HOL – Vorlon.JS: A journey to DevOps

Table of content

Content by Julien Corioland, Technical Evangelist, Microsoft France
Contact me on Twitter @jcorioland for any question.
Vorlon.JS DevOps blog post series: https://aka.ms/devopsvorlonjs

Introduction

In this hand on labs, you will learn how to use Visual Studio Team Services to implement DevOps practices like continuous integration, continuous deployment and release management.

You will work on a real project, Vorlon.JS which is a tool that allows to remotely debug web applications. If you already have used F12 tool in any browser to debug a web app, you can consider that Vorlon.JS does the same but remotely, so you can do it on any device, even mobile devices like Windows Phone, Android, iPhone or an Xbox One, for example.

Vorlon.JS is open source and its source code is hosted on GitHub: https://github.com/microsoftdx/vorlonjs. The team is using Visual Studio Team Services to do continuous integration, deployment and release management. You can read their story here.

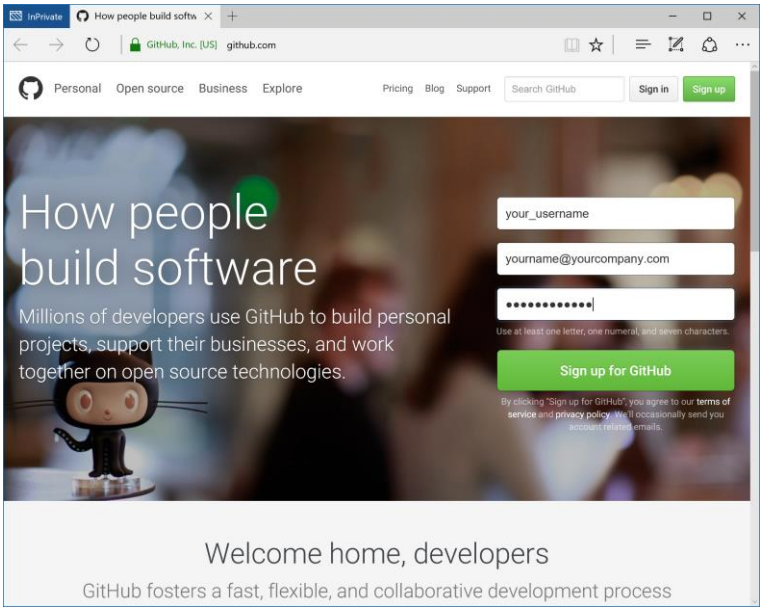
In this lab, you will create a fork of the Vorlon.JS project and then create continuous integration build and release management definition in Visual Studio Team Services, as the Vorlon.JS team do. As Vorlon.JS is developed in Node.JS, you will also learn how to use web / open technologies like, NPM, Gulp, Mocha.JS (and more…) with VSTS.

But first, let's do some check that you are ready to start!

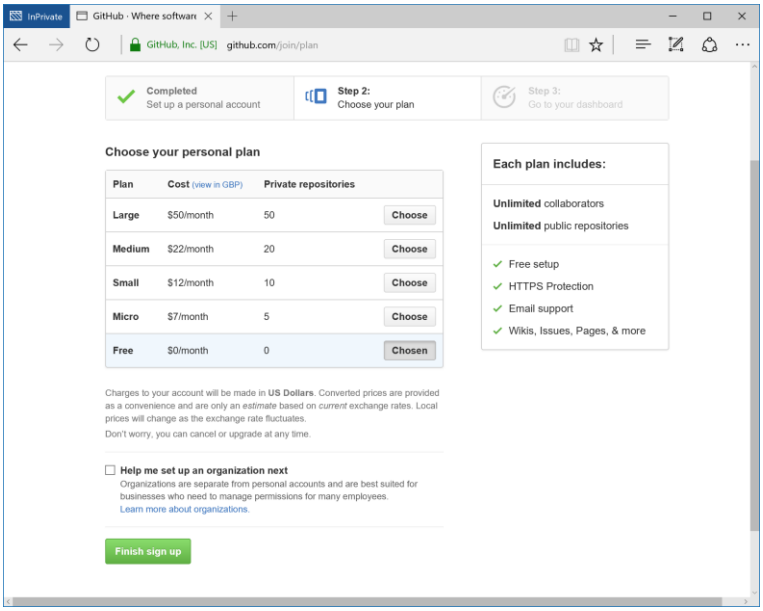
Prerequisites

GitHub account

For this lab, you will need a GitHub account (it's free!). If you already have one, you can skip this part. If you don't have a GitHub account, go to http://github.com and enter your username, your email and the password you want on the home page:



Then, click the **Sign up for GitHub** button. On the next screen, keep the Free option selected:

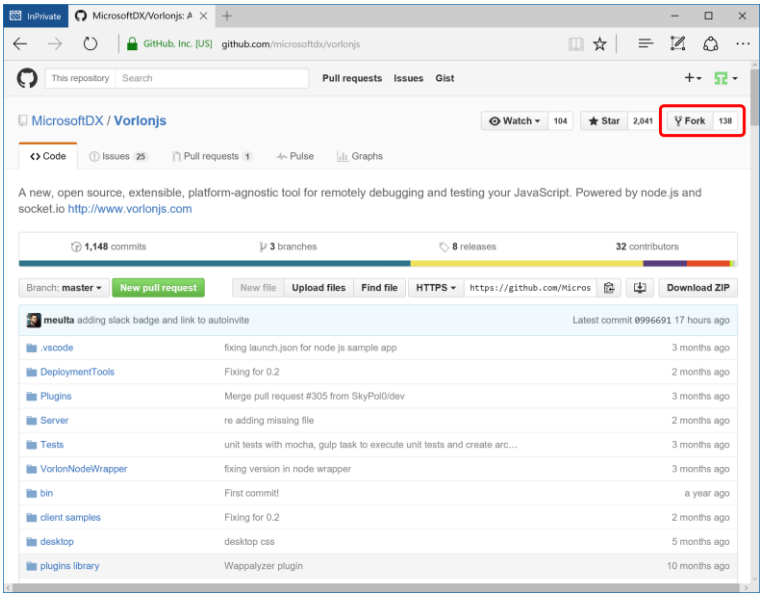


Click the button **Finish sign up**. A confirmation email has been sent to you, so you just have to go in your inbox and click the **Verify email address** button to confirm your account (you need to confirm it before going to the next part).

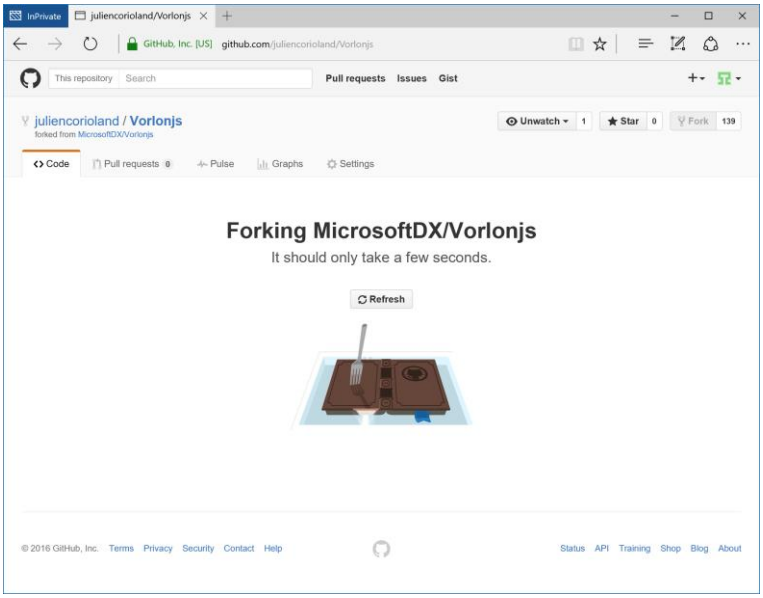
Create a Vorlon.JS fork

Now that you have a GitHub account you can create a fork of the Vorlon.JS project. Doing a fork will duplicate the project in your GitHub account. You will be able to do everything you want on the project, even doing pull requests if you are interested in contributing ☺

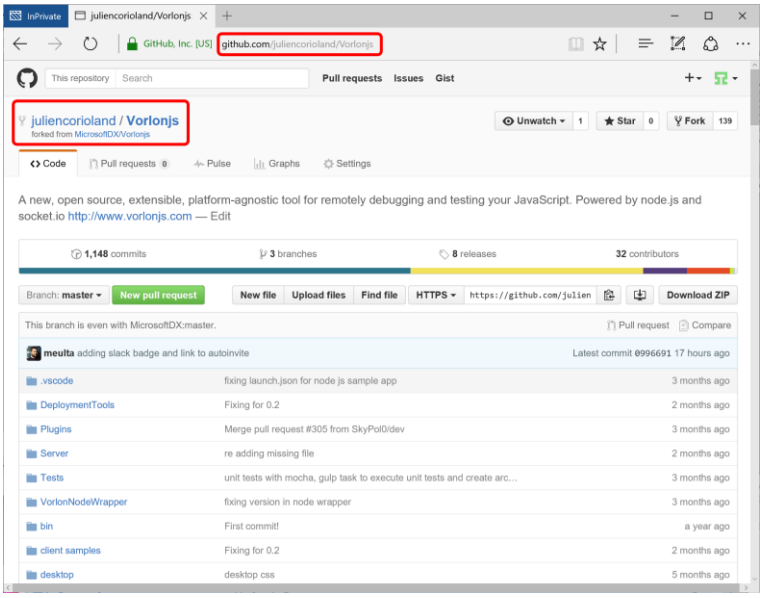
Go to https://github.com/microsoftdx/vorlonjs and click the **Fork** button at the top right of the window:



Then, wait a minute until the fork is completed:



Once done, you will be automatically redirected to your Vorlon.JS repository, on your GitHub account:



Congratulations, you have forked Vorlon.JS ☺

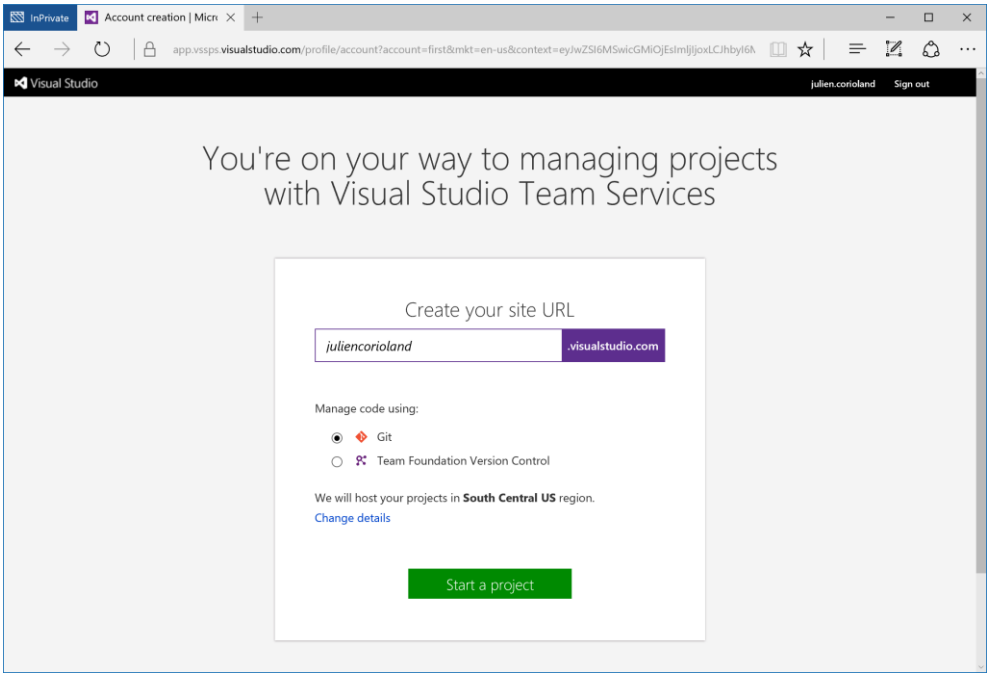
Create a Visual Studio Team Services account

In this part, you will create a Visual Studio Team Services (VSTS) account.

If you already have one, you can skip this part.

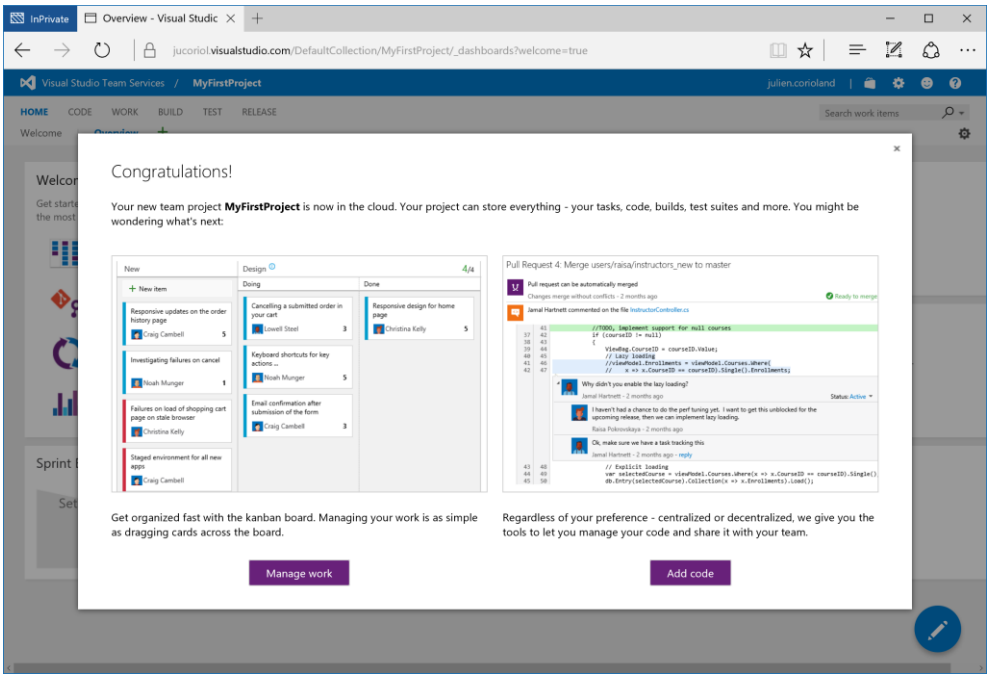
To create a VSTS account, you need a Microsoft Account. If you don't have one, go to http://signup.live.com and create a new one.

Then sign in to Visual Studio Team Services with your Microsoft account. Enter a name for your account:



You can choose either Git or Team Foundation Version Control, as you prefer. In the case of this lab, you will not use the VSTS source control but GitHub. Click on the **Start a project** button and wait a minute until your account is created.

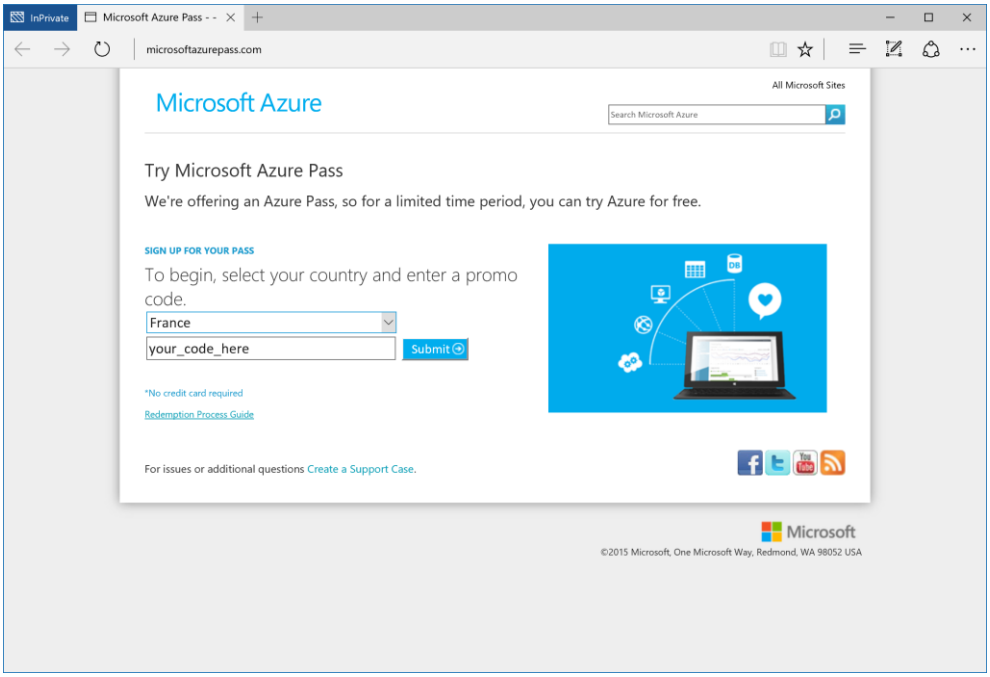
Once done, you will be redirected to this page:



For now, just close the **Congratulations** popup window and go to the next part.

Azure pass activation

For this lab you have received an Azure pass which provide $100 of Azure credits. To activate your Azure account, go to http://www.microsoftazurepass.com, select your country, enter your code and click the **Submit** button:



You will be asked to log in with a Microsoft Account. You can use the same one that you have used to create your VSTS account in the previous part.

Wait until your Azure account is created. Once done, you will be redirected on the Azure management portal.

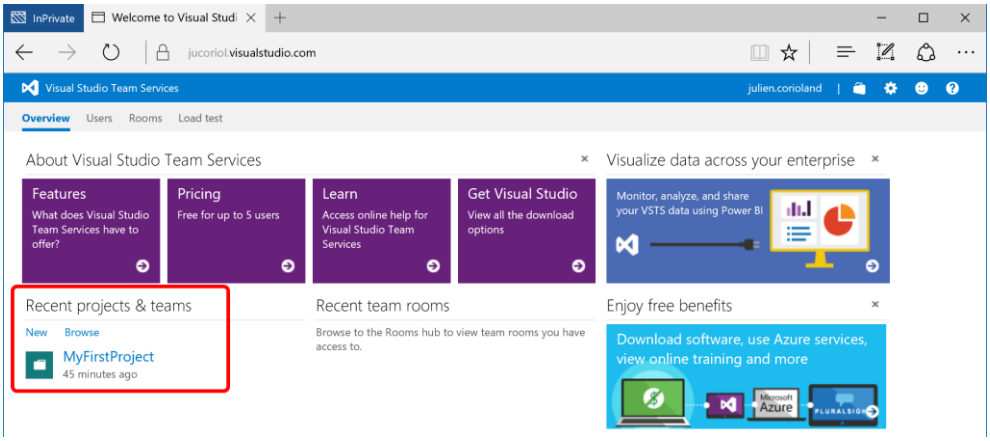
You are now ready to embrace DevOps on the Vorlon.JS project!

Visual Studio Team Services overview

In this part, you will have a quick overview of the Visual Studio Team Services features. If you are familiar with this tool, you can go directly to the next part.

Go to your VSTS account created before (use http://**youraccount**.visualstudio.com) and log in with your Microsoft account.

When you have created the account, a team project named **MyFirstProject** has been created too. Click on the MyFirstProject link on the home page to enter the team project:



The team project is a logical entity that will regroup all the information about a development project. Visual Studio Team Services is a powerful tool that provides source control management, dashboards

to follow project's evolution, work items management (user stories, task, bugs…) and task dashboard, sprint management, build and continuous integration, tests, release management…

For this lab, you can stay in the **MyFirstProject** team project.

If you have other development projects and want to use VSTS with these projects, just click the **New** link on the home page and create a new one.

You can get the full documentation of Visual Studio Team Services on this page: https://www.visualstudio.com/get-started/overview-of-get-started-tasks-vs

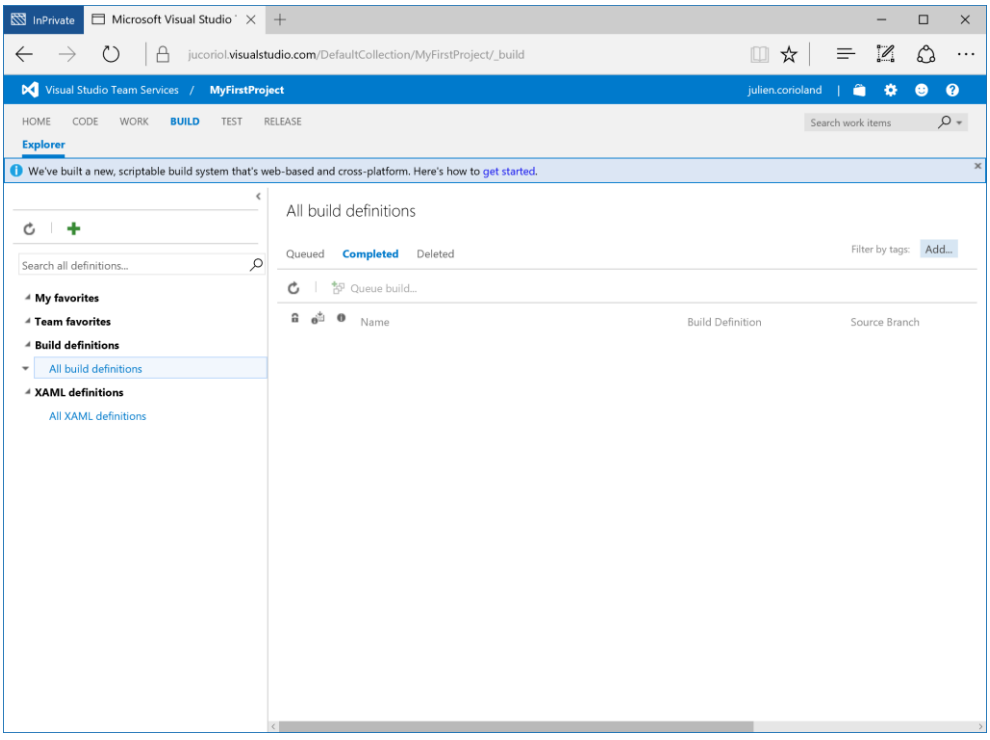
Continuous Integration with Visual Studio Team Services

Introduction

Continuous integration is about making sure that all the modifications that are done by the developers on the source code do not break the work of others or introduce some regressions/bugs in the project.

In order to do that, you can configure one or more build processes that will be triggered each time a developer commits code to package/build the application, execute unit tests, make some code analysis…

Go in the BUILD tab of your team project:

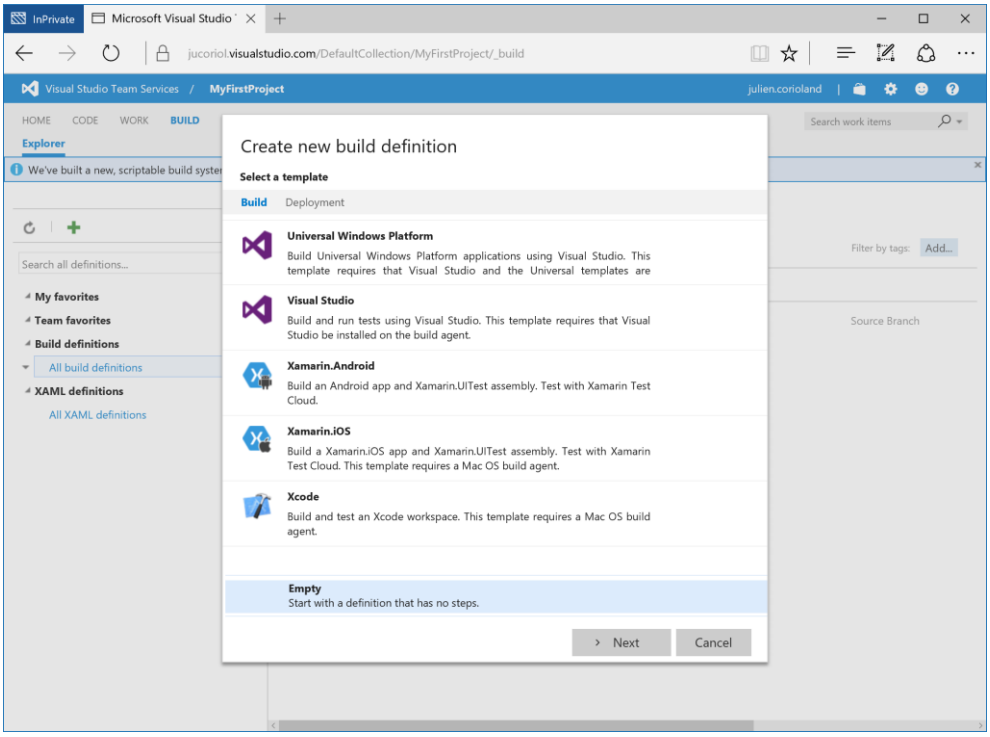


It is the section were you can handle all the build definitions of your project.

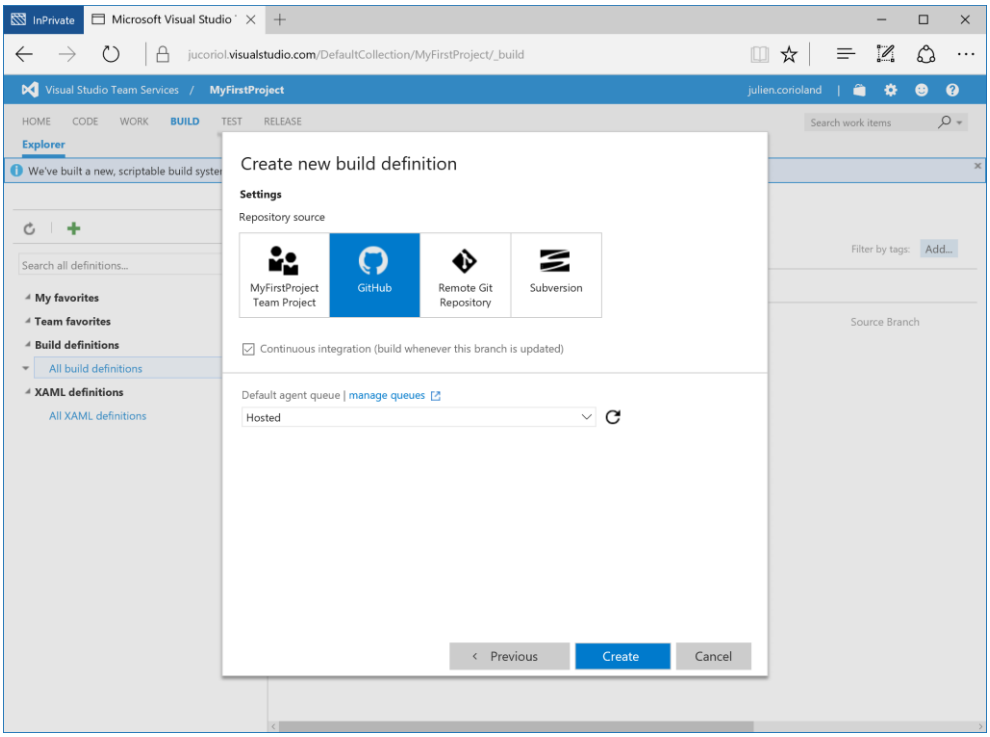
Create a new build definition

To add a build definition, click on the ✚ button in the left pane. As you can see, you can choose to start from existing templates like Universal Windows Platform or Visual Studio builds, for example.

Choose to start from the **Empty** definition and click Next:



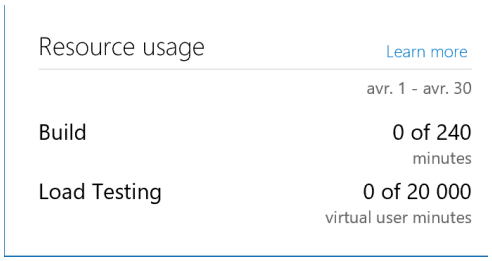
In the next step, you have to choose the source repository that you will use to get the source code to build. Choose GitHub and check the **Continuous integration (build whenever this branch is updated)** box to ensure that the build will be triggered each time a commit is done on GitHub. Click **Create**.



For this lab, keep the Hosted queue as default agent queue (selected by default, as you can see in the capture above).

Note: *With VSTS, builds are running on a build agent (basically any machine where an agent is installed). It can be a Windows, Linux or Mac OS machine, virtual or physical. By default, VSTS*

provides hosted Windows build agent that you can use for free for 240 minutes per month (see on your home page):

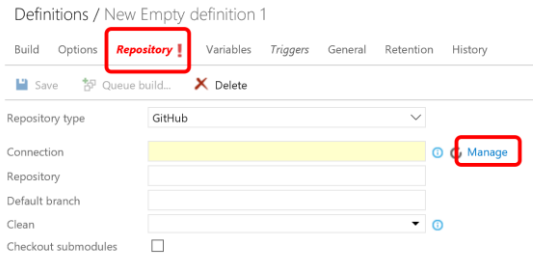


If you have complex build workflow or if you have to use Linux, you can provide your own agent. This means that you can build everything using Visual Studio Team Services. If Microsoft do not support your technology by default, just bring your own agent ☺ For more information about Linux and Mac agent, go to this page: https://github.com/Microsoft/vso-agent.

If you are familiar with Chef, you can check this GitHub repository that provides some cookbooks to bootstrap Windows & Linux VSTS agents using Chef in Azure: https://github.com/Microsoft/vsts-build-agent-cookbook/wiki.

Connect your GitHub account

You are now in the definition of your first Build in VSTS. The first thing to configure is the connection to GitHub. Go to the **Repository** tab. Check that the Repository type is GitHub and click on the Manage link to configure the connection:

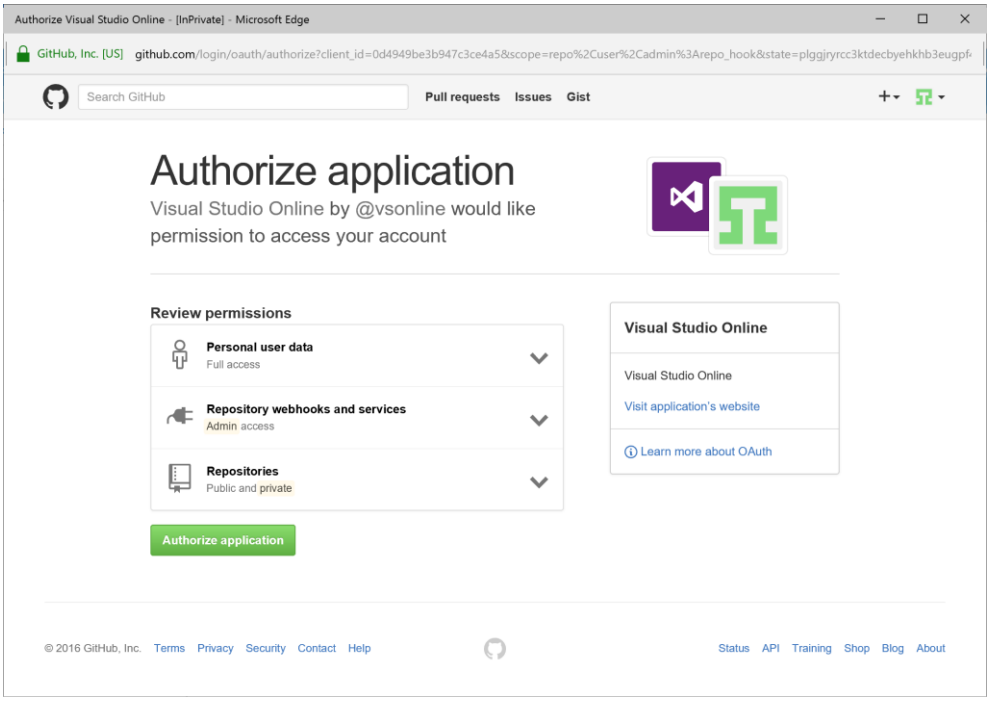


It will open the Services Endpoint parameters of the team project. Visual Studio Team Services can be connected to a lot of external services like Chef, Jenkins, Microsoft Azure or GitHub, for example.

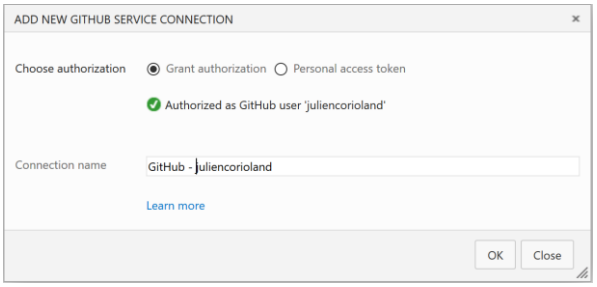
Click the **New Service Endpoint** button in the left pane, and choose GitHub. You have two options to import your GitHub account:

- Grant authorization: just click on the authorize button, it will redirect you to GitHub so you can authorize VSTS to work with your account
- Personal access token: declare a personal access token in your GitHub account security parameters and then provide this token to VSTS.

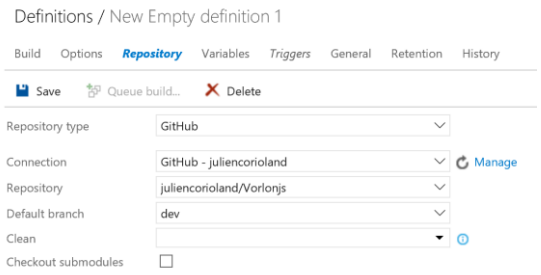
The simple way is to use grant authorization. Click on the **Authorize** button (you may need to authorize popup in your browser), enter your GitHub credentials and authorize the application:



Then click OK to validate the connection in Visual Studio Team Services:

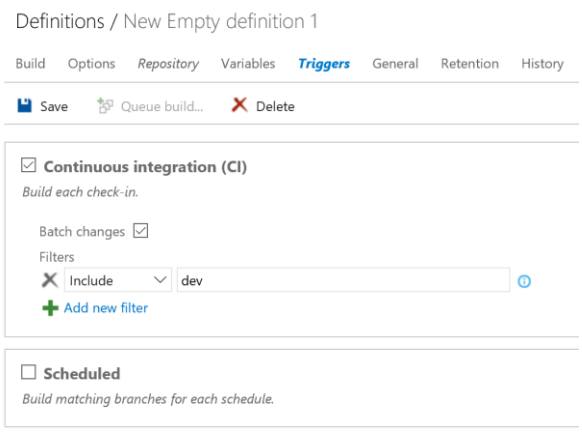


You can go back in the build definition and click the refresh button to reload the GitHub connections. The one you have just configured should appear. Ensure that the fork of Vorlon.JS is selected and pick **dev** for the Default Branch:



Configure build trigger

Go in the trigger tab and check that continuous integration has been selected:



Note: *as you can see, it is also possible to configure scheduled build. For example, you can schedule a build every night on your project to produce nightly builds that will be used by beta tester users.*

You can now go to the Build tab where you will define the different step that compose the build workflow.

Create the build workflow

Overview

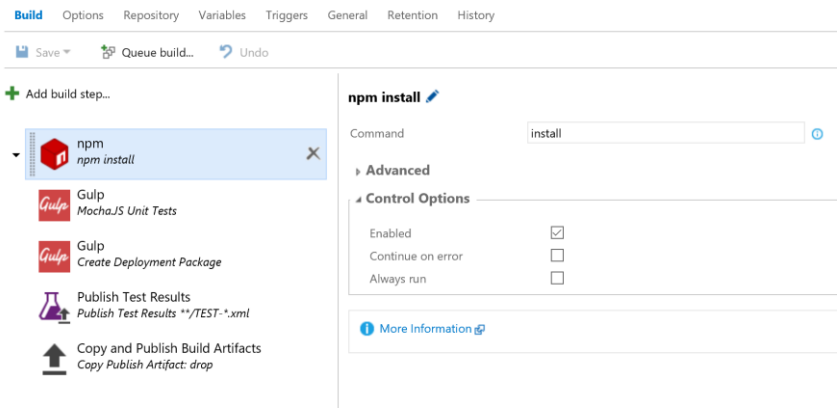
As explained in introduction, Vorlon.JS is developed using Node.JS. The build definition will use **npm** (node package manager) to install all the dependencies, then execute two **Gulp** tasks:

- The first one to execute unit tests, based on the Mocha.JS unit test Framework. The gulpfile is located in the Tests directory of the project. If you see the definition of this file, you will see that the task executes the tests and then use a reporter to produce a test results XML file (using JUnit format) to be able to publish the results in VSTS.
- The second one to create a zip archive with the Vorlon.JS application and all its dependencies (this archive will be used to deploy Vorlon.JS in an Azure Web App in the Release Management part). This task uses the main gulpfile of the application, located in the root directory.

Once the Gulp task are executed, the workflow will execute a **Publish Test Results** task that will take the XML file produced by Mocha.JS and import the results in Visual Studio Team Services to make them visible in the Build report.

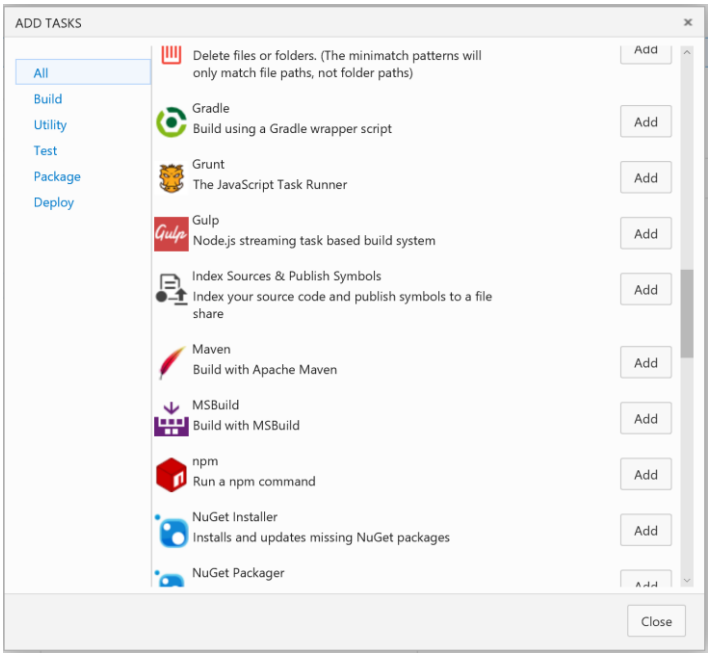
The last task of the workflow is a **Copy and Publish Build Artifacts**. This task will take the content of the **DeploymentTools** directory (see on GitHub) that contains the archive of the application that has been created by the second Gulp task and all Azure Resource Manager, PowerShell script, Dockerfile etc. that are in this directory in the source control. When you will define a release pipeline using Visual Studio Team Services Release Management (in the next part) you will reused these artefacts to create the different environment and deploy the application.

The final workflow will look like the following:

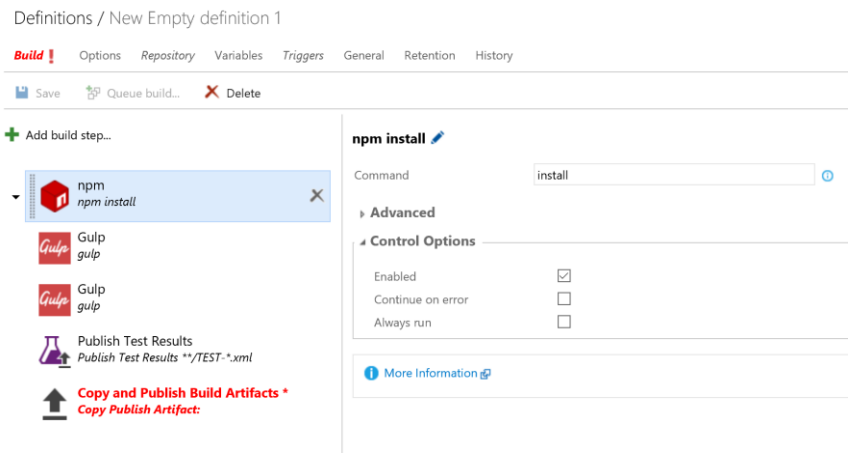


Add the steps to the workflow

To add a new step in the workflow, click on the **Add build step…** button. A popup with all task definitions that are supported by Visual Studio Team Services (you can take a minute to browse all these tasks and see how VSTS can integrate with a lot of other tools like Chef, Android, XCode, Xamarin, Azure, Bash, Ant, Maven, Gulp, Grunt, npm…):



Click on the **Add** button on the **npm** line, then click two times on the Add button on the **Gulp** line, then add a **Publish Test Results** step and a **Copy and Publish Build Artifacts**. Your build workflow now looks like the following:



Build tasks configuration

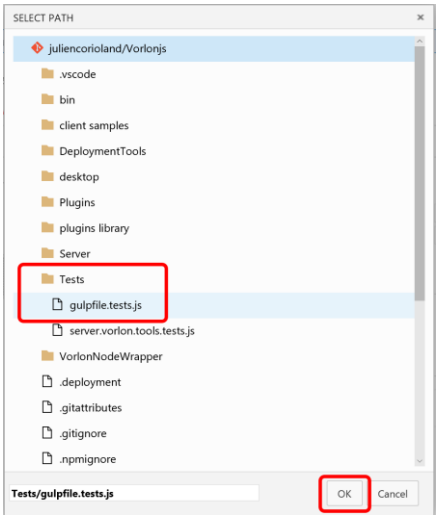
In this part, you will configure the parameters of each task that compose the build definition.

Npm task

You have nothing to configure for this task, it will just execute an npm install in the root folder of the application.

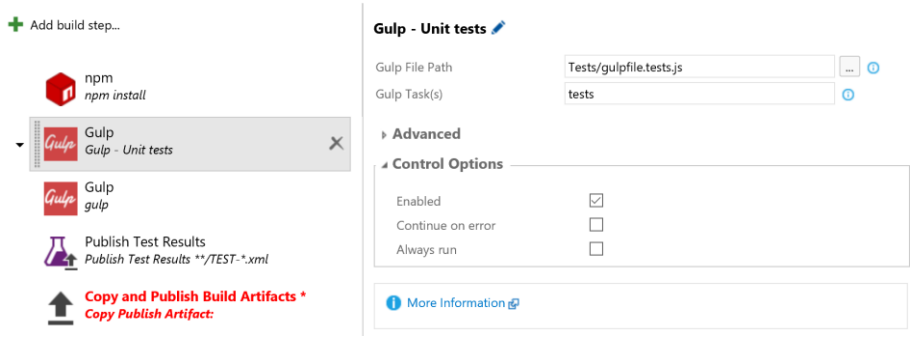
Gulp task #1

This task will execute unit tests. Select the task in the list. In the right pane, click the **…** to browse the source repository and select the file **gulpfile.tests.js** in the Tests directory. Click **OK** to confirm:



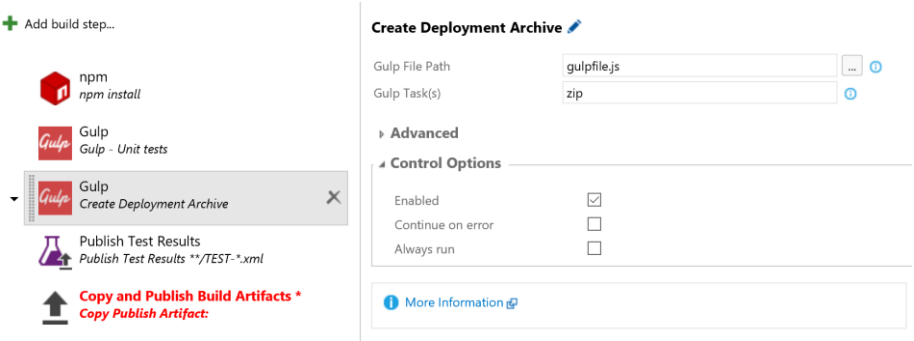
Then enter “tests” in the Gulp Task(s) field. It is the name of the task that is defined in the Gulp file.

You can also click on the icon after the title to give a more detailed description (for example “Gulp - Unit tests”).



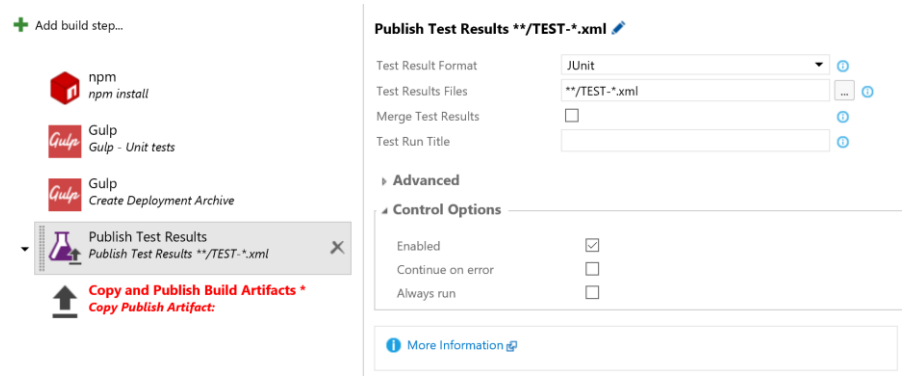
Gulp task #2

This task will produce a zip archive with the application and all its dependencies. It uses the default gulpfile.js located in the root directory and execute the task name “zip”:



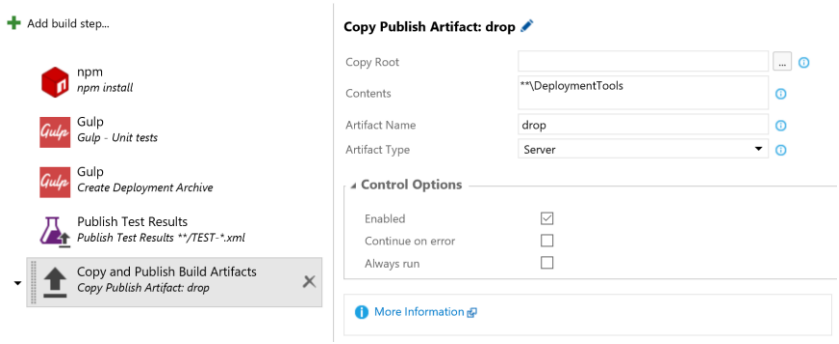
Publish Test Results task

This task will handle the XML file produced by the Gulp tests task and will import the test results in the build report. Just ensure that **JUnit** is selected as Test Result Format:



Copy and Publish Build Artifacts task

This task will copy the artifacts produced by the build and publish them so they can be reuse in other processes, like Release Management. In this case, we want to copy the DeploymentTools directory in a drop folder, on the build server (agent). Enter ****\DeploymentTools** for the Contents field, **drop** as Artefact Name and select **Server** as Artefact Type.

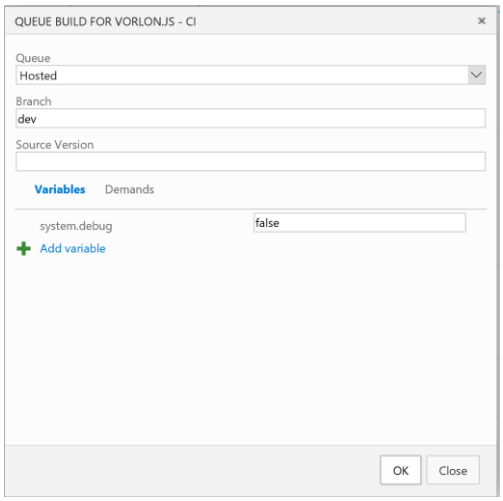


Save the build definition and start a new build

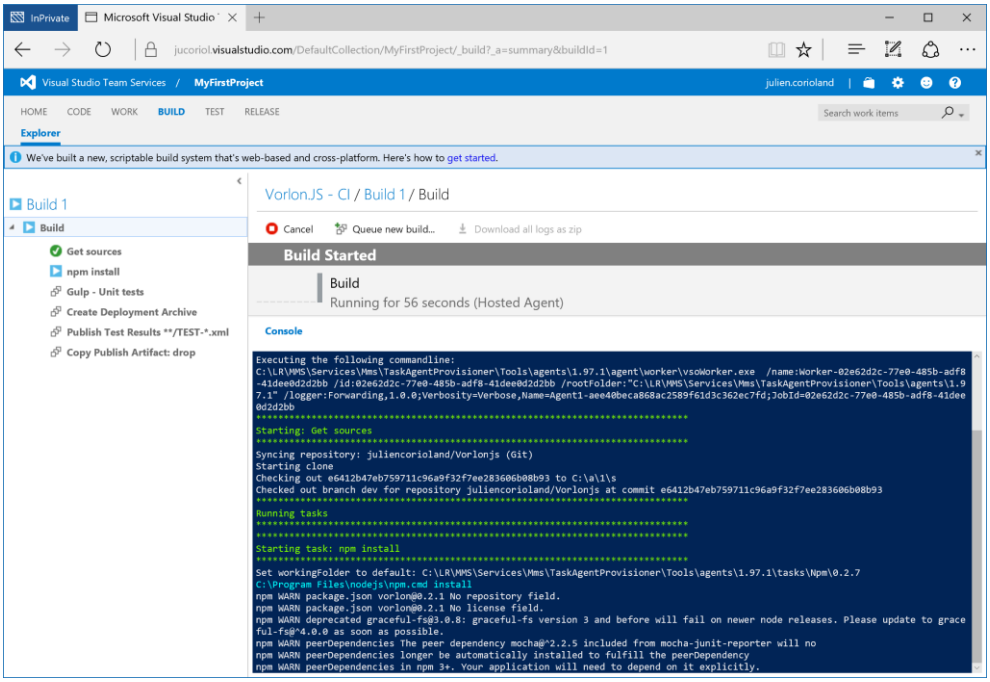
You are done for the workflow definition! ☺

Click the **Save** button in the toolbar and enter a name for this build definition (ex: Vorlon.JS - CI). Click **OK** to save:

You can now test this new build by clicking on the Queue build… button in the toolbar. Make sure that the Hosted option is selected for the Queue and dev for the Branch. Then click OK:



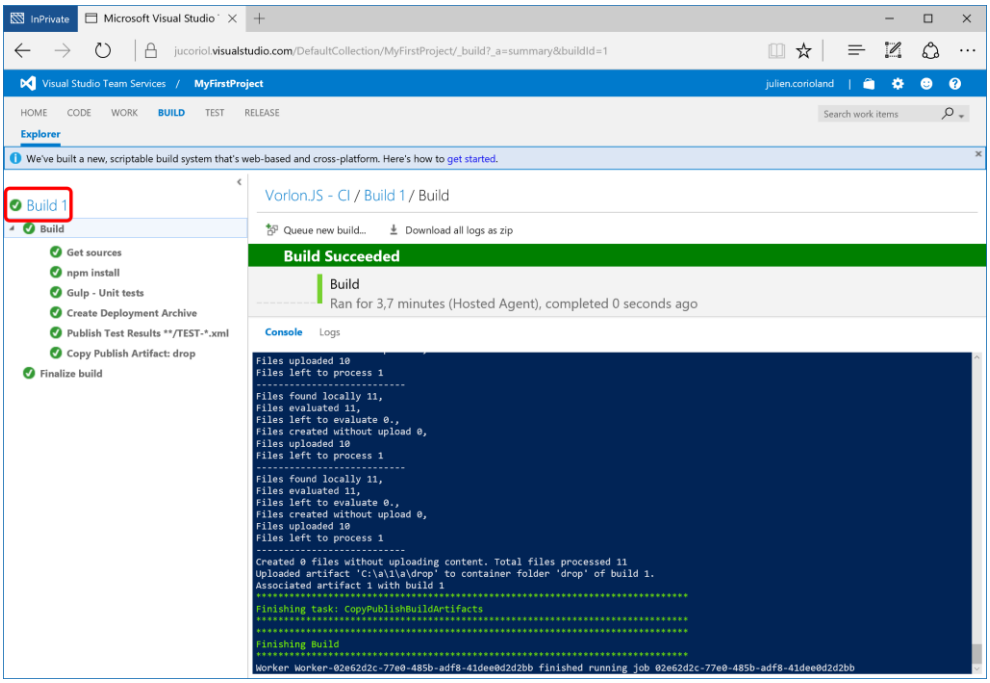
The build will start on an available agent provided by Visual Studio Team Services and you will be redirected to the build console:



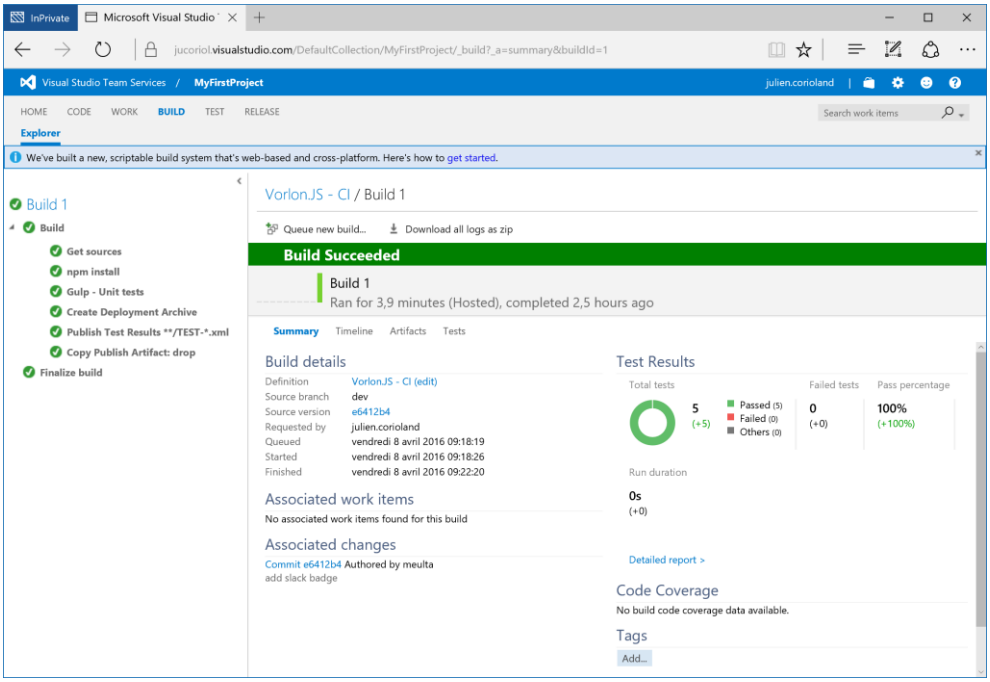
Wait until the build is completed.

Browse the build report

Once the build is completed, you can access the build report by clicking on the link on the left:

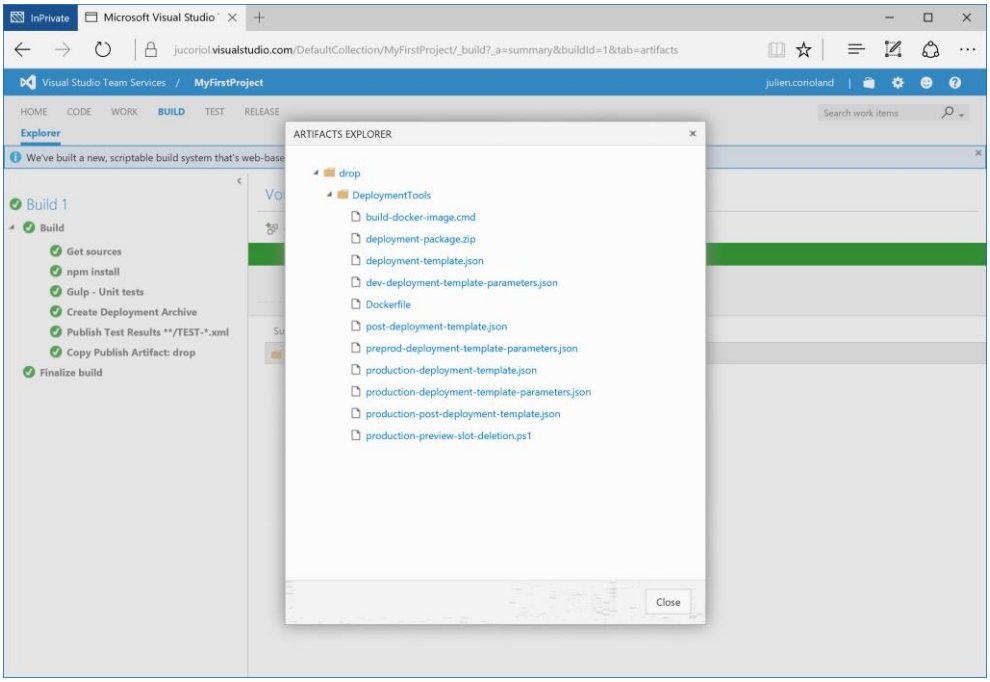


Then you can get all the details about the build execution:



As you can see, the **Summary** tab gives you essentials information like general information about the build and test results (totally integrated to the dashboard, thanks to the Publish Test Results task!).

In the Artefacts tab, you can browse all the content that has been dropped on the server and will be reusable with Release Management:



You can also download an archive with all build's execution logs by clicking on the **Download all logs as zip** button.

You have now implemented continuous integration on the Vorlon.JS project. In the next part, you will learn how to use Visual Studio Team Services Release Management to manage environment using Infrastructure as Code and deploy the application in Azure, automatically and continuously.

Release Management with Visual Studio Team Services

Introduction

In this part, you will learn how to use the Release Management feature of Visual Studio Team Services to deliver Vorlon.JS in Azure, continuously.

Release Management is a process that consists in using the artifacts produced by a build and use them to deploy the application in a succession of environment (like development, test, preproduction, production…).

It also helps to manage the workflow and the transitions between these environments. Consider you have three environments: **Development**, **Tests / QA**, **Production**.

The Development environment is owned by the developer's team and the application could be deployed each time a build succeeds, as it is not critical if a bug is introduced in this environment.

Then, a responsible identified in the team can approve the deployment to the next environment, **Tests**, where a new team of testers will execute some functional tests or load tests for example. If they find bugs in the application, they can reject the release and this version of the application will never go to **Production**. But, if the all tests are successful, they can also approve the release and start the deployment to the Production environment, etc.

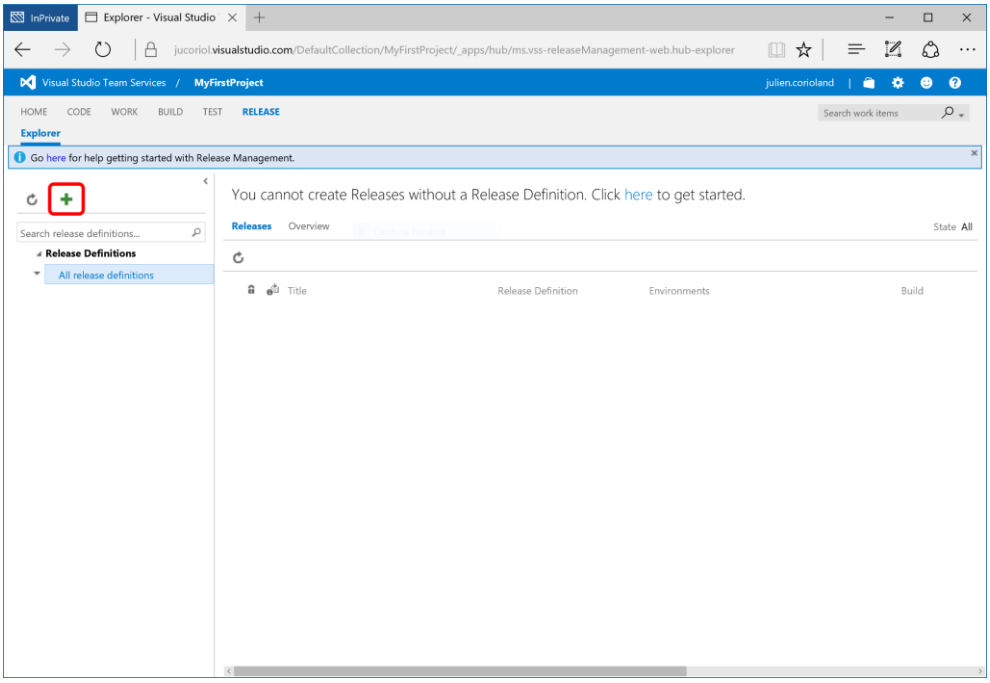
For this hands-on lab, you will only work with two environments:

- Development
- Preproduction

As soon as a new build is completed, a new release will be started and the application deployed into **Development**. Then, a member of the team (you ☺) will approve the build in **Development** to start the deployment into **Preproduction**.

Create a new Release Definition

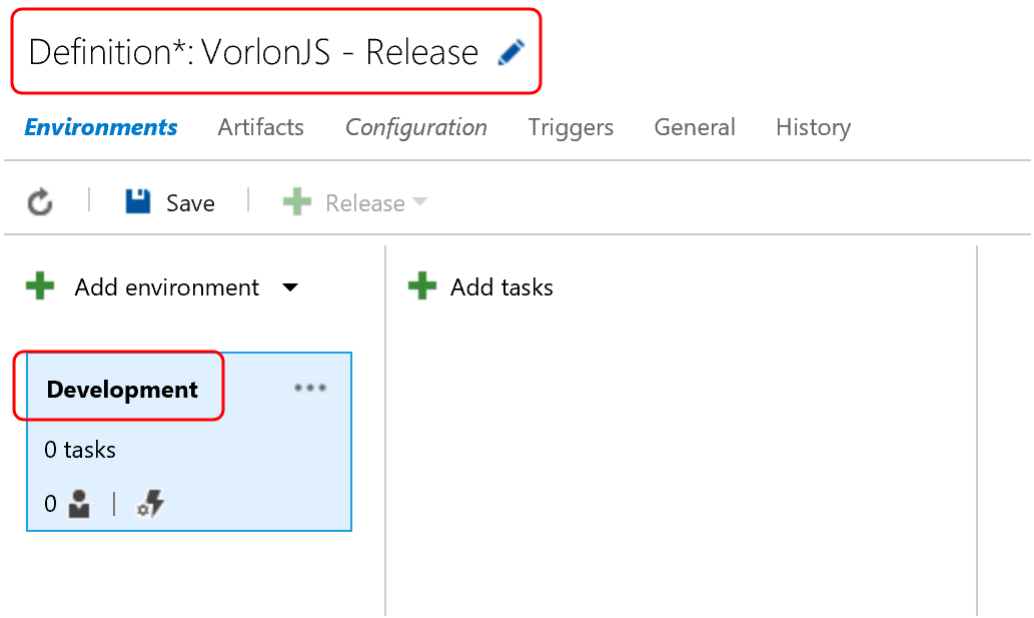
Go to the RELEASE tab of Visual Studio Team Services. Before creating a new release, you need to define a release definition, which is basically the definition of the environments and the different steps to execution to deploy the application. Click on the + button in the left pane to create a new release definition:



Like for the build definition, you can start from an existing template, but in this case, choose to start from an empty one.

Define the environments

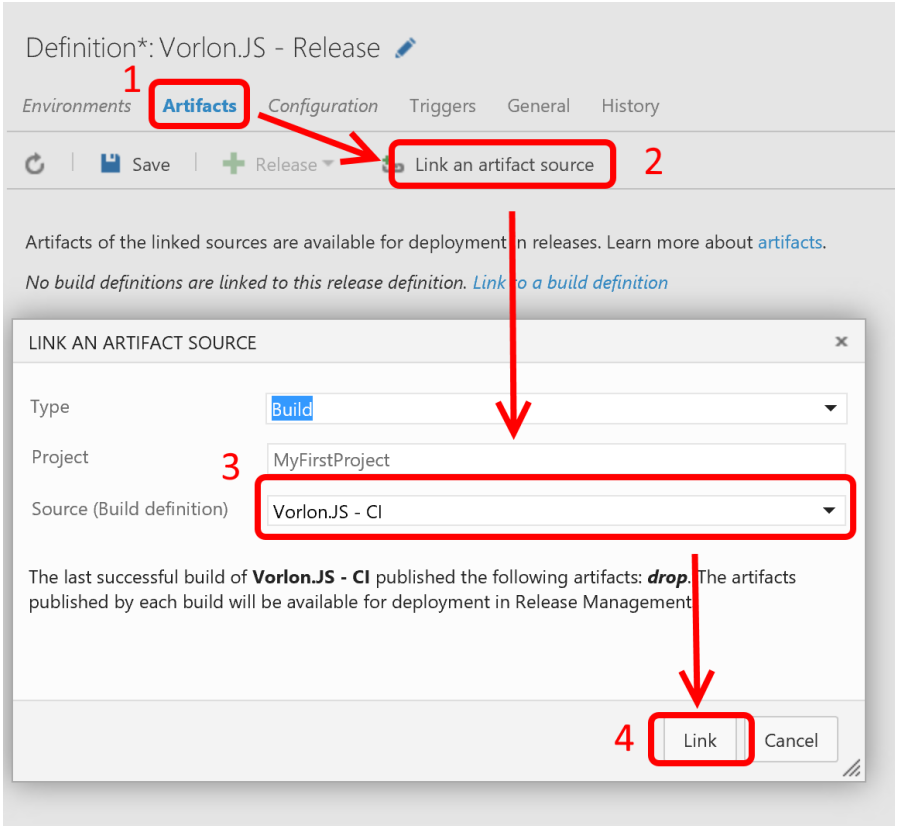
First, you can give a name to the release definition, for example “Vorlon.JS - Release". Then, you can rename the default environment into Development:



Now, you have to link a build definition to this release definition. This step is mandatory because it is the one that indicates to Release Management where are the build artifacts that will be available during the release.

Link the release definition to a build definition

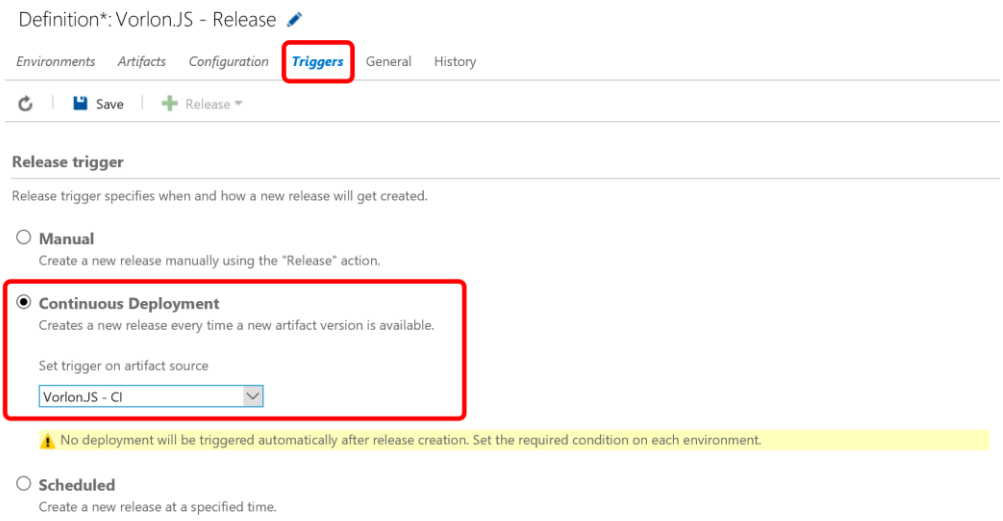
Go to the Artifacts tab and select the build definition that you have created during the previous part:



Click on the **Link** button to validate the association to the build artifacts.

Configure the trigger

A release can be started as soon as the linked build is succeeded. To do that, go in the **Triggers** tab and check **Continuous deployment** option and select the build you have linked to the release definition as source:



Don’t worry about the warning, you will correct the error late in this part.

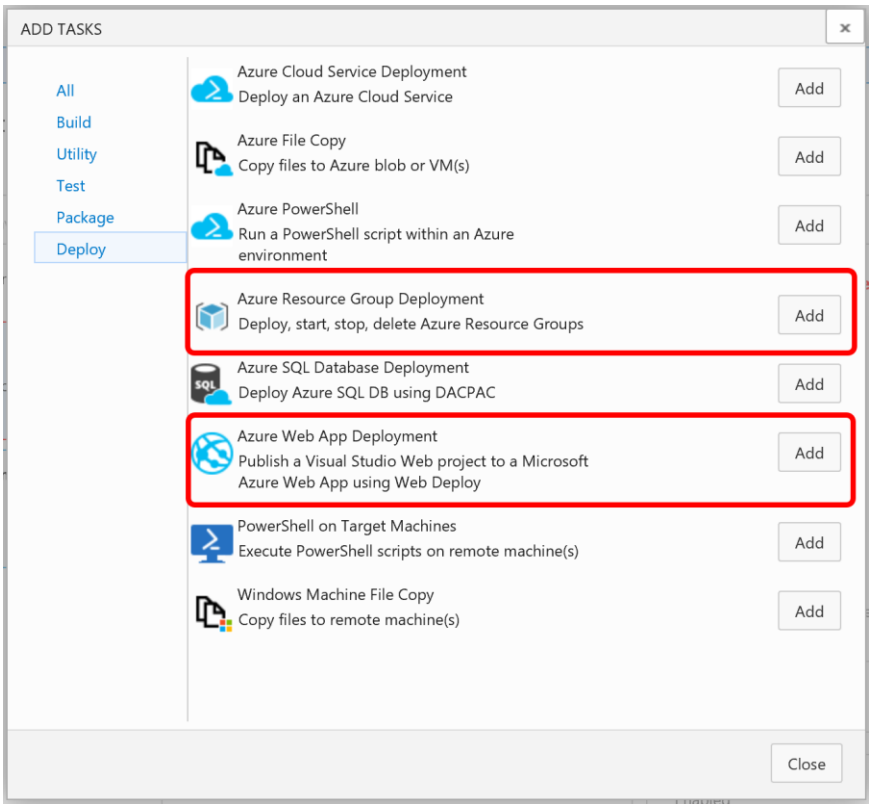
Define the release workflow for each environment

In this case, the two environments release workflows are composed by the same steps:

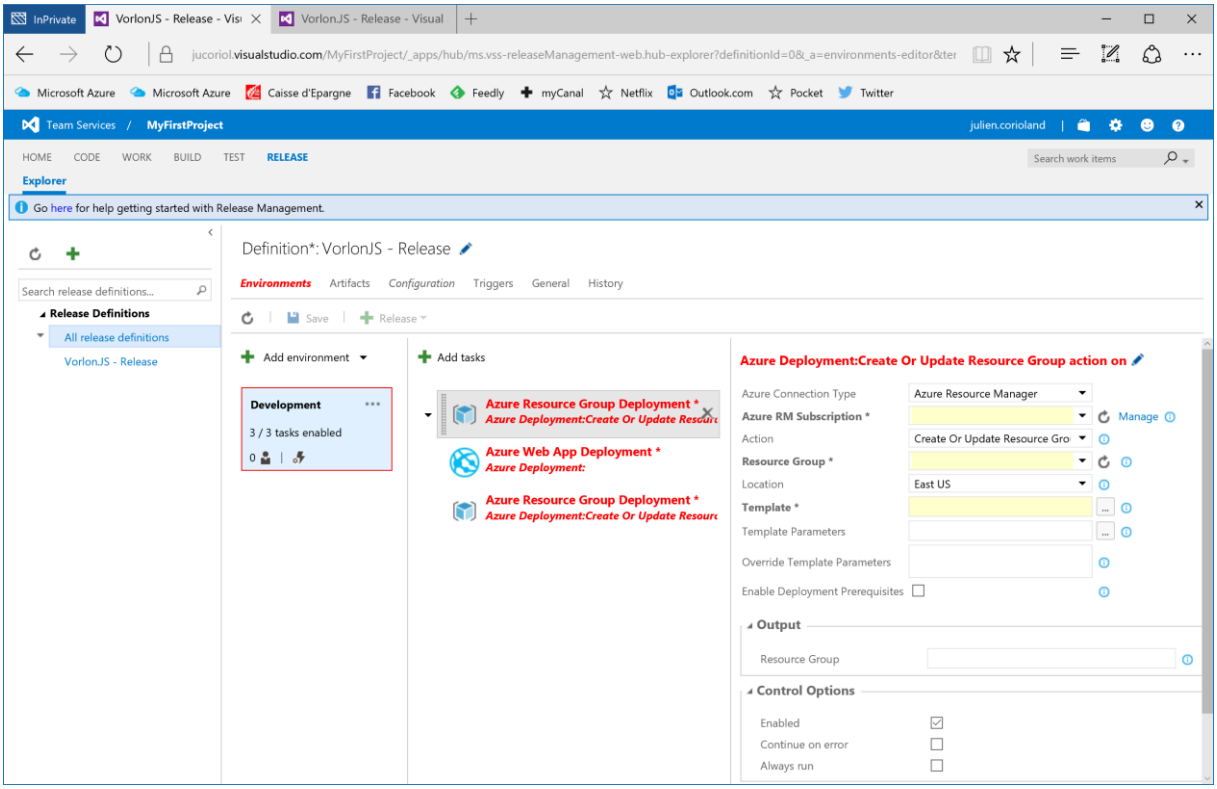
- Azure Resource Group Deployment: this one will use an Azure Resource Manager (ARM) template to create the environment (which is composed by a simple Azure App Service WebApp)
- Azure Web App Deployment: this task will be responsible for deploying the application package (created during the build) to the Azure WebApp
- Azure Resource Group Deployment: this one will apply another ARM template to the resource existing group, to apply a post-deployment configuration to the web

Note: *resource group deployment using ARM templates are idempotent so you can apply the same template many times, the result will be the same.*

Select the **Development** environment in the left pane and click on the **+ Add task** button. A popup will open and you will be able to add the three tasks described above (note that the interface really looks like the build definition one!):



After adding the three tasks, click on the **Close** button. You should have something like that in your release definition:



Connect your Azure Subscription

Because you are going to deploy the application into Azure, you need to import your information into Visual Studio Team Services, as you have done it with GitHub in the previous part.

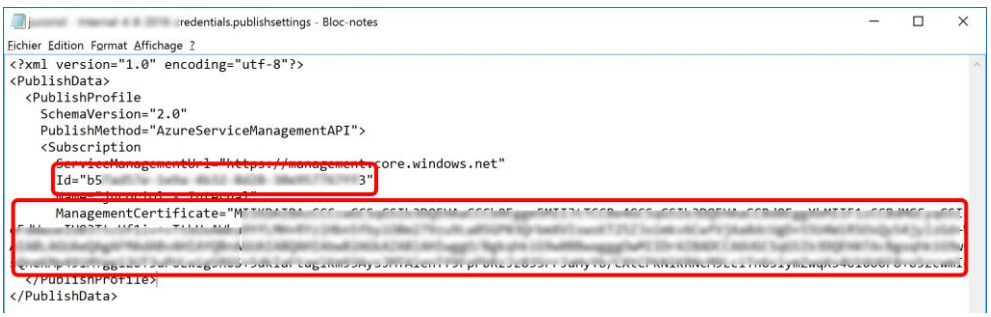
Select the first task in the middle pane and click on the **Manage** link on the Azure RM Subscription line, in the right pane. It will open the service endpoints management page of the project.

For this lab, you will need to add two Azure endpoints, one for the **Classic** deployment model (old Azure APIs, used by the Azure Web App Deployment task) and one for the **Azure Resource Manager** model (new powerful Azure APIs, used by the Azure Resource Group Deployment tasks).

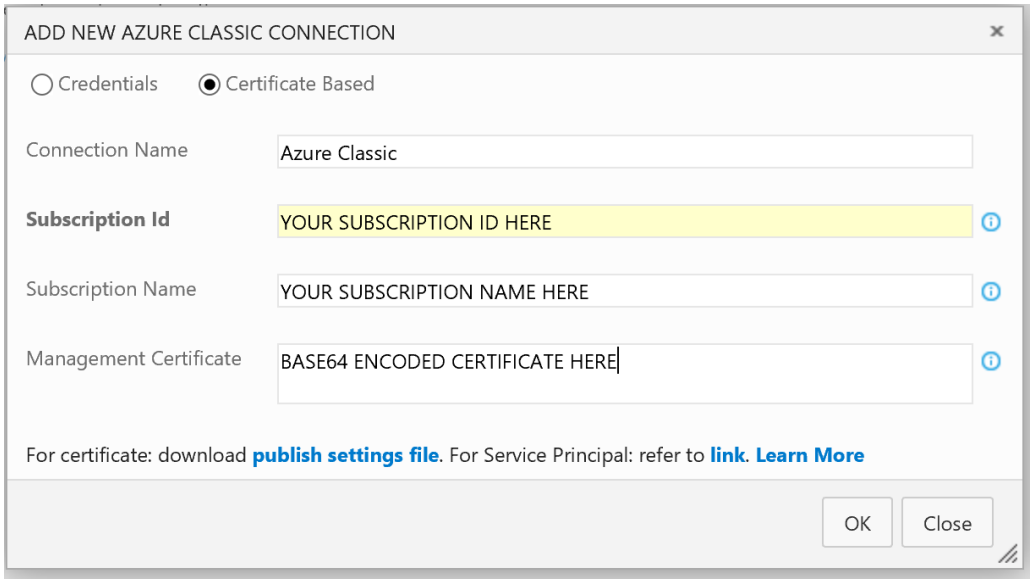
Add an Azure classic endpoint

Click the **+ New service endpoint** button in the left pane and choose **Azure Classic**. In the window that opens, click on the **publish settings file** link to download your subscription information. You may be asked to log into your Azure account. Once downloaded, you can open the file with Notepad.

The two relevant information in this file are the subscription **Id** and **ManagementCertificate** properties:



Go back in the VSTS management and select the Certificate Based option. Then, enter the subscription id, name and management certificate and click OK to add the endpoint:



Add an Azure Resource Manager endpoint

To enable Azure Resource Management in Visual Studio Team Services, you need to configure a Service Principal. Download this PowerShell script and execute it on your machine.

***Note:** If an error occurs, you need to enable script execution on your machine using the **Set-ExecutionPolicy** cmdlet.*

Enter the name of your subscription and the password you want to use for the service principal. In the window that opens enter your Microsoft Azure credentials. Wait until the service principal is generated. Once done, all the information you need are generated in the output window:

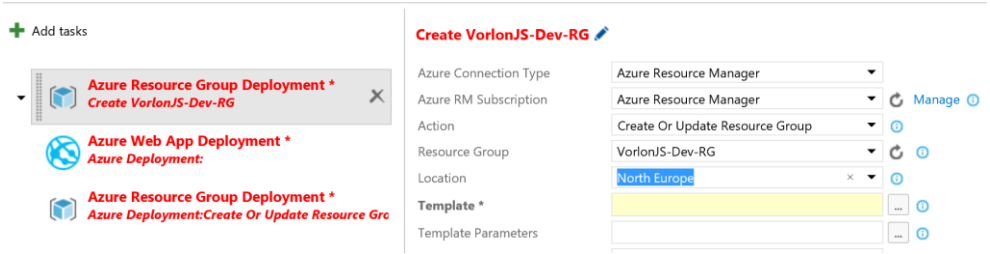
```
Copy and Paste below values for Service Connection
********************************************************************************
Connection Name: Pass Azure(SPN)
Subscription Id: c9edf49b-ab2c-43c3-aa72-be088368c22b
Subscription Name: Pass Azure
Service Principal Id: 893bb33a-1f0d-4ba0-929f-263a44be912c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

Go back in the service endpoint management in VSTS and click the **+ New service endpoint** button in the left pane and choose **Azure Resource Manager**. In the window, enter the information pasted from the PowerShell.

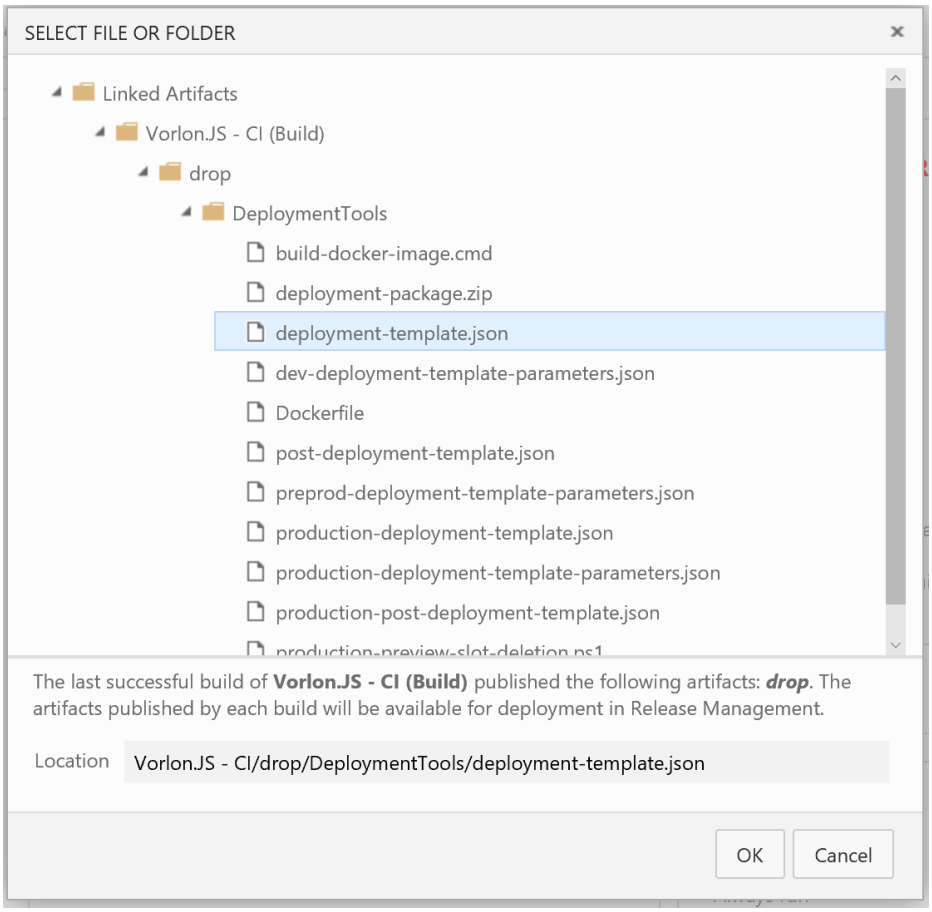
Click on the **OK** button to add the endpoint. Once added, you can go back on the release definition.

Configure the tasks in Development environment

Refresh the Azure connection. Your Resource Manager endpoint should be available now. Give a name to the resource group to create and choose the data center where you want to deploy the application:



Then, click on the … button on the Template line. It will open the last build artifacts that have been produced. Browse the **DeploymentTools** folder and select the **deployment-template.json** file:

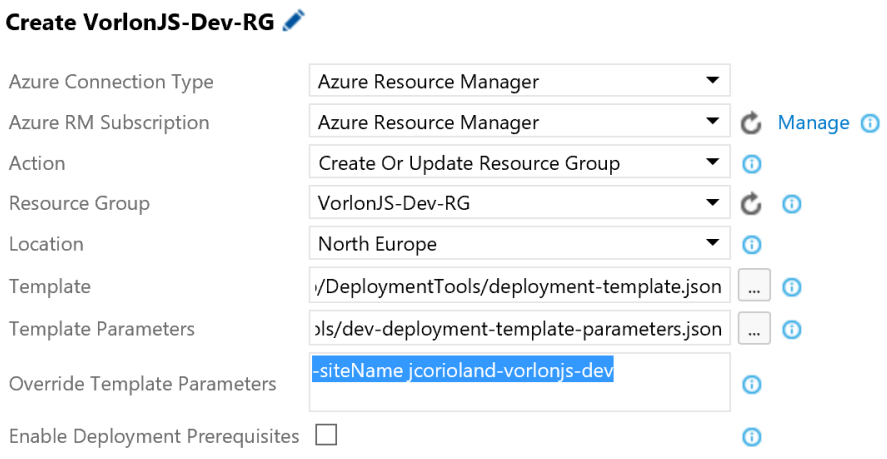


Do the same for the template parameters and select the **dev-deployment-template-parameters.json** file.

As web app name should be unique in Azure, you should override the default name provided by the parameter file. To do that, add the following in the Override Template Parameters field:

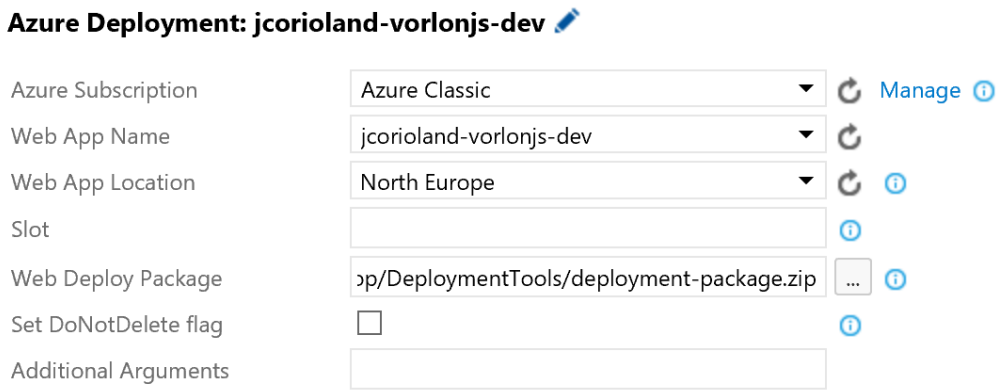
-siteName **YOURNAME**-vorlonjs-dev

The configuration should look like the following:

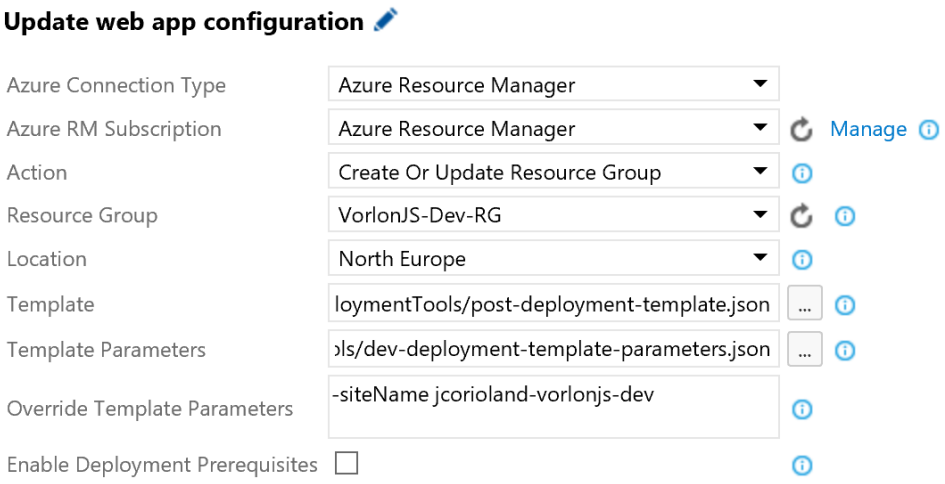


Now you can configure the Azure Web App Deployment Task. This time, choose the Azure Classic connection. Enter the name of the web app you have chosen in the previous step (while overriding the siteName parameter) and choose the same data center than before.

For the Web Deploy Package, browse the build artifacts and select the deployment-package.zip file located in the **DeploymentTools** folder:



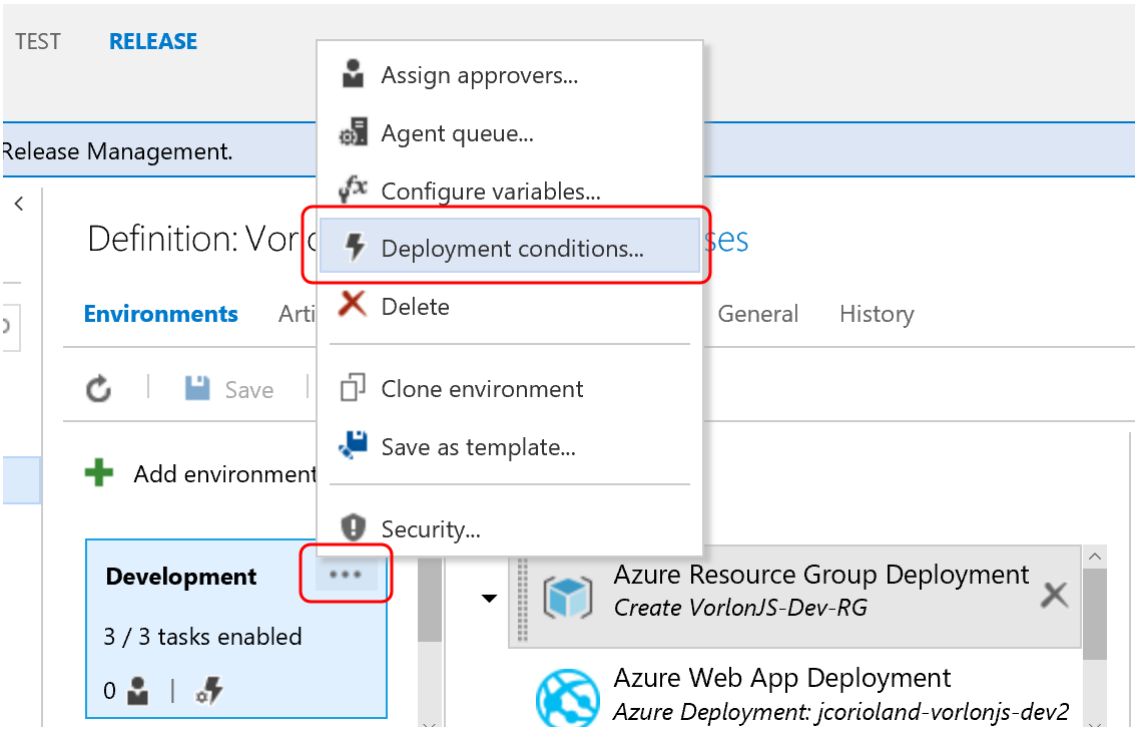
For the third step, repeat the same operation than for the first one, instead that this time you will use the **post-deployment-template.json** file:



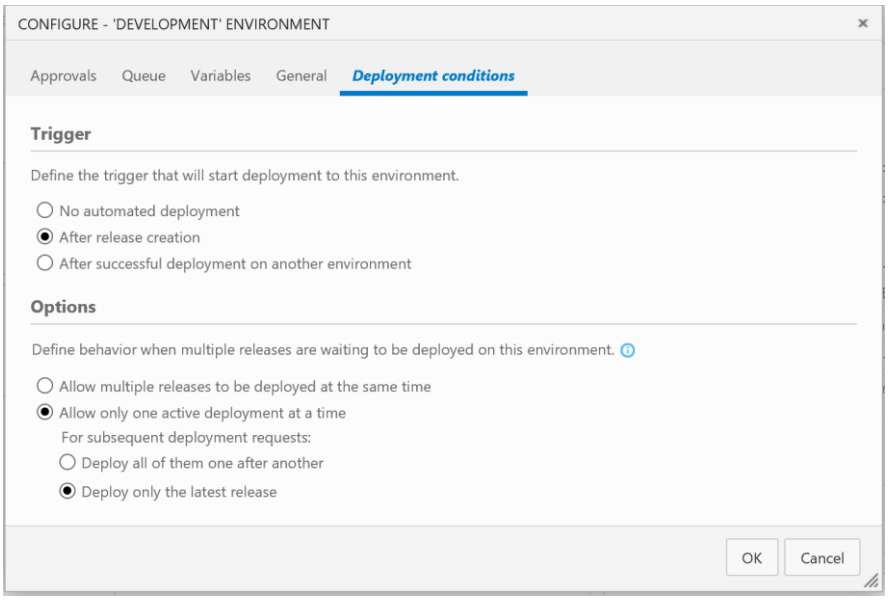
Make sure to use the same Resource Group, Location and override parameters than the first step.

Set the deployment condition for the Development environment

Click on the … button on the Development environment tile to open the contextual menu and choose Deployment conditions:



In the **Trigger** section, select **After release creation** to make sure that the deployment will start as soon as the release is created (i.e. as soon as a build is successfully completed). Click the **OK** button to validate:

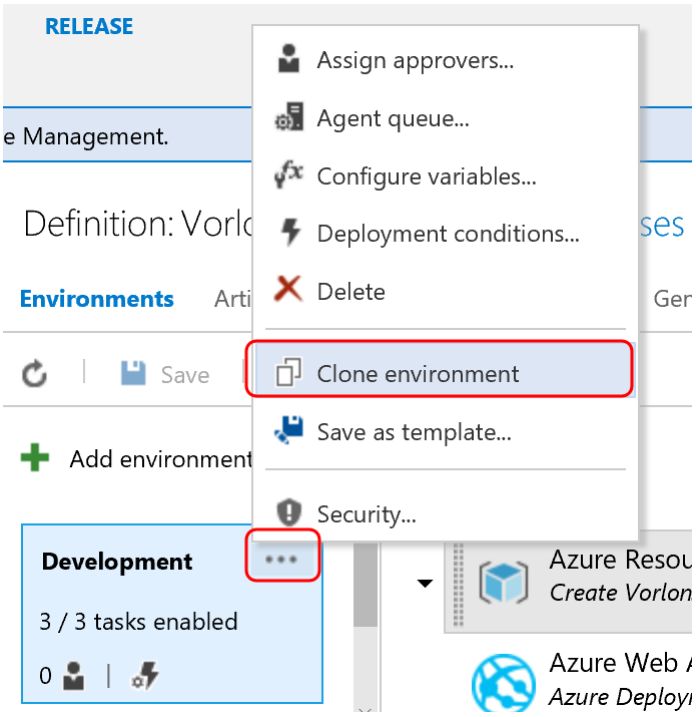


Add a Preproduction environment

The Preproduction environment use the same three tasks than the **Development** environment:

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

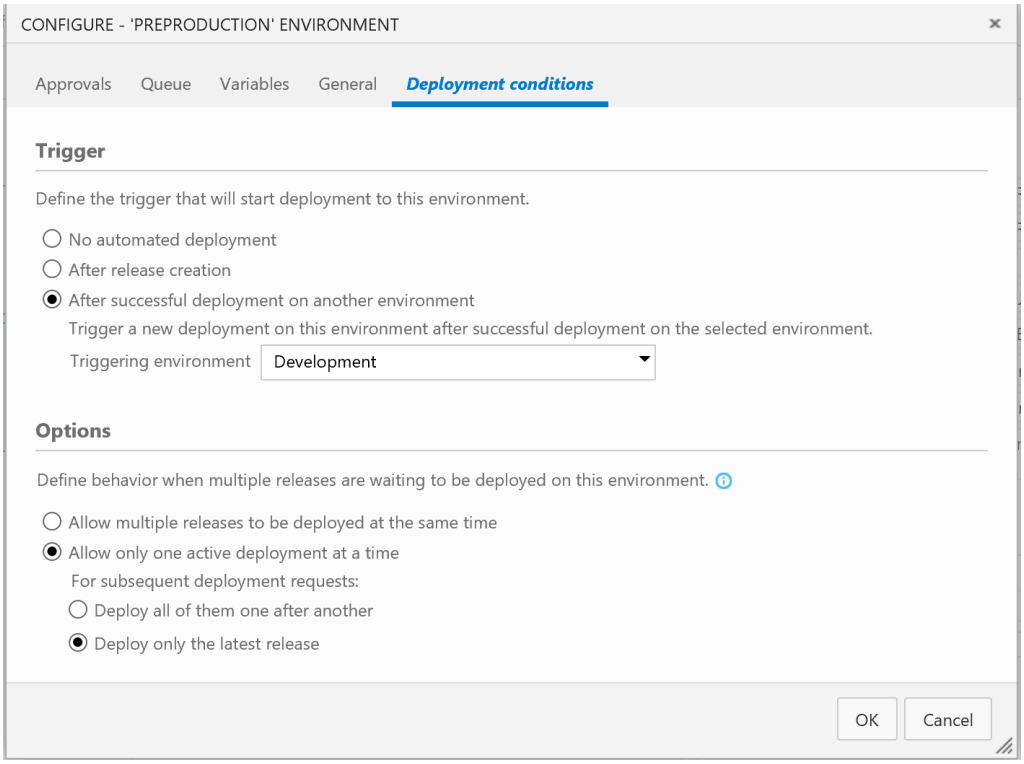
To create it quickly, you can use the clone function on the Development environment:



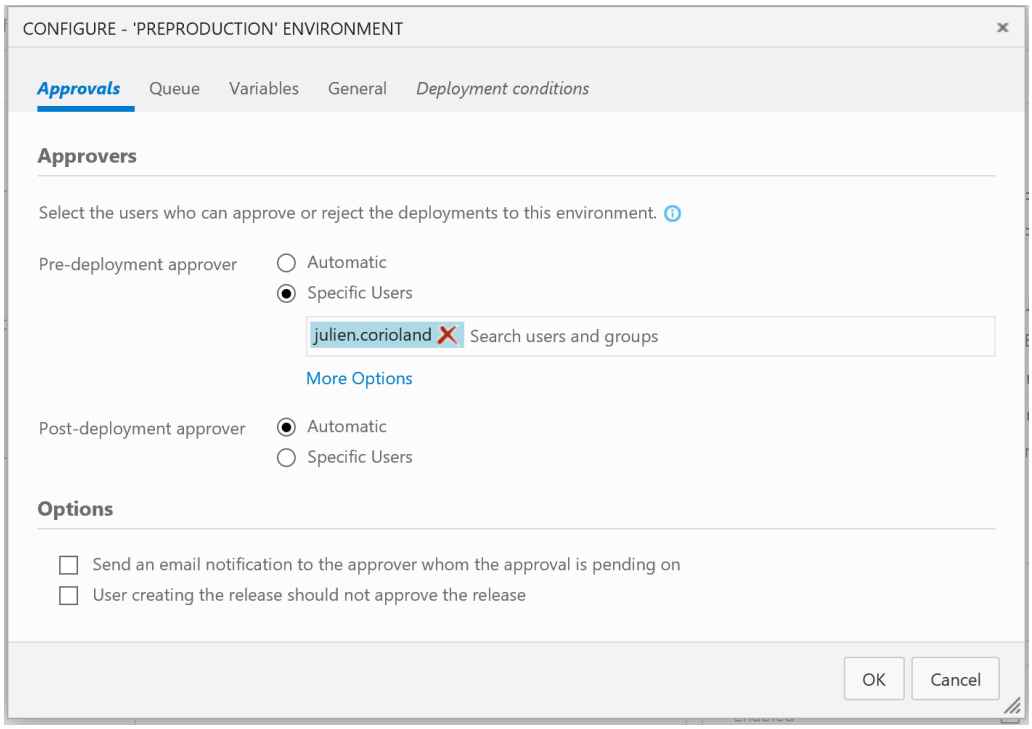
Make sure update all the fields for Preproduction and especially to use the **preprod-deployment-template-parameters.json** parameters file and to suffix your web app by **preprod**, instead of **dev**.

Set the deployment condition for the Preproduction environment

On the Preproduction environment tile, click on the … button to open the contextual menu and select Deployment conditions… In the **Trigger** section, make sure to select After successful deployment on another environment, and select the Development environment in the list:



Click on the **Approvals** tab and add your user as a pre-deployment approver so the deployment will not start to the Preprodution environment without your GO:

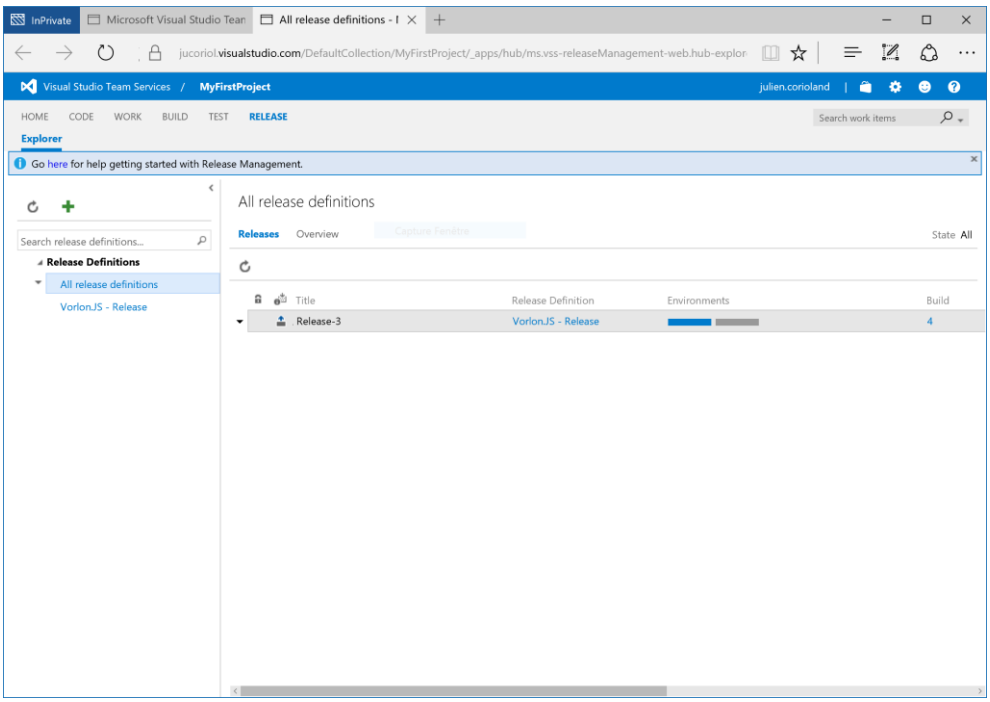


Click on the **OK** button to validate.

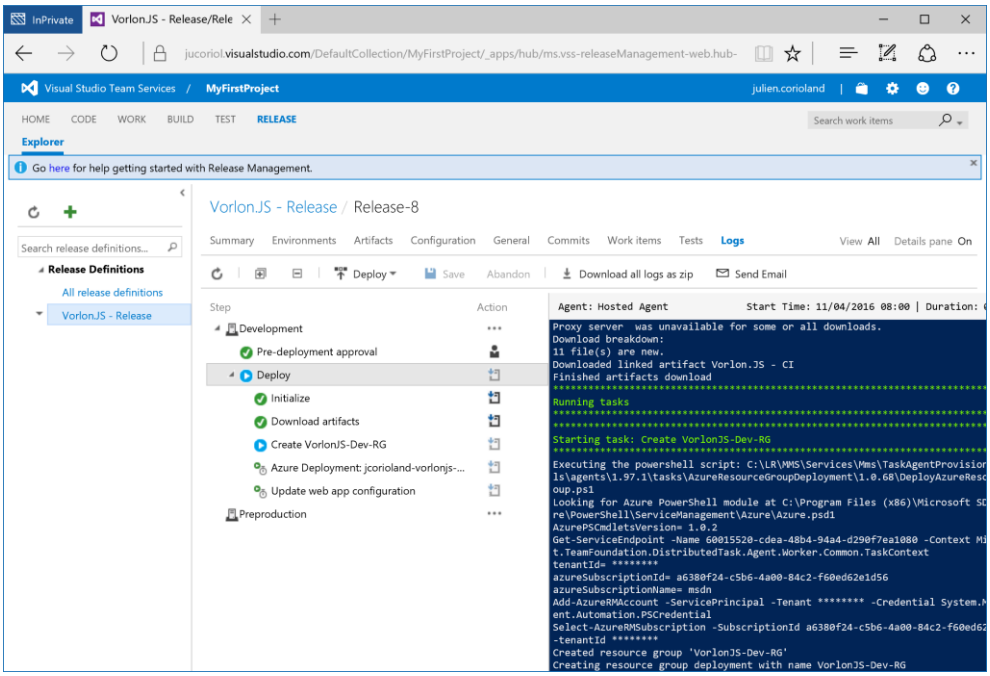
Save the release definition and create a new release

You can now save the release definition, as it is completed! Then, go back in the BUILD tab and queue a new build, as you have done before.

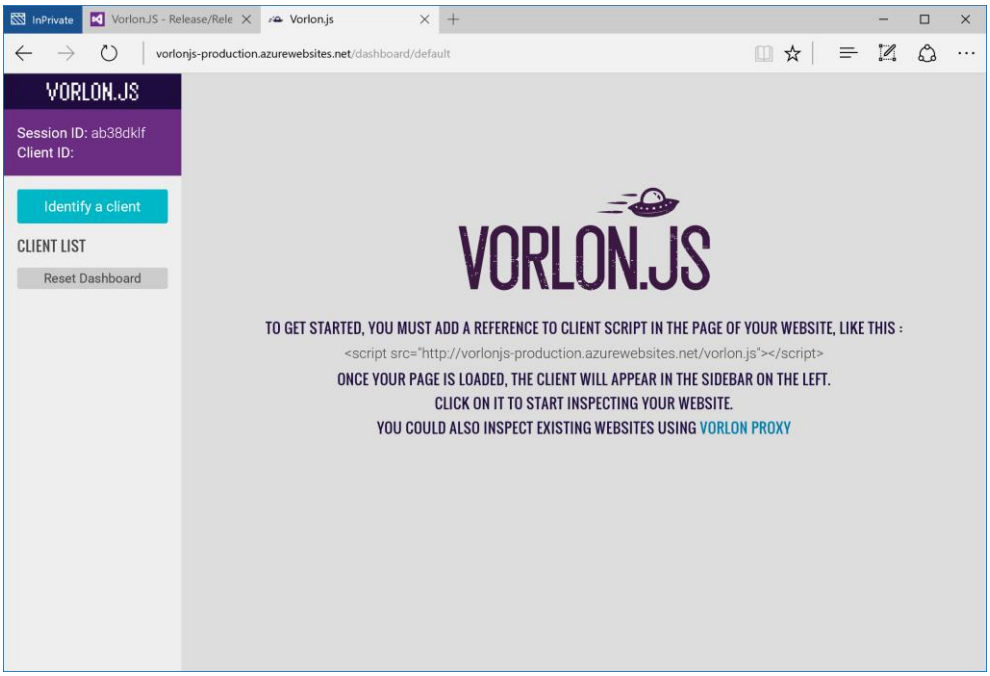
When the build will be completed, a new release will be automatically created and the deployment will start into the Development environment:



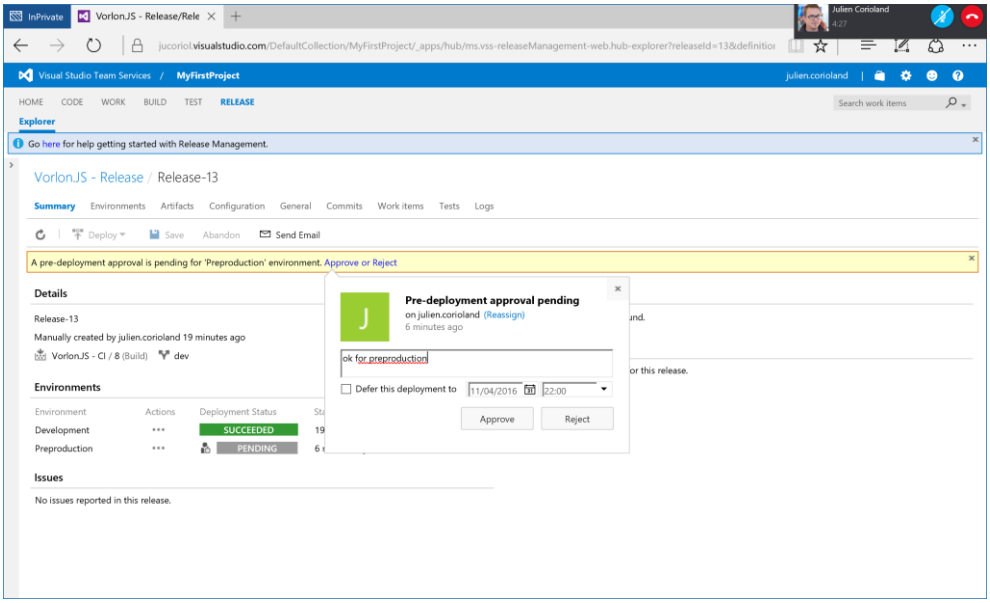
If you double-click on the release, you can get details like the step-by-step logs:



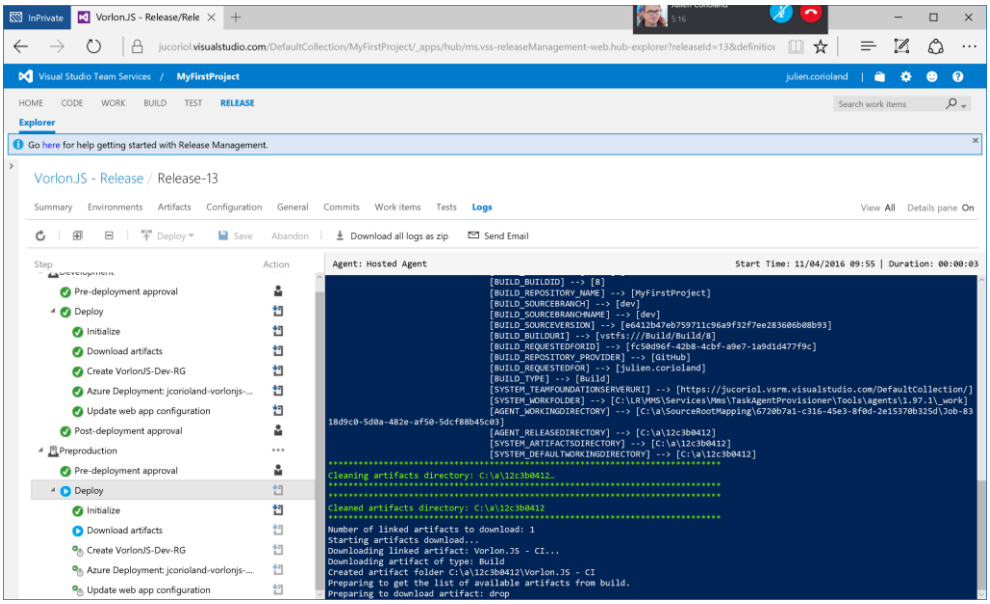
As soon as the release is completed in the Development environment, you can go test the web app which is hosted on http://**the_name_you_choose_for_your_site**.azurewebsites.net:



If you go back in Visual Studio Team Services Release Management and select the current release, you will see a message that indicates your validation is pending before going to **Preproduction**. Click on the link and then **Approve** the deployment.



It will start to deploy the application into **Preproduction**!



Content by Julien Corioland, Technical Evangelist, Microsoft France
Contact me on Twitter @jcorioland for any question.
Vorlon.JS DevOps blog post series: https://aka.ms/devopsvorlonjs