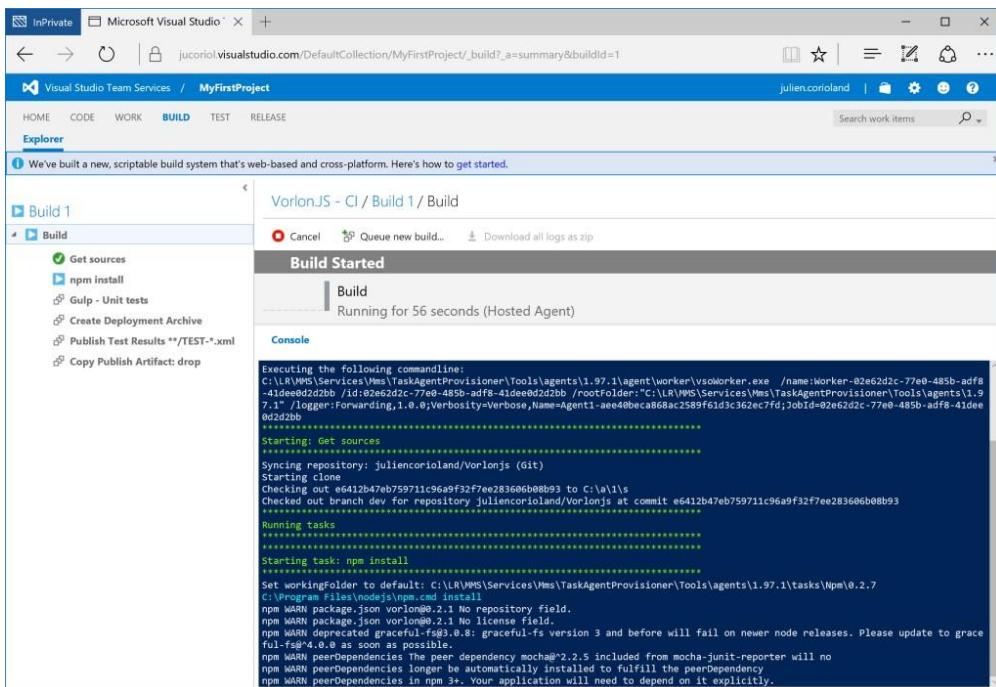


Теперь вы можете протестировать новую сборку, нажав на кнопку **Queue build...** на панели инструментов. Убедитесь, что выбран параметр **Hosted** для **Queue** и параметр **dev** для **Branch**. Затем нажмите **OK**:



Сборка начнется на доступном агенте, предоставляемом Visual Studio Team Services, и вы будете перенаправлены на консоль для сборки:

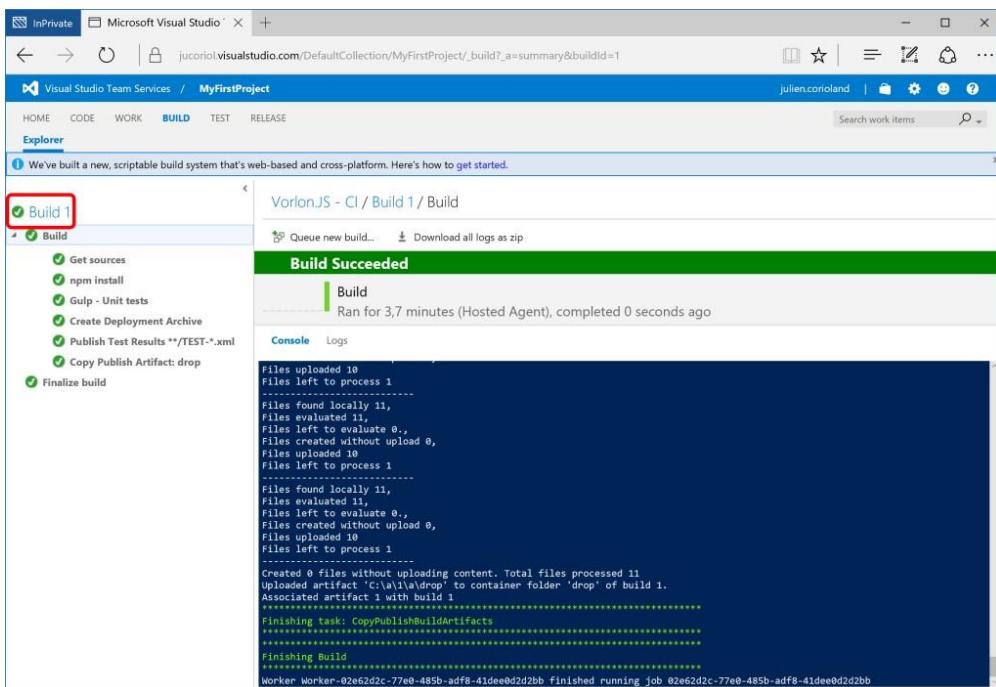




Подождите, пока сборка завершится.

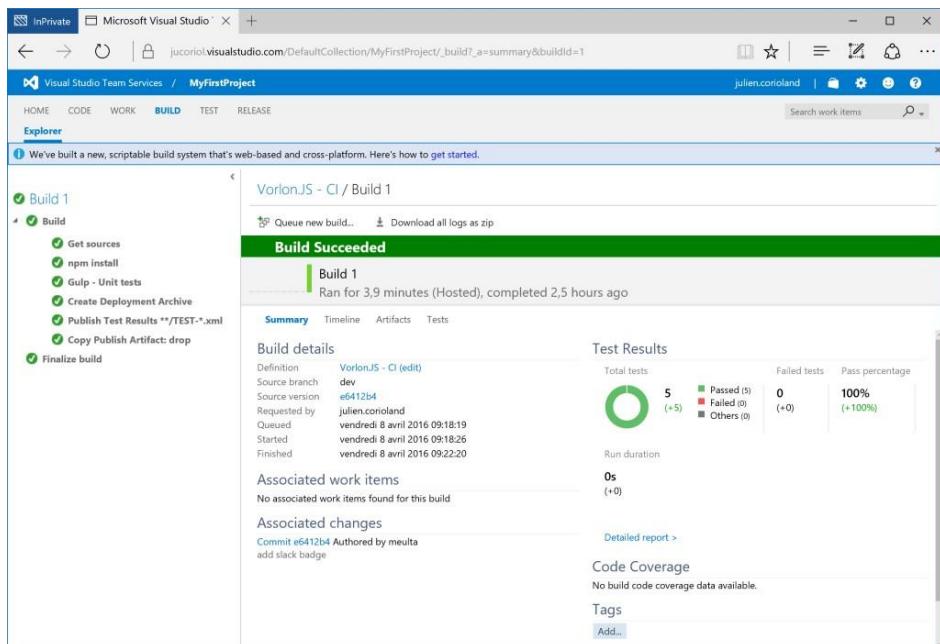
Просмотр отчета после сборки

После того, как сборка будет завершена, вы можете получить доступ к отчету о сборке, нажав на ссылку слева:



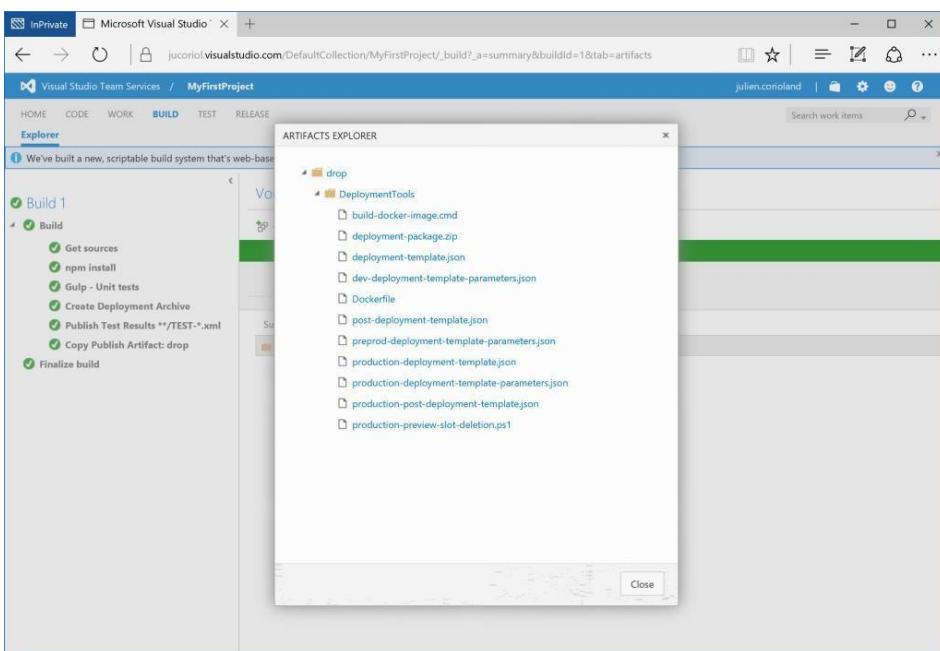
Затем вы получите все детали о процессе выполнения сборки:





Как вы можете видеть, вкладка **Summary** предоставляет вам важную информацию, такую, как общая информация о сборке и результатах тестирования (полностью интегрированы в приборную панель, благодаря задаче Publish Test Results).

На вкладке **Artifacts** вы можете просмотреть все содержимое, которое находится на сервере и будет повторно использоваться Release Management:



Вы также можете скачать архив с журналом всех логов, нажав на кнопку **Download all logs as zip**.

На данный момент вы реализовали процесс непрерывной интеграции для проекта Vorlon.JS. В следующей части вы узнаете, как использовать Visual Studio Team Services Release Management для управления средами с использованием подхода Infrastructure as Code и развертыванием приложения в Azure, автоматически и непрерывно.



Управление релизами с помощью Visual Studio Team Services

Введение

В этой части вы узнаете, как использовать функцию Release Management в Visual Studio Team Services, чтобы доставлять и развертывать Vorlon.JS в Azure, непрерывно.

Release Management представляет из себя процесс, который заключается в использовании артефактов, произведенных в процессе сборки, и использования их для развертывания приложения в соответствующей среде (например Dev, QA, Preproduction, Production). Release Management также помогает управлять рабочим процессом и переходами между этими средами. Представьте, что у вас есть три среды: **Development, Tests / QA, Production**.

Среда Development предназначена для команды разработчиков, и в ней приложение может быть развернуто каждый раз, когда сборка прошла успешно, так как эта среда не имеет решающего значения в процессе производства ПО. Следовательно ситуация, при которой там обнаружится ошибка, окажется не критичной для бизнеса.

Затем ответственный член команды может одобрить развертывание в следующей среде, **Tests**, где команда тестировщиков выполнит некоторые функциональные или нагрузочные тесты. Если они находят ошибки в приложении, они могут запретить выпуск этой версии приложения и она никогда не уйдет в **Production**. Но если все тесты будут успешными, команда тестирования сможет одобрить релиз и начнется развертывание приложения в Preproduction и т.д.

В этой практической лабораторной работе вы будете работать только с двумя средами:

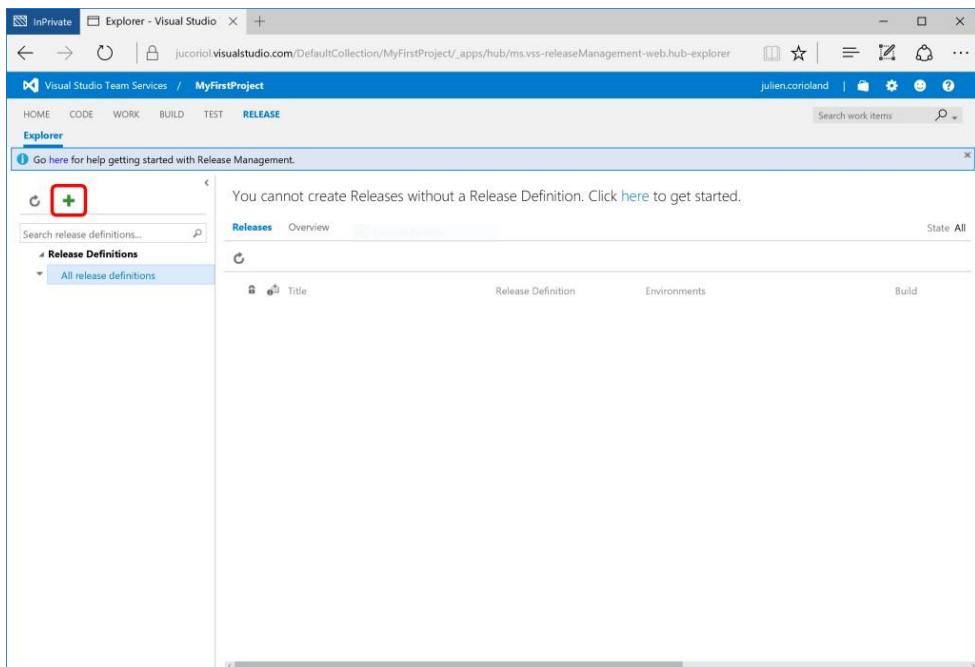
- Development
- Preproduction

Как только новая сборка завершится, будет запущена новый релиз, и приложение начнет развертываться в среду **Development**. Затем член команды (вы ☺) одобрит сборку, развернутую в **Development** среде, чтобы начать развертывание в **Preproduction**.

Создание нового определения релиза

Перейдите на вкладку RELEASE в Visual Studio Team Services. Перед созданием нового релиза вам необходимо создать определение релиза, которое в основном является определением сред и настройкой различных шагов для выполнения развертывания приложения. Нажмите на кнопку + в левой панели, чтобы создать новое определение релиза:





Как и для определения сборки, вы можете начать с существующего шаблона, но в данном случае следует выбрать пустой - Empty.

Определение сред

Во-первых, вы можете дать имя для определения релиза, например, "Vorlon.JS - Release". Затем, вы можете переименовать среду по умолчанию в Development и добавить вторую, названную Preproduction, нажав на кнопку + Add environment:

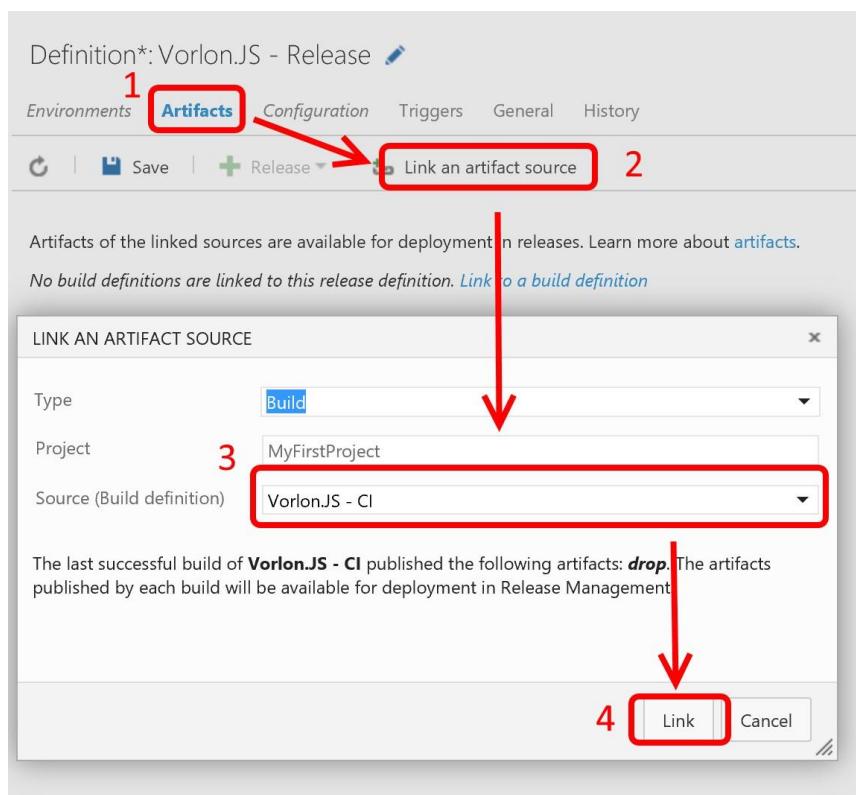
The screenshot shows the 'Add environments' dialog. At the top, it says 'Definition*: Vorlon.JS - Release'. Below that are tabs for Environments, Artifacts, Configuration, Triggers, General, and History. Under the Environments tab, there is a 'Save' button and a 'Release' dropdown. A message says 'No build definitions are linked to this release definition. Link to a build definition'. Below this, there is a 'Add environments' button with a red box around it. Two environments are listed: 'Development' and 'Preproduction', each with a red box around its name. To the right, there is a 'Add tasks' button.

Теперь вы должны связать определение сборки с этим определением релиза. Этот шаг является обязательным, потому что он указывает в Release Management, где будут находиться артефакты сборки, которые станут доступны во время релиза.

Ссылка на описание релиза к определению сборки

Перейдите на вкладку Artifacts и выберите определение сборки, которое вы создали во время предыдущего задания:



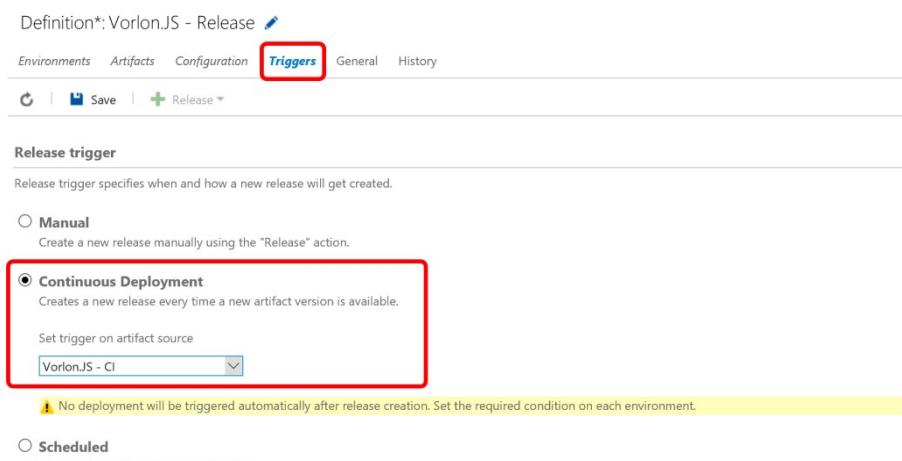


Нажмите на кнопку **Link**.

Настройка триггера

Процесс выпуска приложения сможет быть запущен, как только успешно завершится связанная сборка.

Чтобы сделать это, перейдите на вкладку **Triggers** и проверьте опцию **Continuous deployment**, выберите сборку, которую вы связали с определением релиза в качестве источника:



Не беспокойтесь по поводу предупреждения, вы исправите эту ошибку в конце этого задания.

Определение процесса релиза для каждой среды

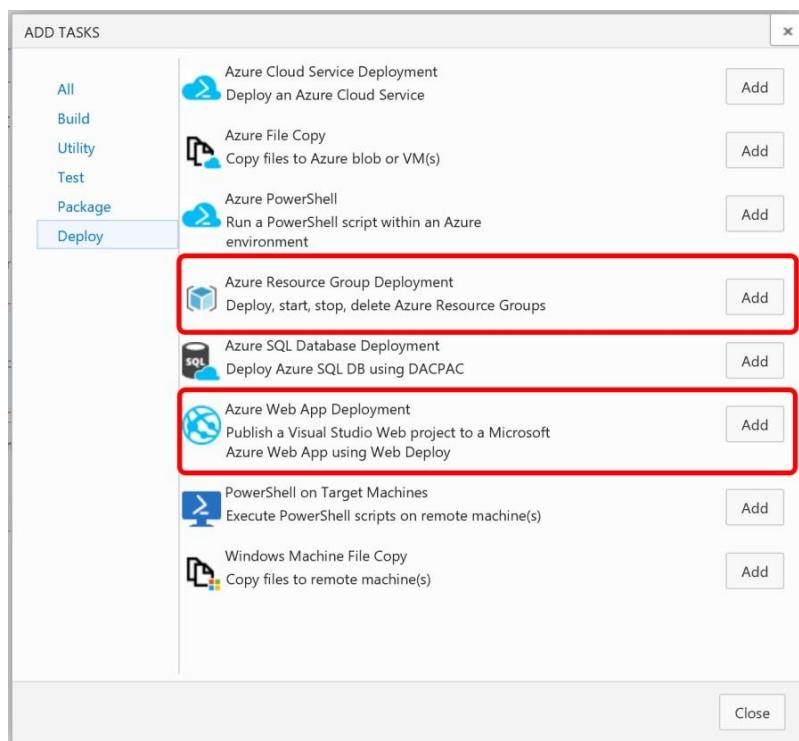
В данном случае процессы релиза для двух сред будут состоять из одинаковых шагов:



- Azure Resource Group Deployment: будет использовать шаблон Azure Resource Manager (ARM), чтобы создавать среду (которая состоит из простого Azure App Service WebApp)
- Azure Web App Deployment: эта задача несет ответственность за развертывание пакета приложения (созданного в процессе сборки) в Azure WebApp
- Azure Resource Group Deployment: будет применять другой шаблон ARM к существующей группе ресурсов, чтобы применить post-deployment конфигурацию в веб

Примечание: развертывание группы ресурсов с помощью шаблона ARM идемпотентно, так что вы можете применять тот же шаблон много раз, и результат будет таким же.

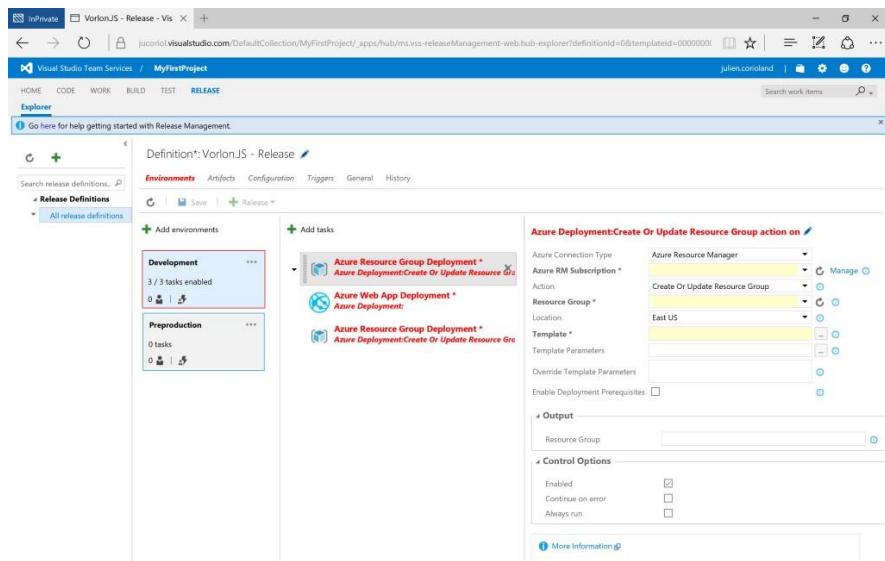
Выберите среду **Development** в левой панели и нажмите на кнопку **+ Add task**. Появится всплывающее окно, и вы сможете добавить три описанных выше задачи (обратите внимание, что интерфейс действительно выглядит точь-в точь, как определение сборки):



После добавления трех задач нажмите на кнопку **Close**.

Вы должны видеть похожую на скриншот картину в вашем определении релиза:





Подключение подписки Azure

Так как вы собираетесь развернуть приложение в Azure, вам нужно импортировать информацию об этом подключении в Visual Studio Team Services, как вы сделали это для GitHub в предыдущей части.

Выберите первую задачу в списке и нажмите на ссылку **Manage** напротив Azure RM Subscription, в открывшейся панели справа. Откроется страница для управления конечными точками проекта (endpoints).

Для этой лабораторной работы вам нужно добавить две конечных точки Azure, одну для **Classic** модели развертывания (старые Azure API, используемые задачей Azure Web App Deployment) и одну для модели **Azure Resource Manager** (новые мощные Azure API, используемые задачей Azure Resource Group Deployment).

Добавление Azure classic endpoint

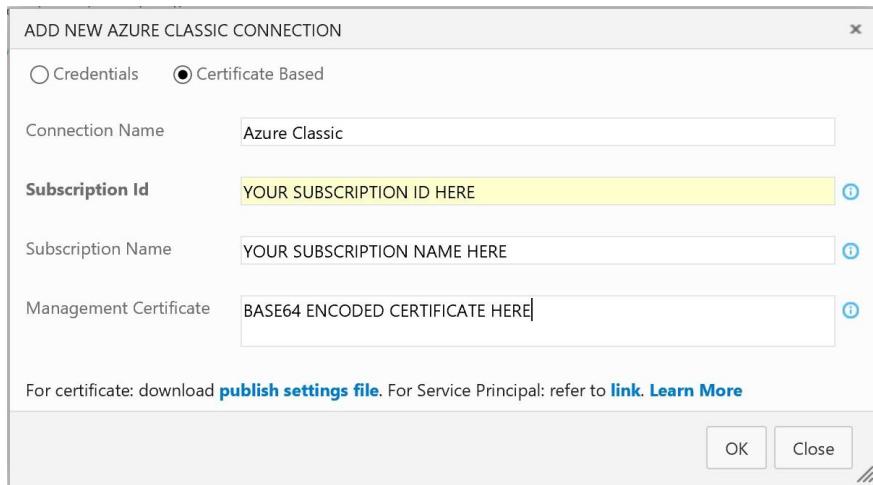
Нажмите кнопку **+ New service endpoint** в левой панели и выберите **Azure Classic**. В открывшемся окне нажмите на ссылку **publish settings file** для загрузки информации о подписке. Вам может быть предложено войти в свой аккаунт Azure. После загрузки откройте файл, например, с помощью Notepad.

Важная информация в этом файле: **Id** подписки и свойство **ManagementCertificate**:

```
<?xml version="1.0" encoding="utf-8"?>
<PublishData>
  <PublishProfile
    SchemaVersion="2.0"
    PublishMethod="AzureServiceManagementAPI">
    <Subscription
      ServiceManagementUri="https://management.core.windows.net"
      Id="b5[REDACTED]3">
      <ManagementCertificate>MIIKTAATB...[REDACTED]</ManagementCertificate>
    </Subscription>
  </PublishProfile>
</PublishData>
```

Вернитесь в управление VSTS и выберите вариант Certificate Based. Затем введите идентификатор подписки, имя и Management Certificate, затем нажмите кнопку OK, чтобы добавить конечную точку:





Добавление Azure Resource Manager endpoint

Чтобы подключить Azure Resource Management в Visual Studio Team Services, необходимо настроить Service Principal.

Для этого, загрузите [этот PowerShell скрипт](#) и выполните его на вашей машине.

Примечание: В случае возникновения ошибки, вам необходимо включить выполнение сценариев на вашем компьютере с помощью **Set-ExecutionPolicy cmdlet**.

Введите имя подписки и пароль, который вы хотите использовать. В открывшемся окне введите учетные данные Microsoft Azure. Подождите, пока генерируется service principle. После этого вся необходимая информация генерируется в окне вывода:

```
Copy and Paste below values for service connection
*****
Connection Name: Pass_Azure(SPN)
Subscription Id: c9edf49b-ab2c-43c3-aa72-be088368c22b
Subscription Name: Pass Azure
Service Principal Id: 893bb33a-1f0d-4ba0-929f-263a44be912c
Service Principal key: <Password that you typed in>
Tenant Id: b2bd0122-2afe-44df-a21b-6ab5e56b21e4
*****
```

Вернитесь в управление конечных точек служб в VSTS, нажмите кнопку **+ New service endpoint** в левой панели и выберите **Azure Resource Manager**. В окне введите информацию из PowerShell.

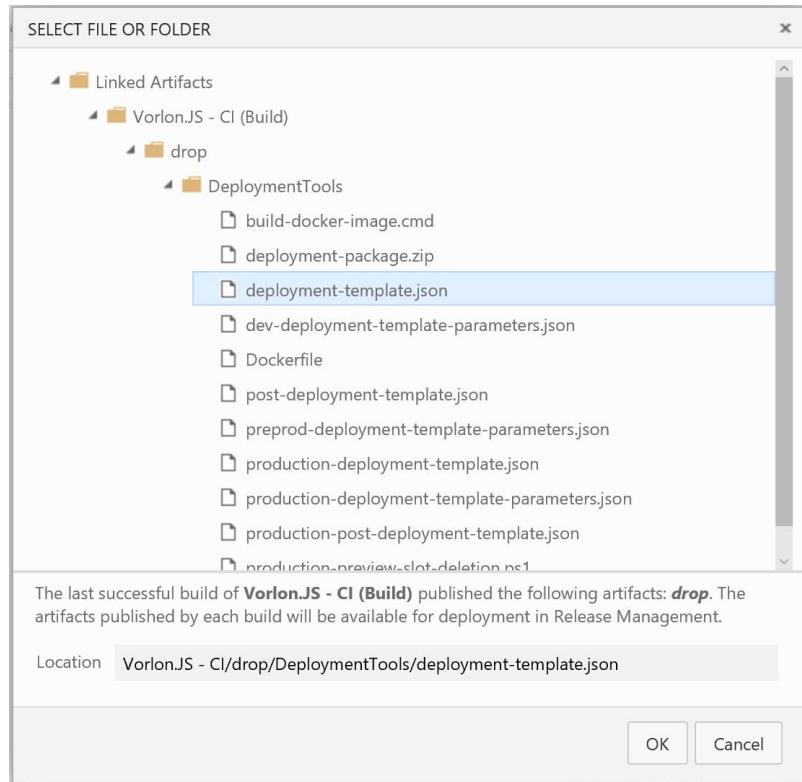
Нажмите на кнопку **OK**, чтобы добавить конечную точку. После добавления вы можете вернуться на определение релиза.

Настройка задач в среде Development

Обновите соединение Azure. Ваша конечная точка для Azure Resource Manager уже должна быть доступна. Дайте имя создаваемой группе ресурсов и выберите ЦОД, где требуется развернуть приложение:



Затем нажмите на кнопку ... напротив Template. Она откроет артефакты последней завершенной сборки. Просмотрите папку **DeploymentTools** и выберите файл **deployment-template.json**:



Сделайте то же самое для параметров шаблона и выберите файл **dev-deployment-templateparameters.json**.

Так как в Azure имя веб-приложения должно быть уникальным, вы должны переопределить имя по умолчанию в файле параметров. Чтобы сделать это, добавьте следующие строки в поле Override Template Parameters: `-siteName YOURNAME-vorlonjs-dev`

Конфигурация должна выглядеть следующим образом:

Create VorlonJS-Dev-RG

Azure Connection Type	Azure Resource Manager
Azure RM Subscription	Azure Resource Manager
Action	Create Or Update Resource Group
Resource Group	VorlonJS-Dev-RG
Location	North Europe
Template	/DeploymentTools/deployment-template.json
Template Parameters	ols/dev-deployment-template-parameters.json
Override Template Parameters	<code>-siteName jcorioland-vorlonjs-dev</code>
Enable Deployment Prerequisites	<input type="checkbox"/>



Теперь вы можете настроить задачу Azure Web App Deployment. На этот раз выберите подключение Azure Classic. Введите имя веб-приложения, которое вы указали на предыдущем шаге (в то время, как переопределяли параметр siteName) и выберите тот же ЦОД, что и раньше.

Для Web Deploy Package просмотрите артефакты сборки, нажав на ..., и выберите файл **deployment-package.zip**, расположенный в папке **DeploymentTools**:

Azure Deployment: jcorioland-vorlonjs-dev

Azure Subscription	Azure Classic	
Web App Name	jcorioland-vorlonjs-dev	
Web App Location	North Europe	
Slot		
Web Deploy Package	DeploymentTools/deployment-package.zip	
Set DoNotDelete flag	<input type="checkbox"/>	
Additional Arguments		

Для третьего шага повторите те же действия, что и для первого, но на этот раз используйте файл **postdeployment-template.json**:

Update web app configuration

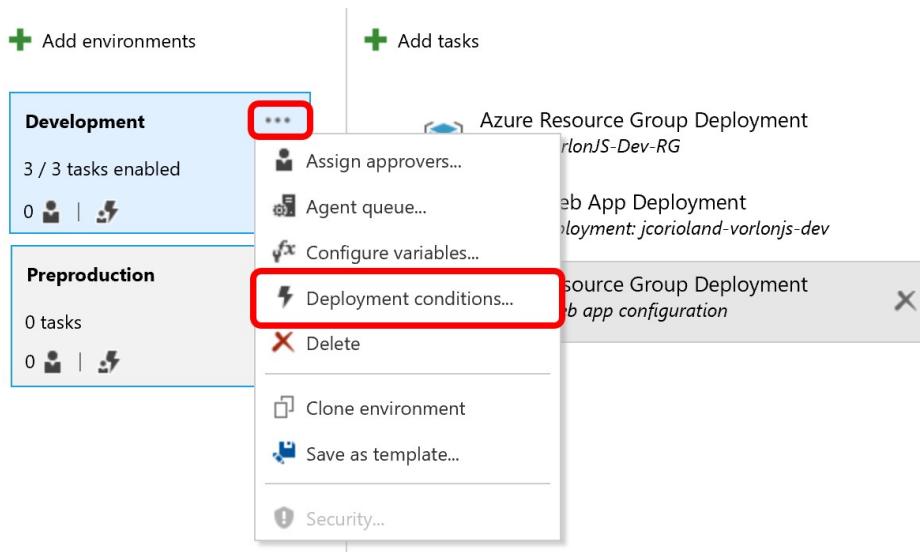
Azure Connection Type	Azure Resource Manager	
Azure RM Subscription	Azure Resource Manager	
Action	Create Or Update Resource Group	
Resource Group	VorlonJS-Dev-RG	
Location	North Europe	
Template	DeploymentTools/post-deployment-template.json	
Template Parameters	DeploymentTools/dev-deployment-template-parameters.json	
Override Template Parameters	-siteName jcorioland-vorlonjs-dev	
Enable Deployment Prerequisites	<input type="checkbox"/>	

Убедитесь в том, что вы используете ту же группу ресурсов (Resource Group), Location и переопределенные параметры, как и на первом этапе.

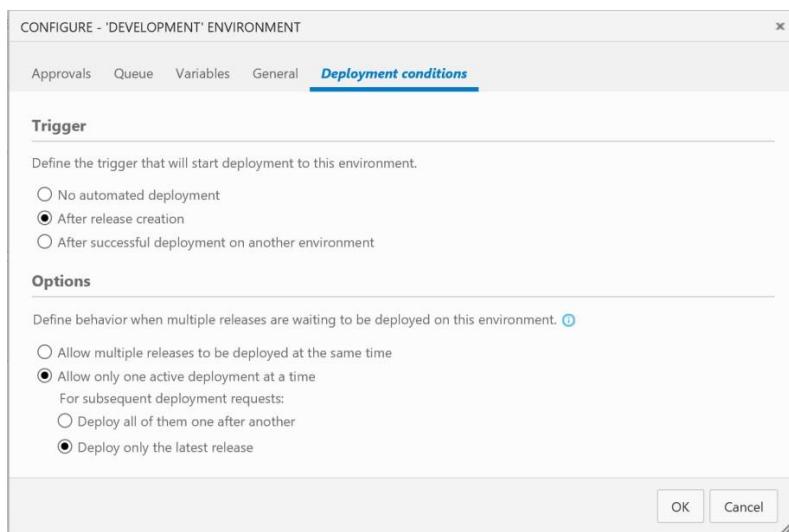
Создание условий развертывания для среды Development

Нажмите на кнопку ... на определении среды Development, чтобы открыть контекстное меню, и выберите Deployment conditions:





В разделе **Trigger** выберите **After release creation**, чтобы убедиться, что развертывание начнется, как только будет создан релиз (т.е. как только сборка будет успешно завершена). Нажмите кнопку **OK**, чтобы изменения вступили в силу:



Настройка задач в среде PreProduction

Среда Preproduction использует те же три задачи, что и среда Development:

- Azure Resource Group Deployment
- Azure Web App Deployment
- Azure Resource Group Deployment

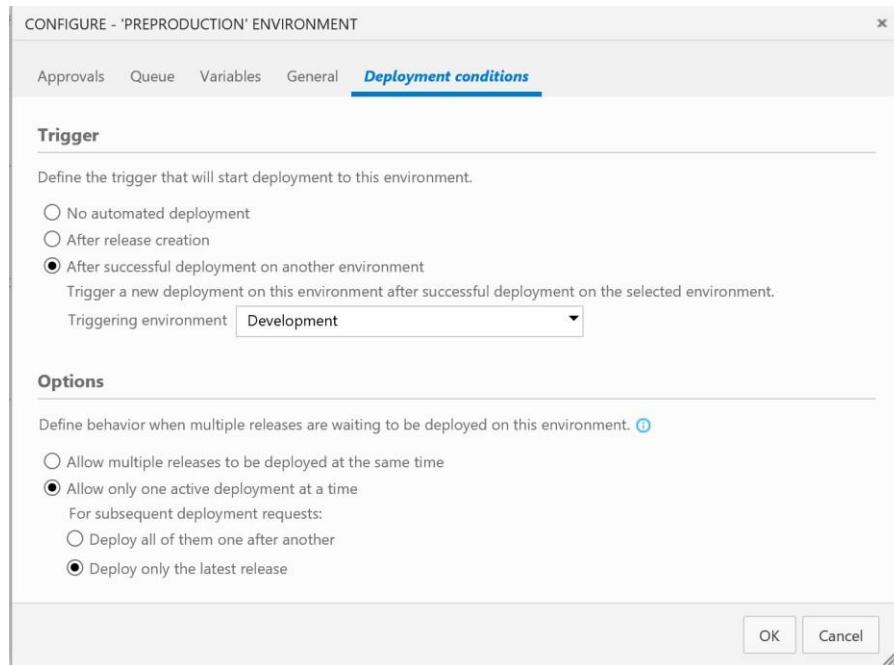
Повторите те же операции, что и ранее, но убедитесь, что вы используете параметры файла **preproddeployment-template-parameters.json** и суффикс вашего веб-приложения **preprod**, а не **dev**.

Вы так же можете клонировать среду **Development**, нажав на ..., для вызова контекстного меню и переименовать ее в Preproduction, затем изменить соответствующие параметры, как указано выше. Это в значительной степени позволит сэкономить вам время.

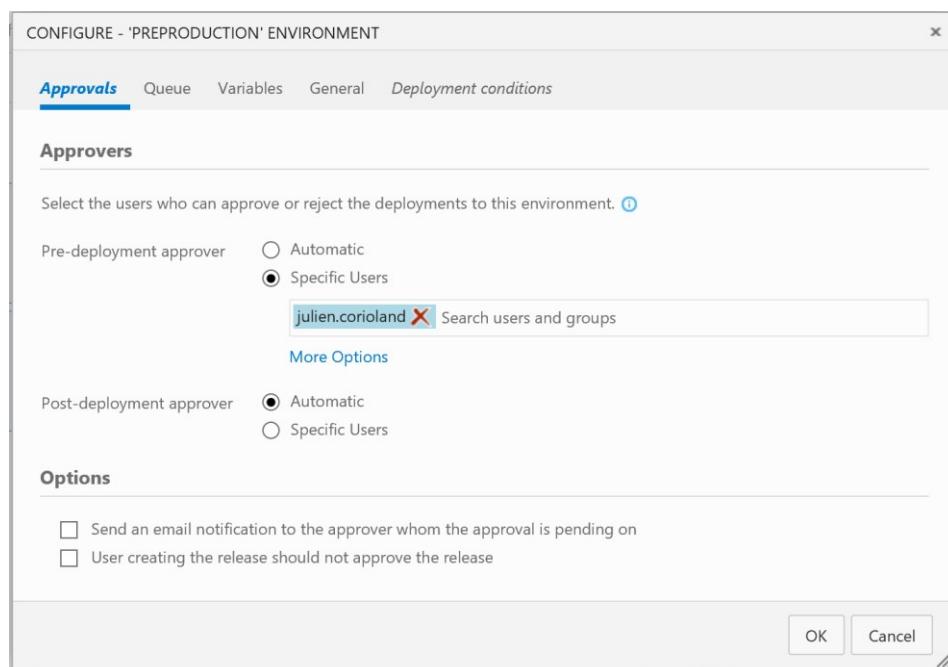


Создание условий развертывания для среды Preproduction

На элементе среды Preproduction нажмите на кнопку ..., чтобы открыть контекстное меню, и выберите Deployment conditions... В разделе **Trigger** убедитесь, что вы выбрали настройку After successful deployment on another environment, и выберите Development в поле Triggering environment:



Нажмите на вкладке **Approvals** и добавьте того пользователя, который сможет утверждать развертывание, которое не запустится в среде Preproduction без разрешения:



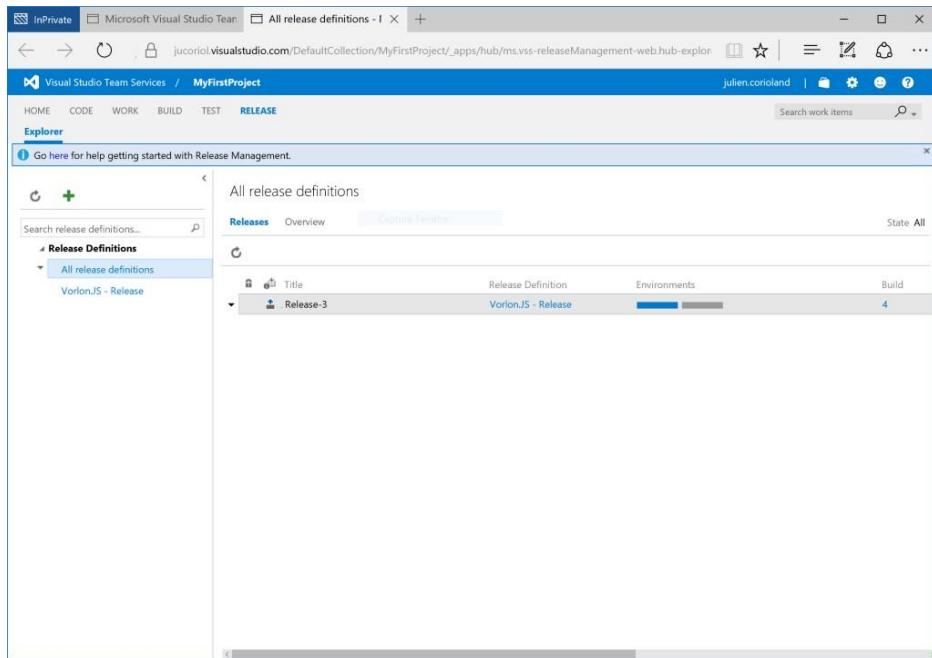
Нажмите на кнопку **OK**.



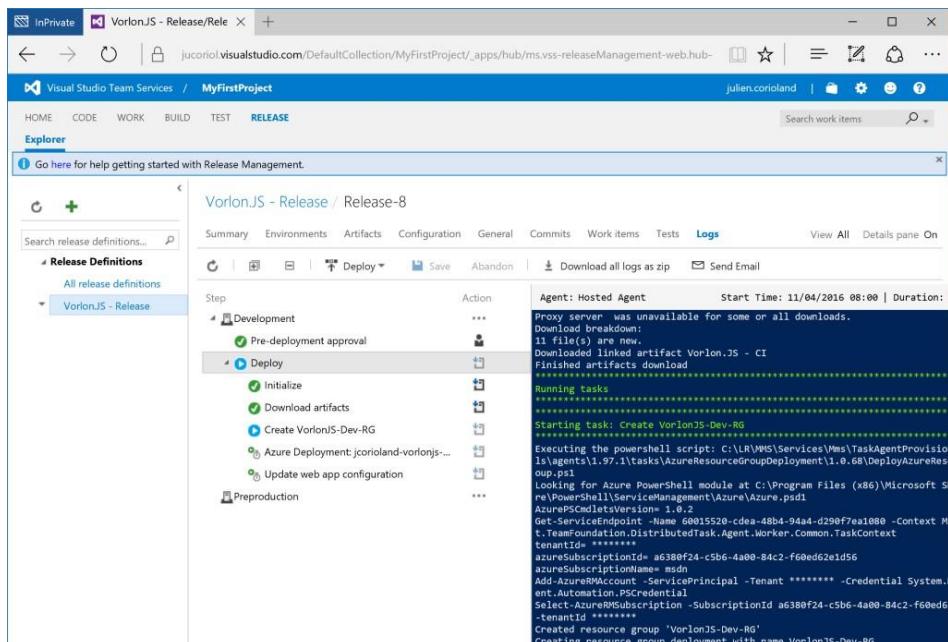
Сохранение определения релиза и запуск нового релиза

Теперь вы можете сохранить определение релиза, так как оно завершено! Затем вернитесь на вкладку BUILD и запустите новую сборку, как вы это делали раньше.

Когда сборка будет завершена, новый релиз будет создан автоматически и развертывание начнется в среде разработки:

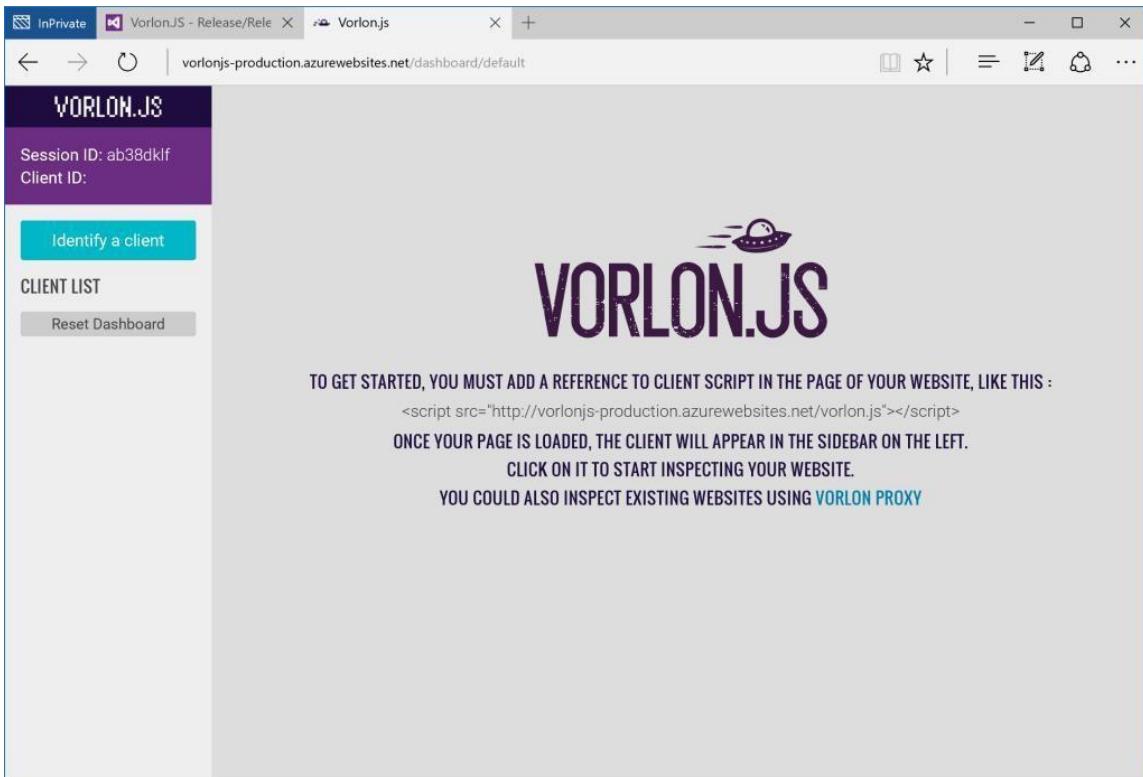


Если вы дважды щелкните на Release, вы можете получить пошаговые логи:

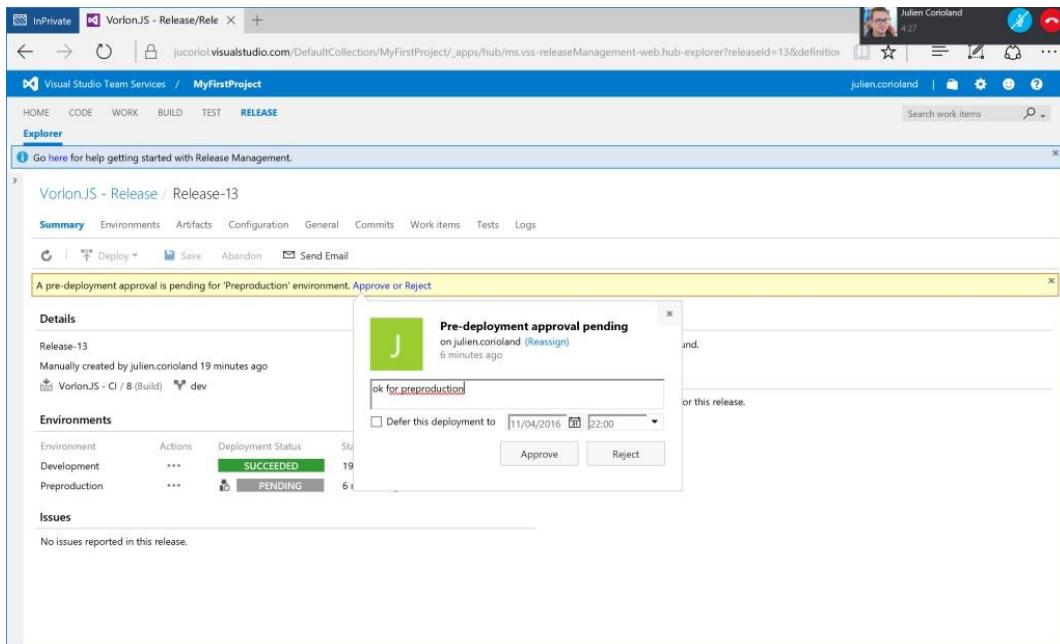


Как только релиз будет завершен в среде Development, вы сможете проверить веб-приложение, которое теперь развернуто на http://the_name_you_choose_for_your_site.azurewebsites.net:





Если вернуться в Visual Studio Team Services Release Management и выбрать текущий релиз, вы увидите сообщение, указывающее, что вам необходимо принять решение о продолжении развертывания в приложению уже в другую среду - **Preproduction**. Нажмите на ссылку, а затем на **Approve**.



И начинается развертывание приложения в **Preproduction**!



