Laboratory for App Services Dev.

Made by Jhosua Arias Quintero.

What are you going to learn with this laboratory?

- To develop a *Function App* with *PowerShell* scripts.
- To create an *App Registration* for assigning permissions to the *Function App* within the Azure Subscription.
- To use basic *Az module* commands on *PowerShell* scripts.
- To test *HTTP triggered functions* from the *Portal*.
- To develop an *Angular* project for consuming our *Function Apps* in a local environment.
- To create a project on *Azure DevOps*.
- To bind the *DevOps project* with a *GitHub Repository*.
- To configure a *build pipeline* and a *release pipeline* for the *Angular project*.
- To bind the *DevOps project* with a *Linux Web App*.

In this laboratory, you are going to create a *Function App* with two (2) *HTTP Triggered functions* being capable to turn on/off an *Azure Virtual Machine*. Then, you are going to create an *Azure DevOps project* with a *build pipeline* and a *release pipeline*. In order to *get*, *build* and *deploy* an *Angular application* into a *Web App*, every time a *push* is made to the *repository*. At the end of the laboratory, you should be able to use the *Angular Web App* to make a *call* to the *Function App* for turning on/off the Virtual Machine.

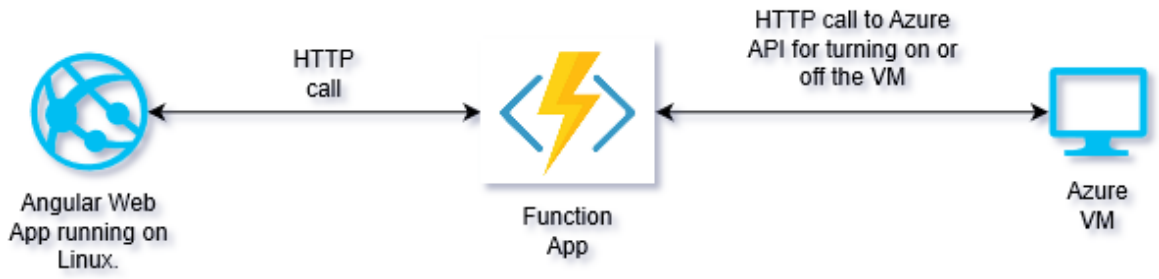


Table of Contents

Section 1: Resource Creation.

In this section, you are going to create all the required *Azure resources*, to use in the whole laboratory.

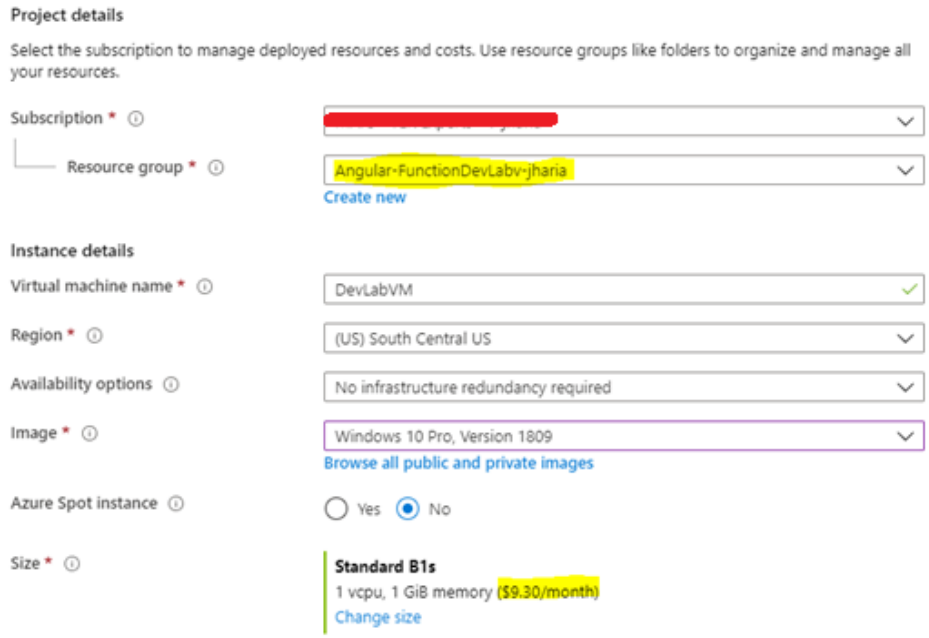
Start by creating a *Resource Group*, named "Angular-FunctionDevLab", to hold our *Windows resources*. Also, create another *Resource Group* named "Angular-FunctionDevLabLinux" to hold our *Linux resources*. This is because both, *Windows* and *Linux resources*, might have some conflicts when *living* in the same *Resource Group*.

In summary, we are going to create the following resources:

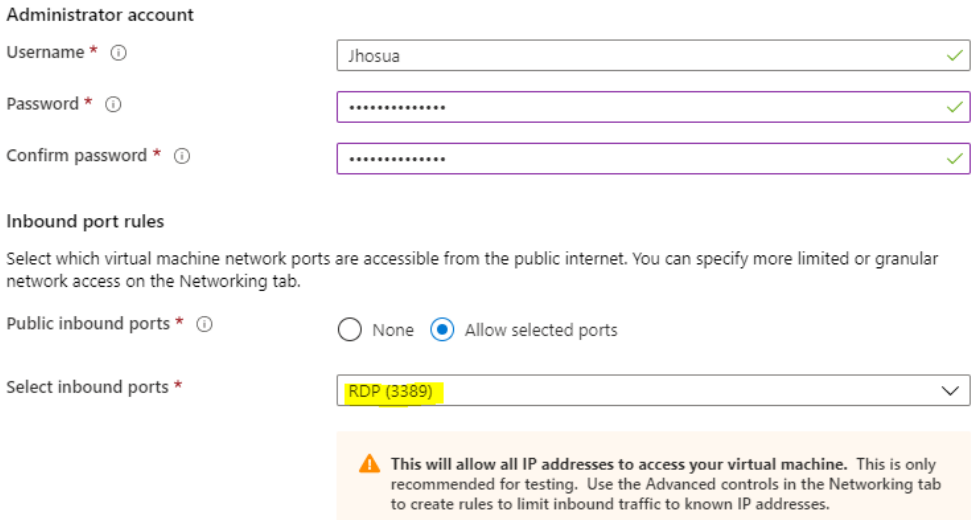
- Function App.
- Storage account.
- Application Insights for the Function App.
- Consumption Plan or App Service Plan.
- Linux App Service Plan.
- Linux Web App.
- Azure Virtual Machine.

A. Virtual Machine Creation

This resource is needed to test other resources' functionality. It is meant to be turned on or off. There are no specific configurations for this resource for making it work. Just try to minimize costs while creating it and be sure that all the required resources are in the respective *Resource Group* (*Disks, VNET, IP, Network Interface*). You can even use *RDP* to test the *VM's* availability.



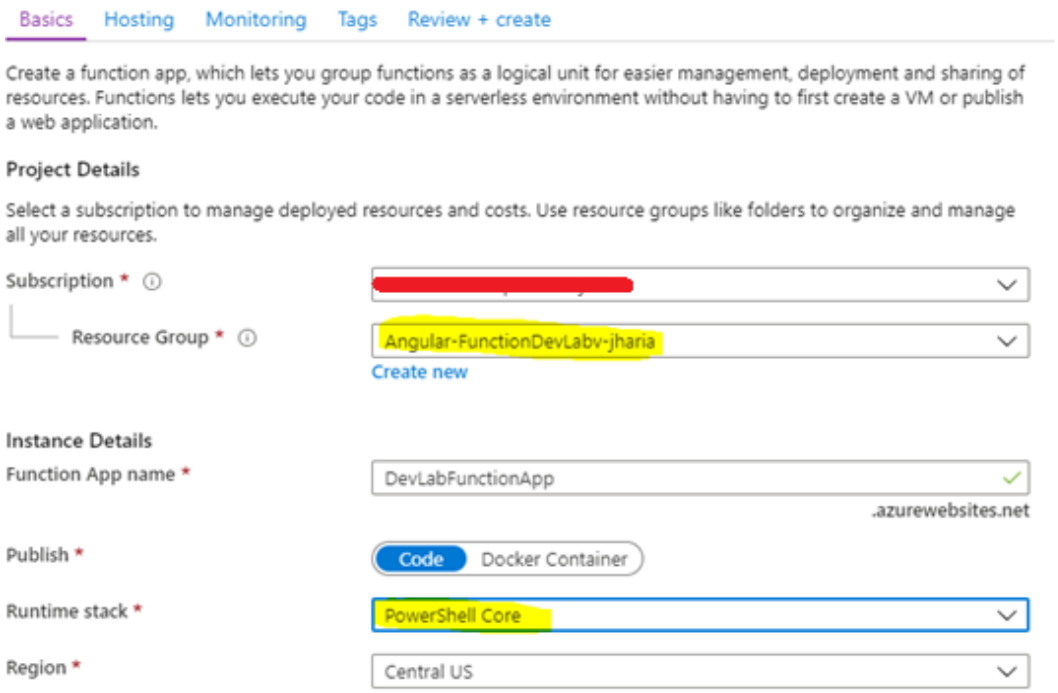
If you want to access your *VM*, in order to check its availability, you must enable the *RDP port* and *create* an *administrator account*.

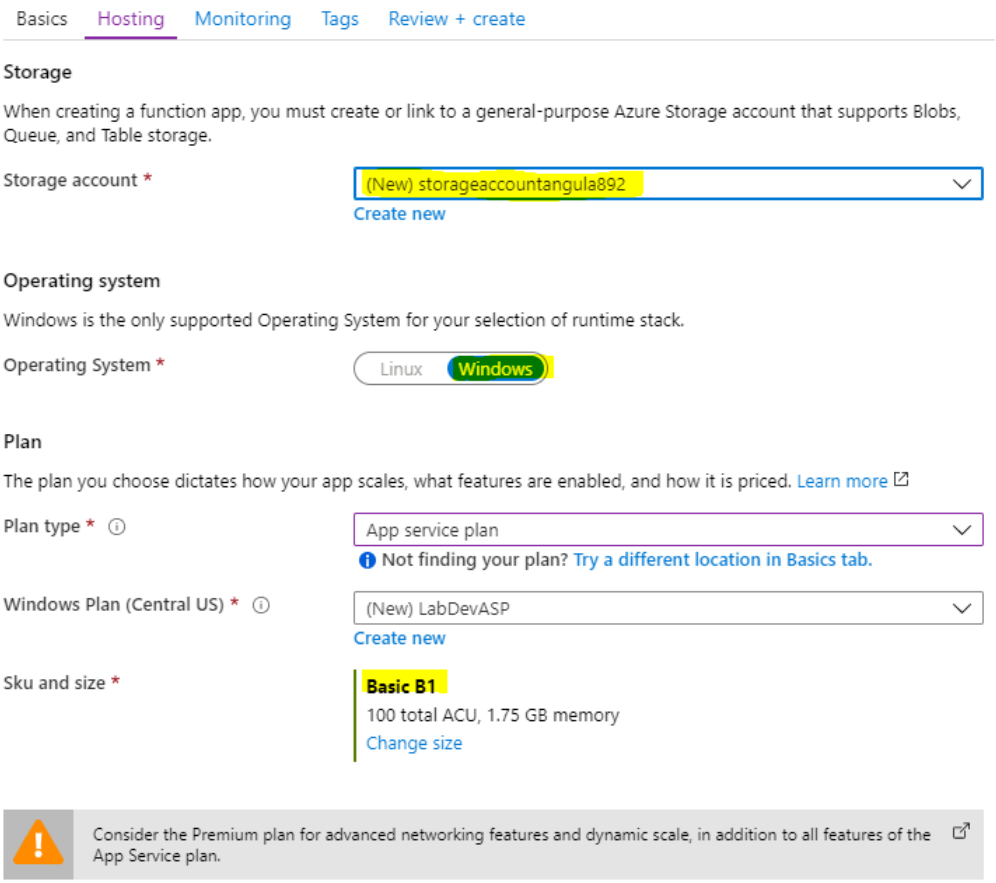


B. Function App Creation

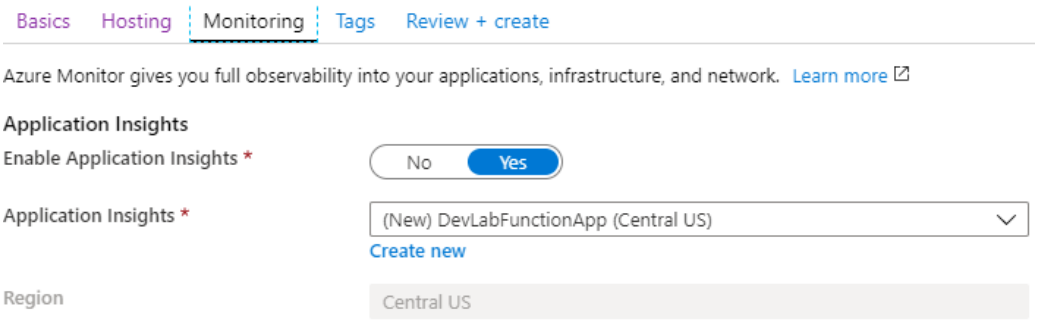
This resource is needed for establishing communication with the *Azure's API*, in order to turn on/off the *VM*. This *Function App* requires to be *created* with a *PowerShell stack*. This is to make usage of the *Az module*, which is only available on *PowerShell*. You can find more information about the *Az Module* here.

Below, you can find the basic configurations for the *Function App*:





- The *Storage account* is used to store the *Function App's* content, such as *scripts*, *dependencies*, and *configurations*.
- Notice that it is only allowed to use *Windows environments* while using *Powershell*.
- For this Lab, we chose to use an *App Service Plan* instead of a *Consumption* one. What are the differences between both *plans*? What about a *Premium Plan*?

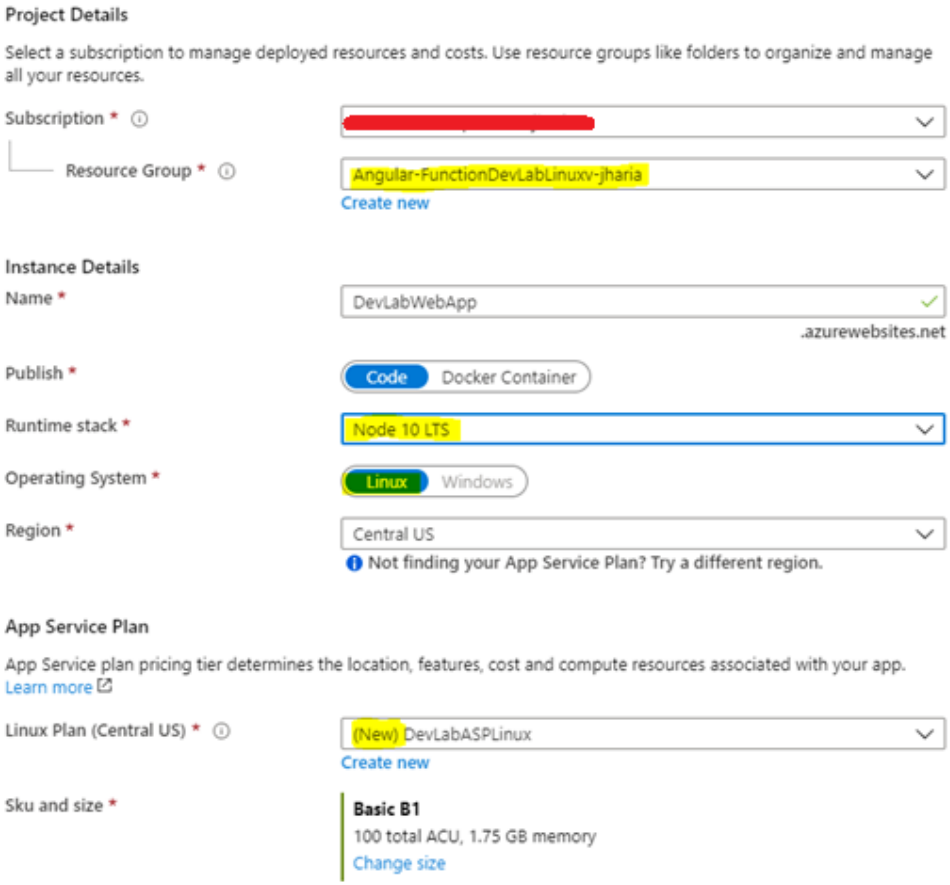


If you want to *debug* your *Function App*, then create an *Application Insights* resource.

C. Linux Web App Creation.

This resource will host your *Angular application*. This *Web App* is going to be the "*Consumer*" of your *functions*.

Below you can find the Web App basic creation configuration:



Note: Application Insights for this resource is optional.

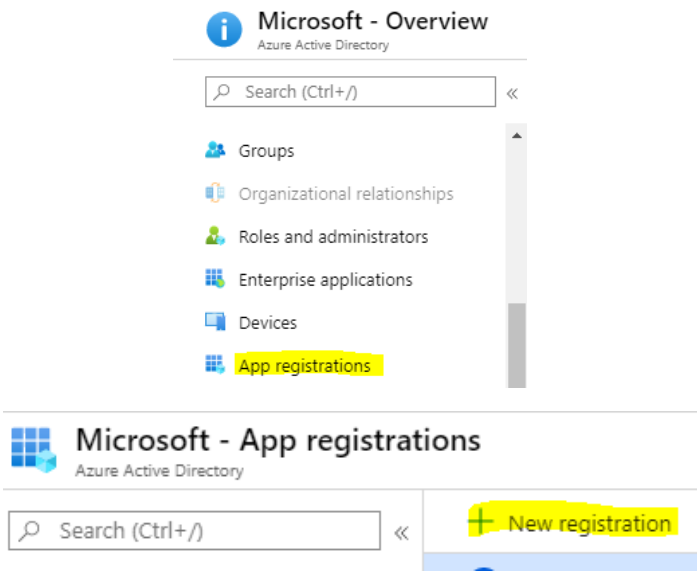
Section 2: Function App's Development.

In this section, you are going to provide *access* and *permissions* over the *Azure's subscription* to the *Function App*. Also, you are going to create two (2) *HTTP triggered functions* to turn on/off your *VM*.

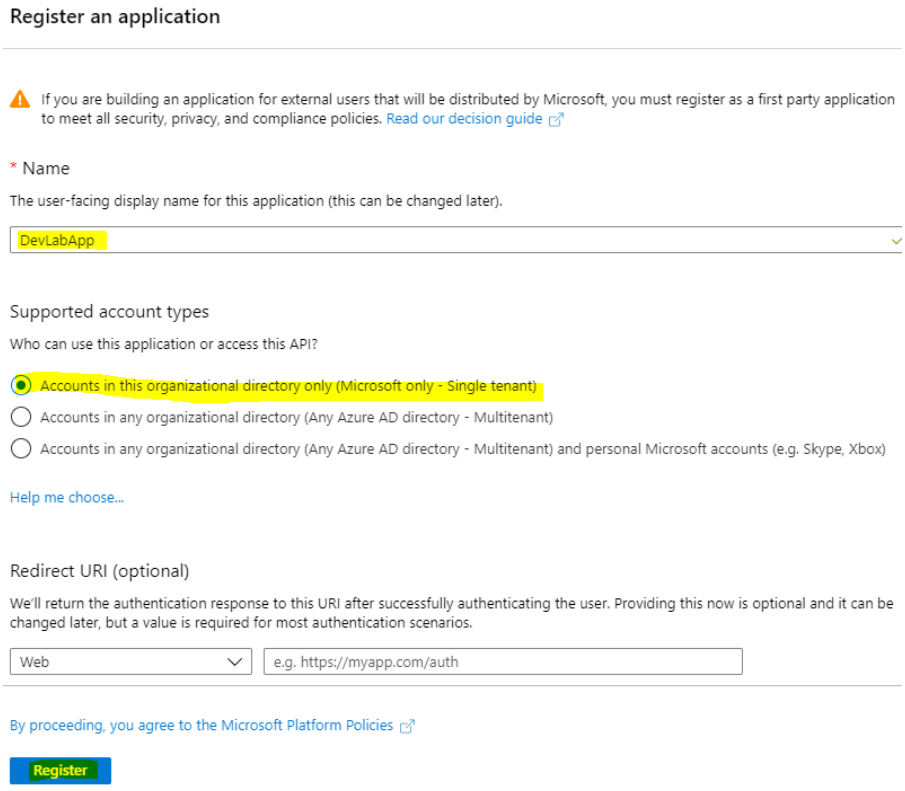
A. App Registration with Azure Active Directory.

In order to not compromise your *Azure subscription,* by *hard-coding* the *username* and *password* on the *functions*, you can create an *App Registration* on *Azure Active Directory* and give to your *Function App* the required *permissions* to *manage* an *Azure VM, allocated* within the *subscription*.

- First, go to the *Azure Active Directory* section, from the *dashboard*.
- Once you are there, click on "*App registrations*" and then on "*New Registration*".

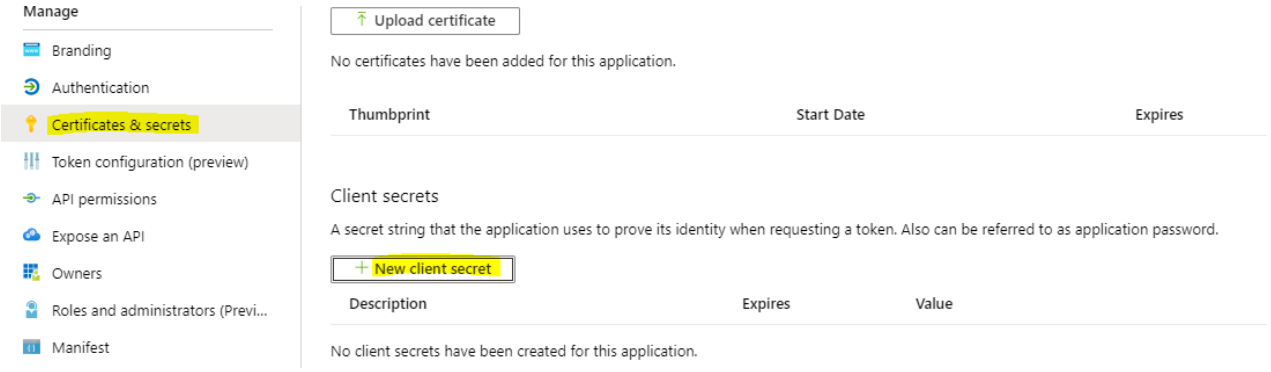


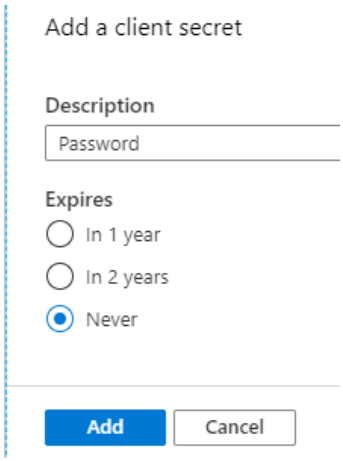
- Name your *App registration*, choose *Single Tenant* as the supported account types and click on *Register*.

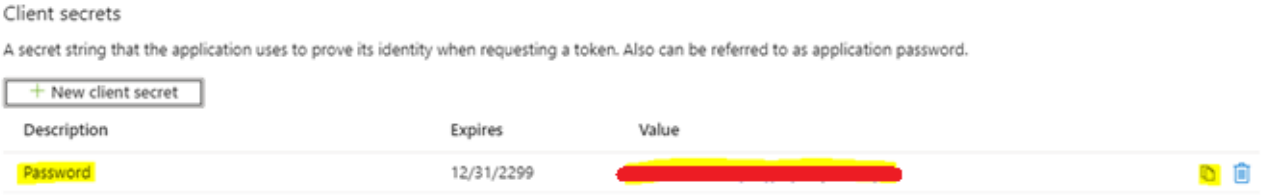


- Once the *App Registration* is created, it will redirect you to its *Overview* menu. Here you can copy the *Application ID* and the *Directory ID*.

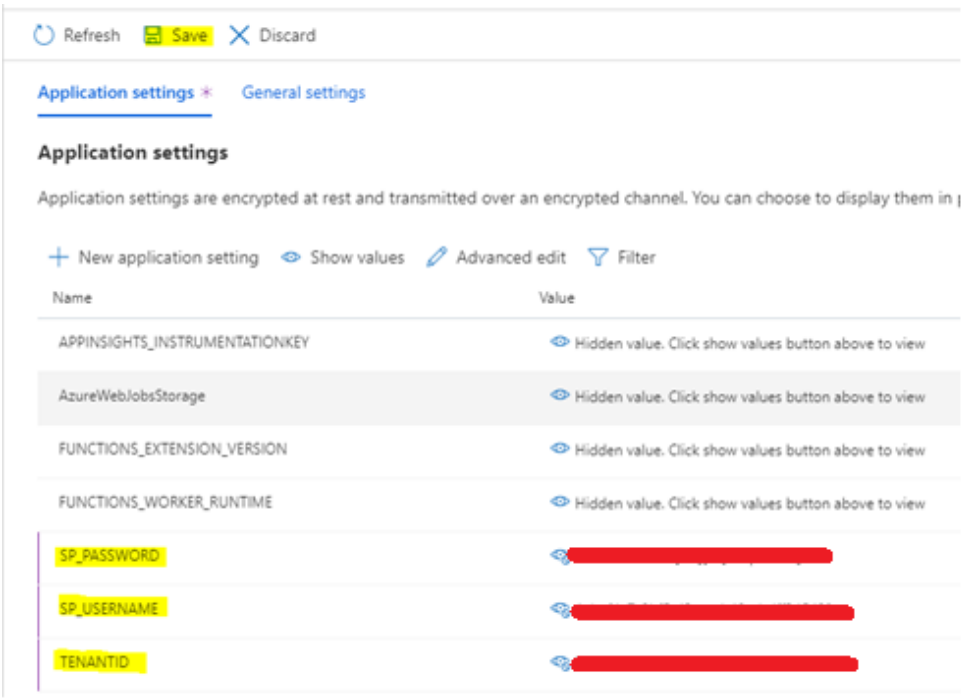
- Then, you are going to create a *secret* for this *App registration*. This is going to be the *App Registration's Password*, which is going to be used alongside the *Application ID* and *Directory ID*, while the *Function App* requires information from your *Azure subscription*.



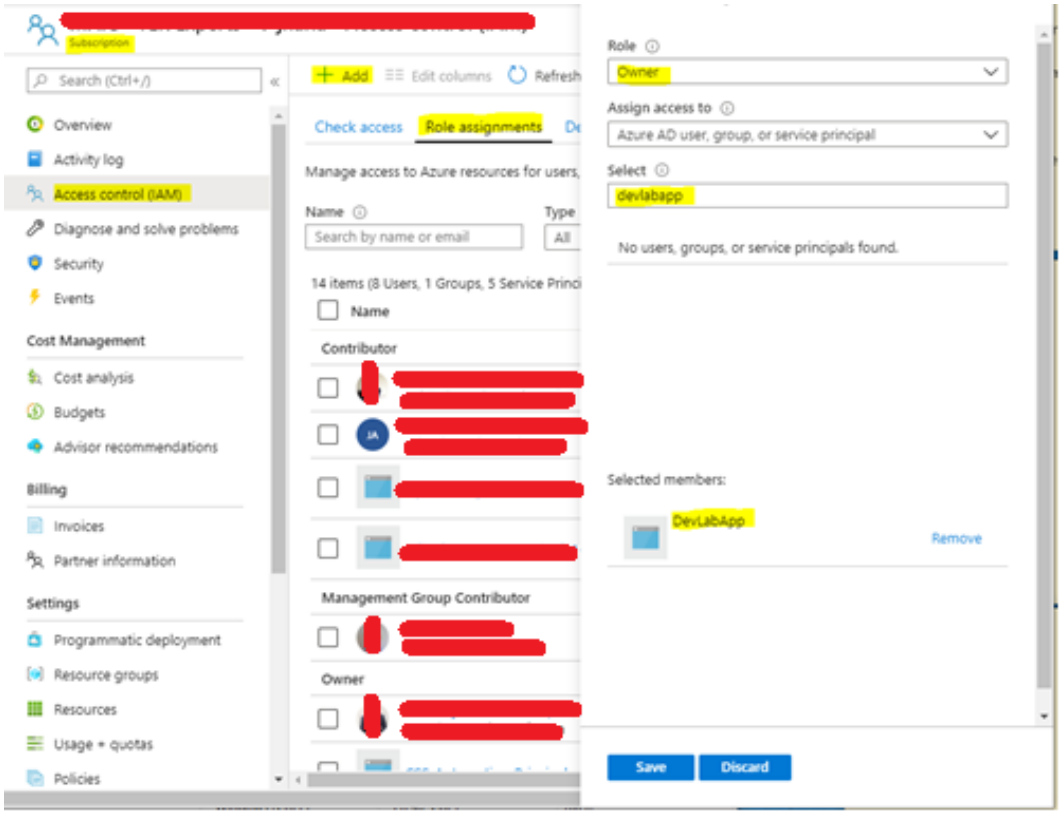




- Now, go to your *Function App's settings* and from here, *add* these three (3) *parameters* as shown below. Don't forget to *save* it.



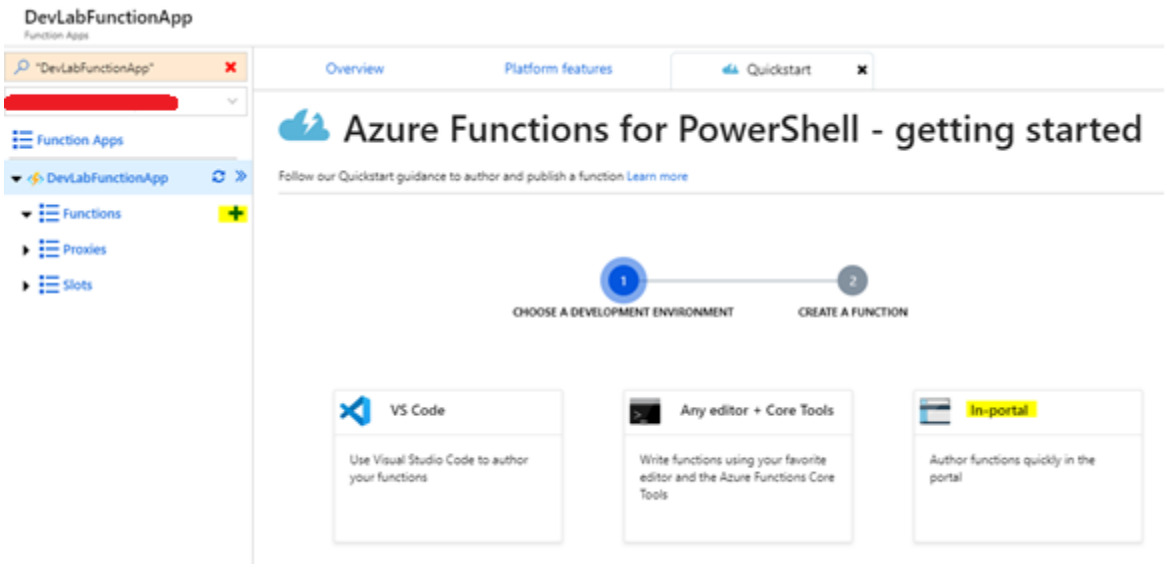
- **Important**: the *App Registration* must have permissions under the subscription. In this laboratory, we chose to *grant* an *owner role* to it.



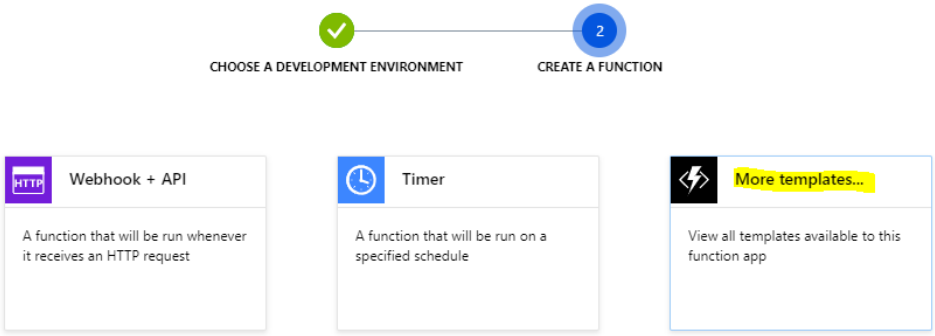
B. Creating our ON/OFF functions.

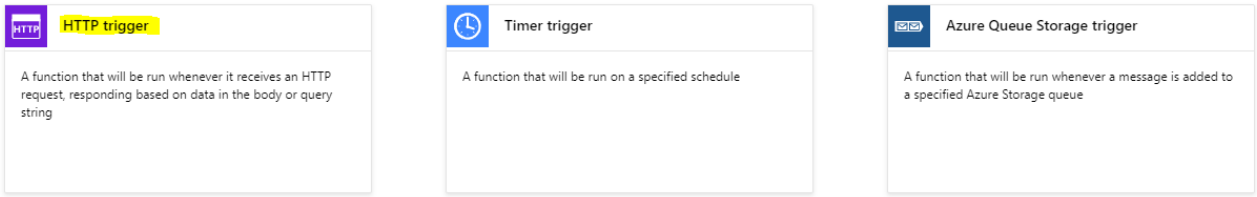
In this section, we are going to create a Start and a Stop function, using Powershell as the runtime stack and making usage of the Az Module.

- Proceed to enter the *Function App* and click on the "**+**" button, next to the "*Functions*" label. Notice that you can *develop functions* using *external tools* such as *VS Code*. For this case, we chose to use the in-portal option.

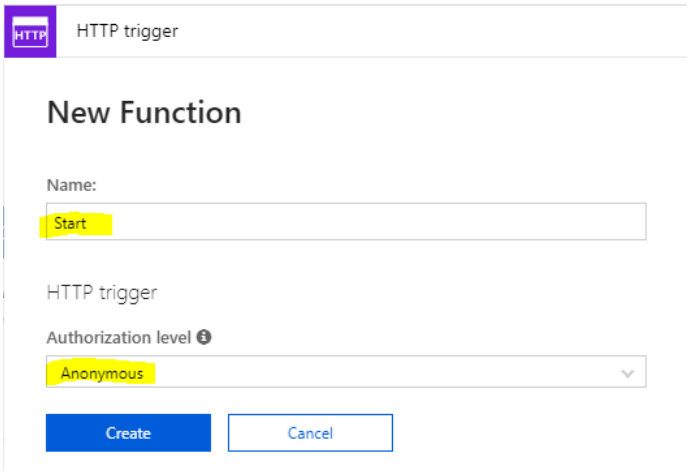


- There are several options to *trigger* a *Function*. For this specific case, the Function will be *triggered* by an *HTTP call*.





- Now, on this menu, give it a name like “Start” and set the *Authorization level* to “*Anonymous*”. What are the differences between each *Authorization level*?



- Do the same for the “Stop” function.
- Once you have both functions, get rid of the code and place the respective one for each one:

Start Function:

```
using namespace System.Net
using namespace Az.Compute

# Input bindings are passed in via param block.
param($Request, $TriggerMetadata);

# Write to the Azure Functions log stream.
Write-Host "PowerShell HTTP trigger function processed a request.";

Write-Host $Request.Body.vmname;
Write-Host $Request.Body.resourcegroup;

# Set Service Principal credentials
# SP_PASSWORD, SP_USERNAME, TENANTID are app settings
$secpasswd = ConvertTo-SecureString $env:SP_PASSWORD -AsPlainText -Force;
$mycreds = New-Object System.Management.Automation.PSCredential
($env:SP_USERNAME, $secpasswd)

Add-AzAccount -ServicePrincipal -Tenant $env:TENANTID -Credential $mycreds;
$context = Get-AzContext;
Set-AzContext -Context $context;

try{
    # Start VM
    Start-AzVM -ResourceGroupName $Request.Body.resourcegroup -Name  $Re-
quest.Body.vmname -NoWait | Out-String;

    Push-OutputBinding -Name response -Value ([HttpResponseContext]@{
        StatusCode = [System.Net.HttpStatusCode]::OK
        Body = "{message:'Vm Has started :)'}"
    })
}catch{
    Push-OutputBinding -Name response -Value ([HttpResponseContext]@{
        StatusCode = [System.Net.HttpStatusCode]::BadRequest
        Body = "{message:'There was something wrong :('}"
    })
}
```

(If you want to copy and paste, right-click on the image-> Document Object-> Open)

Stop Function:

```
using namespace System.Net
using namespace Az.Compute

# Input bindings are passed in via param block.
param($Request, $TriggerMetadata);

# Write to the Azure Functions log stream.
Write-Host "PowerShell HTTP trigger function processed a request.";

Write-Host $Request.Body.vmname;
Write-Host $Request.Body.resourcegroup;

# Set Service Principal credentials
# SP_PASSWORD, SP_USERNAME, TENANTID are app settings
$secpasswd = ConvertTo-SecureString $env:SP_PASSWORD -AsPlainText -Force;
$mycreds = New-Object System.Management.Automation.PSCredential
($env:SP_USERNAME, $secpasswd)

Add-AzAccount -ServicePrincipal -Tenant $env:TENANTID -Credential $mycreds;
$context = Get-AzContext;
Set-AzContext -Context $context;

try{
    # Stop VM
    Stop-AzVM -ResourceGroupName $Request.Body.resourcegroup -Name  $Re-
quest.Body.vmname -Force | Out-String

    Push-OutputBinding -Name response -Value ([HttpResponseContext]@{
        StatusCode = [System.Net.HttpStatusCode]::OK
        Body = "{message:'Vm Has stopped :)'}"
    })
}catch{
    Push-OutputBinding -Name response -Value ([HttpResponseContext]@{
        StatusCode = [System.Net.HttpStatusCode]::BadRequest
        Body = "{message:'There was something wrong :('}"
    })
}
```

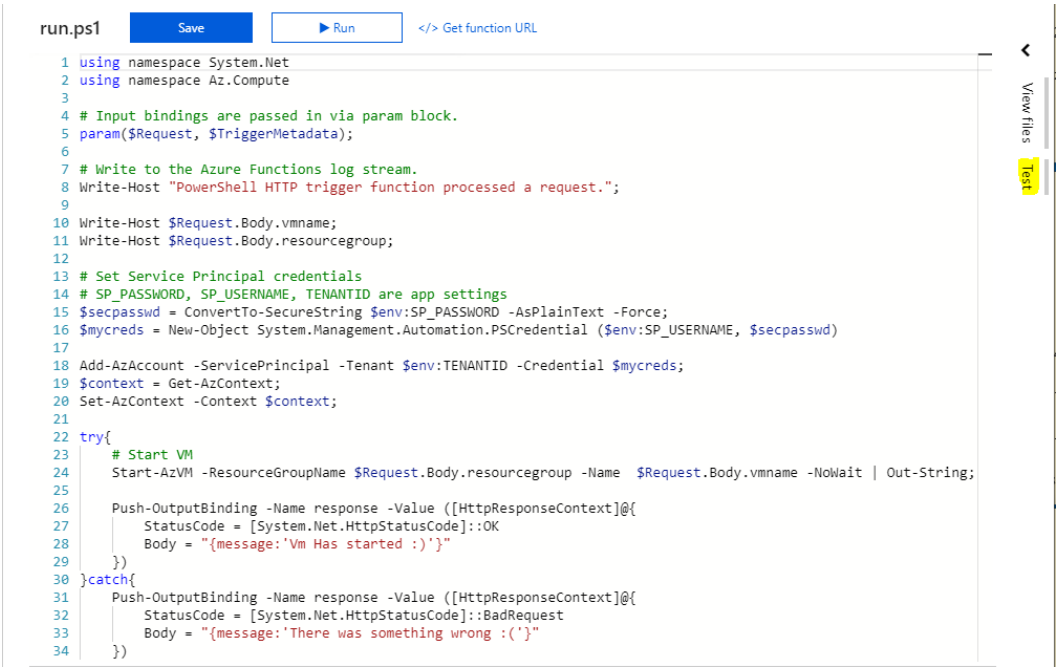
(If you want to copy and paste, right-click on the image-> Document Object-> Open)

Take your time to analyze the code. In summary, the first *block imports* the required *libraries*. Then the code gets the *trigger's parameters* (the *request* has the required information to turn on/off a *VM*), then the code prints data to allow a user of keeping track of the request's parameters. In the next *block*, the *function gets* the *credentials* and looks for the *subscription context*, this information is required to use the Start(Stop)-AzVM command.

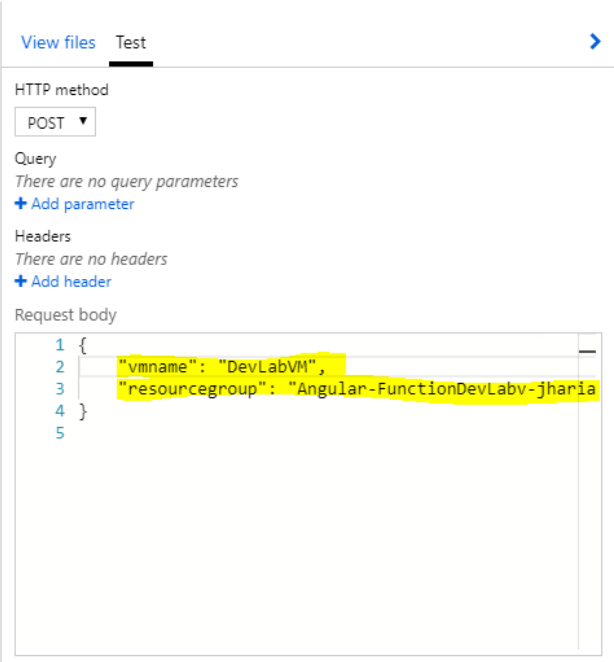
C. Testing the functions.

Now that you have the two (2) *functions*, you can proceed to *test* them from the *portal*.

- On the right side of each *function*, you can see a "*Test*" blade.

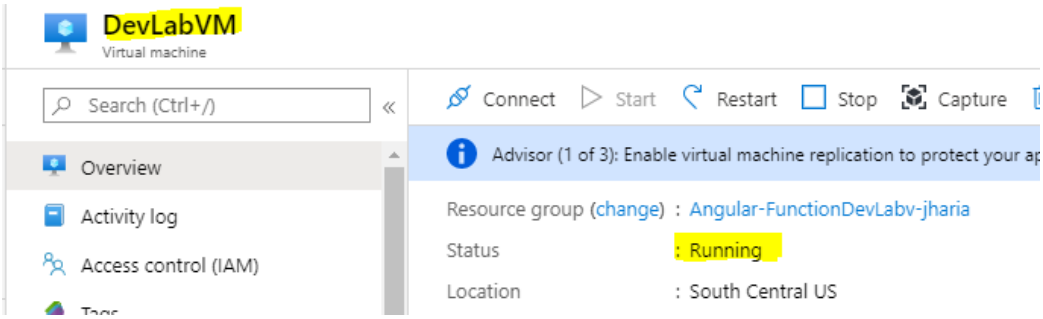


- Note that these *functions* respond to *HTTP GET* and *POST* calls. Where can you find and edit this configuration to only manage *POST* calls?
- On the *Test* blade, you are going to make a *POST* call to the *function* passing a *JSON* with the information of an *Azure VM*.



- Then, click on the *Run button* to make a call to the *function*. And then, check the result at the *Output console*.

- You can also check the VM status, to confirm if it started.



- Do the same test for both *functions*.

Notice that a *Function App* might respond slow on their first calls. If you are getting *500 HTTP errors*, consider *debugging* the *function* with the *console logs* and *Application Insights*. Most of the time, it could be something with the *role permissions* or a miss-configuration.

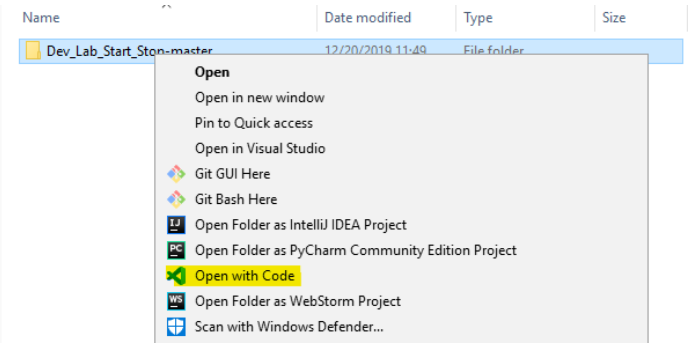
Section 3: Angular application Development.

In this section, you are going to *clone* an *Angular project* and set it up *locally* to turn ON/OFF an *Azure VM*.

Important: You must have *NodeJS* installed *locally* on your computer to *test* the application. If you don't have it installed yet, go to https://nodejs.org/en/download/.

Now, *clone* or download the following Angular Project: https://github.com/JhosuaArias/Dev_Lab_Start_Stop. If you don't have access to it, ask the trainer for a zipped version of the *project*.

Once you have an *unzipped version* of the *project*. Proceed to open *Visual Studio Code* or any other *text editor* of your preference and open the *Angular project* on it.



From here, open a new *Terminal*, in order to do this on Visual Studio, you can press **Ctrl** + **Shift** + ` or you can find the *Terminal Section* at the top of the window.

Now, you require to install *Angular* in the computer, so you use the *module's commands* and *libraries*. For this, use the following command in the *Terminal*:

- **npm install -g @angular/cli**

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Jhosua\Desktop\Dev_Lab_Start_Stop-master\Dev_Lab_Start_Stop-master> npm install -g @angular/cli
C:\Users\Jhosua\AppData\Roaming\npm\ng -> C:\Users\Jhosua\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng

> @angular/cli@8.3.21 postinstall C:\Users\Jhosua\AppData\Roaming\npm\node_modules\@angular\cli
> node ./bin/postinstall/script.js

+ @angular/cli@8.3.21
added 13 packages from 6 contributors, removed 1 package and updated 67 packages in 57.781s
PS C:\Users\Jhosua\Desktop\Dev_Lab_Start_Stop-master\Dev_Lab_Start_Stop-master>
```

After that, install the required *Angular modules* for this specific *project*. You can use the command below:

- **npm install**

```
PS C:\Users\Jhosua\Desktop\Dev_Lab_Start_Stop-master\Dev_Lab_Start_Stop-master> npm install

> core-js@2.6.10 postinstall C:\Users\Jhosua\Desktop\Dev_Lab_Start_Stop-master\Dev_Lab_Start_Stop-master\node_modules\babel-runtime\node_modules\core-js
> node postinstall || echo "ignore"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)


> core-js@3.1.4 postinstall C:\Users\Jhosua\Desktop\Dev_Lab_Start_Stop-master\Dev_Lab_Start_Stop-master\node_modules\core-js
> node scripts/postinstall || echo "ignore"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)


> core-js@2.6.10 postinstall C:\Users\Jhosua\Desktop\Dev_Lab_Start_Stop-master\Dev_Lab_Start_Stop-master\node_modules\karma\node_modules\core-js
> node postinstall || echo "ignore"


> @angular/cli@8.1.3 postinstall C:\Users\Jhosua\Desktop\Dev_Lab_Start_Stop-master\Dev_Lab_Start_Stop-master\node_modules\@angular\cli
> node ./bin/postinstall/script.js

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\webpack-dev-server\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\watchpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\karma\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\@angular\compiler-cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

added 1082 packages from 1045 contributors and audited 17199 packages in 219.834s

7 packages are looking for funding
  run `npm fund` for details

found 3 moderate severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details
```

(Since this is not a real on-production project, you don't have to worry about the warnings)

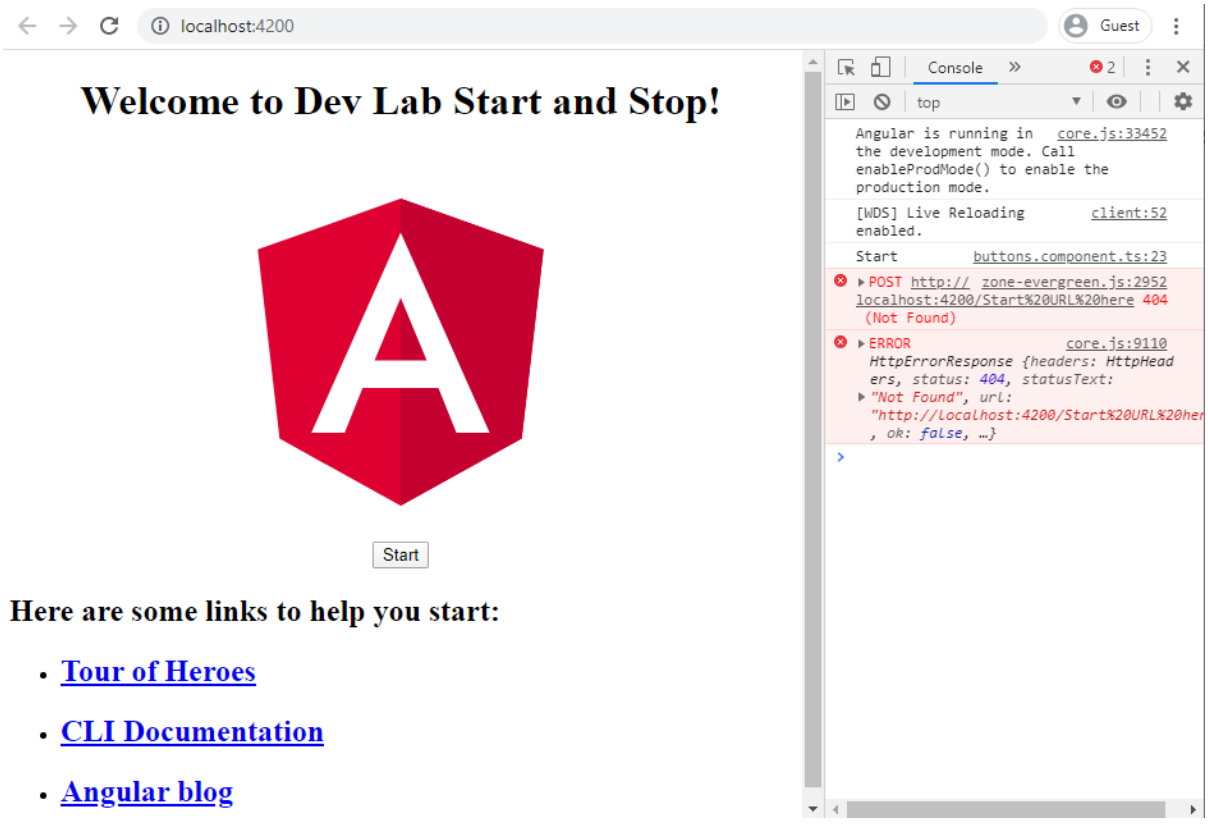
At this point, you have all the required *modules* to run the *project locally*. You can *start* a *local server* to *run* the *application* by using:

- **ng serve**

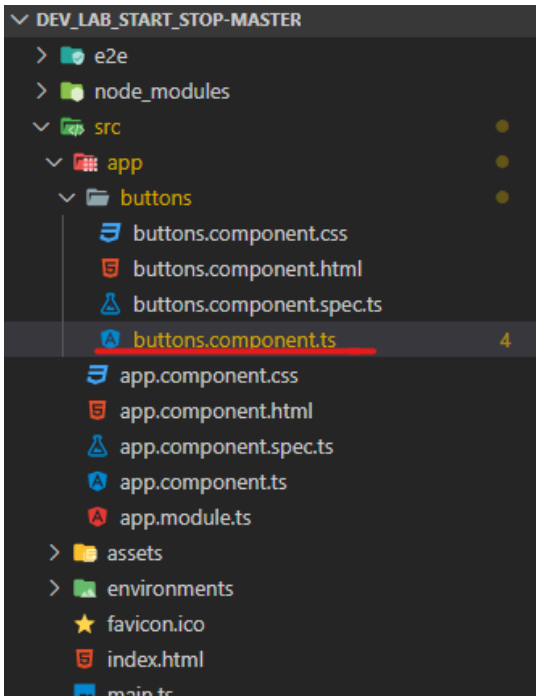
```
PS C:\Users\Jhosua\Desktop\Dev_Lab_Start_Stop-master\Dev_Lab_Start_Stop-master> ng serve
Your global Angular CLI version (8.3.21) is greater than your local
version (8.1.3). The local Angular CLI version is used.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".
Browserslist: caniuse-lite is outdated. Please run next command `npm update`
 10% building 3/3 modules 0 activei ｢wds｣: Project is running at http://localhost:4200/webpack-dev-server/
i ｢wds｣: webpack output is served from /
i ｢wds｣: 404s will fallback to //index.html

chunk {main} main.js, main.js.map (main) 13.8 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 251 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.09 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.3 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.9 MB [initial] [rendered]
Date: 2019-12-21T18:10:49.186Z - Hash: 728525a9dc4c44a44516 - Time: 28581ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
i ｢wdm｣: Compiled successfully.
```

The application is going to be running on http://localhost:4200/. If you open that URL it will look like the following:

Since you haven't changed the *code* yet, the *Start button* is going to *return* an *error* if clicked. (Want to see the error? Press F12 in case of using Edge or Chrome, and then select the *Console* tab).



This is because the *application* is making a *POST call* to a non-existing *URL*. In order to make the *Start* button to turn on your *VM*, open the *project* and look for a *file* called "**buttons.component.ts**".



As you can see, *Angular* manages a *website section* as a *component*. You can stand a *component* like a *pre-made* or "little" section of a *webpage* (**Such a header, footer, aside, etc.**) which has its own *HTML structure*, *TypeScript/Javascript logic* and *styles*. *Components* are really useful because you can put one inside of another, just by using a single *custom HTML tag*. In this case, there is a single *component* to isolate the button's *logic*, *HTML* and *styles*. Feel free to take a look at the *component's* three (3) main files (**buttons.component.css, buttons.component.html and buttons.component.ts**).

The file called "**buttons.component.ts**" contains the code to be executed on the *client's side*. It is going to look like this:

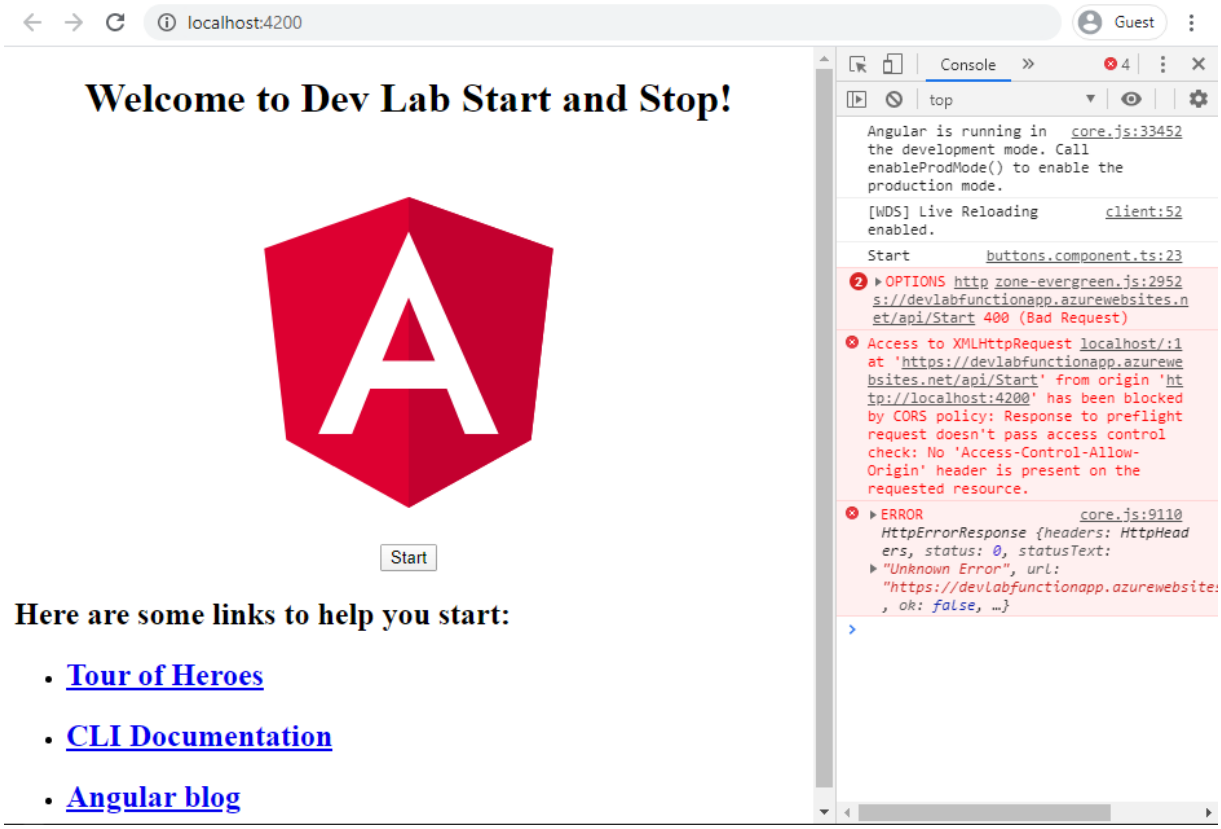
```
buttons.component.ts
src > app > buttons > buttons.component.ts > ButtonsComponent
1   import { Component, OnInit } from '@angular/core';
2   import { HttpClient } from '@angular/common/http';
3   @Component({
4     selector: 'app-buttons',
5     templateUrl: './buttons.component.html',
6     styleUrls: ['./buttons.component.css']
7   })
8   export class ButtonsComponent implements OnInit {
9
10    /**Please, fill with the function URL**/
11    private startUrl = 'Start URL here';
12
13    /**Please, fill with the VM's information**/
14    private VMInfo = {vmname: 'VM name', resourcegroup: 'VM Resource group'};
15
16    constructor(private http: HttpClient) { }
17
18    ngOnInit() {
19    }
20
21    startVM() {
22      this.http.post(this.startUrl, this.VMInfo).subscribe(something => console.log(something));
23      console.log("Start");
24    }
25
26    /**Here you can create a stop function**/
27  }
```

Notice that there is a *private variable* called **startURL**, here you can put the *Function's App URL* for the *Start VM function*. Additionally, there is another *private variable* called **VMInfo**, this is a *JSON* which is going to be sent on a *POST* call to turn ON/OFF the *VM*. Please, fill it with the respective information.

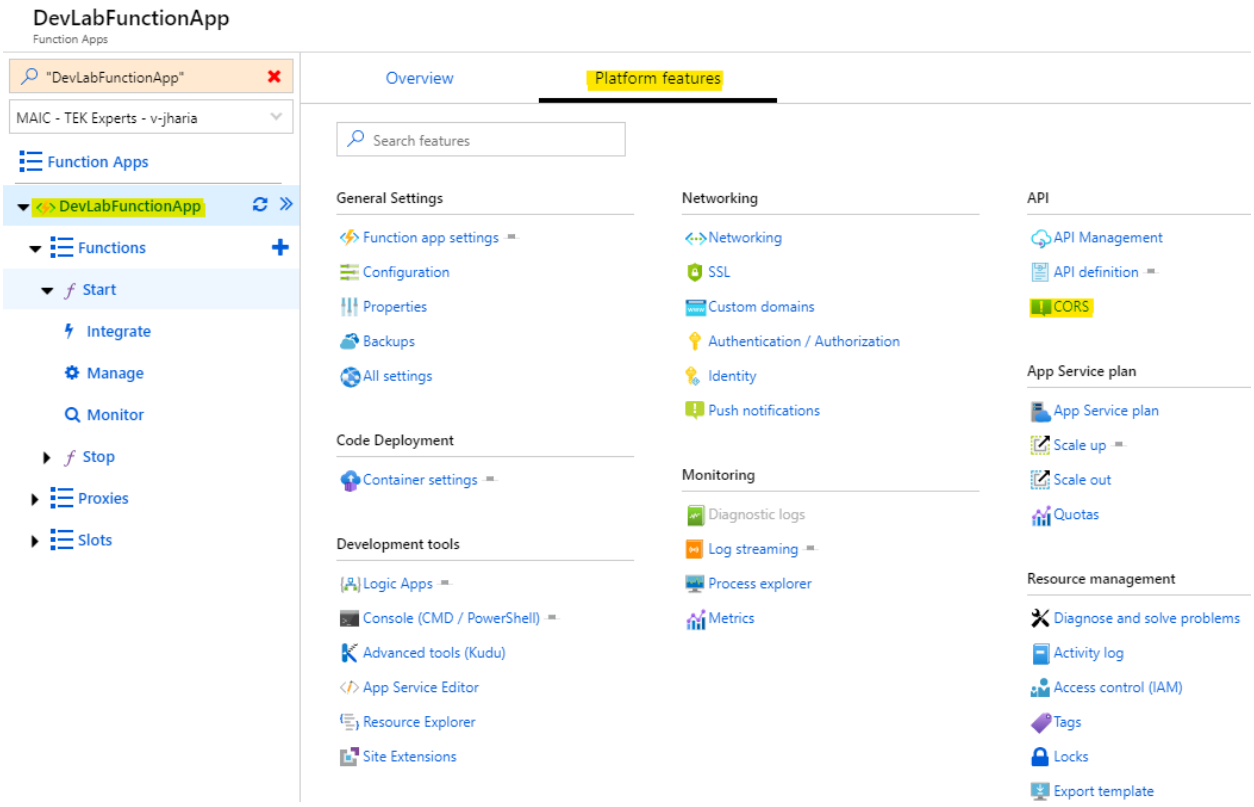
Also, there is a *function* called **startVM()**, this is going to make a *POST* call to the given **startURL**, with the *VM's* information. As an extra step, it is going to create a *listener* and *wait* for a *response* from the *server* and will print that *response*. This *function* is called while clicking the *HTML button* with the "Start" label.

```
buttons.component.html
src > app > buttons > buttons.component.html > ...
1   <div>
2     <button class="spacing" (click)="startVM()">Start</button>
3     <!-- Here you can create a  Stop button-->
4   </div>
5
```

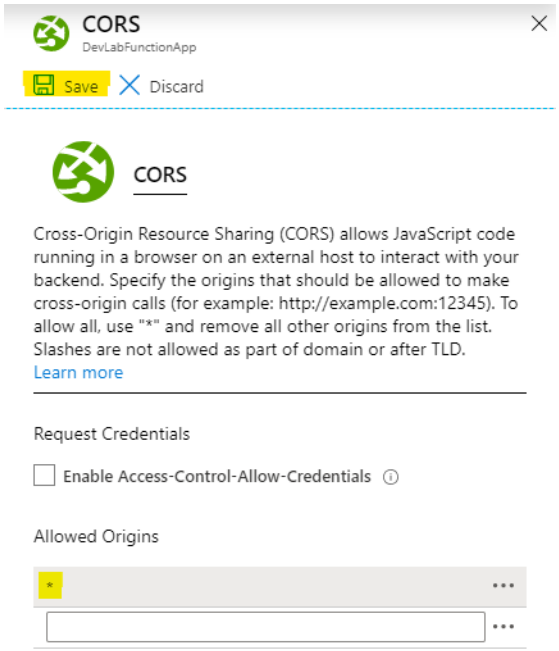
Once you have filled all the respective data, proceed to *test* the *application* to turn on the *VM*.



As you can see on the above picture, it is still giving an error. This is because, *by default*, *Function Apps* do not allow **external applications or JavaScript code** to make calls to it. In order to change this configuration, you must update the *CORS policies* on the *Function App's* configuration, which can be done at the Portal.



On the *CORS* configuration, you can find several "*Allowed Origins*". Here you must put the **origin endpoint** (the endpoint where the call is starting) for the *Function App* to accept calls from that specific *endpoint*. However, if you want to allow anyone to make calls to your *Function App* from an external application, you can set a (*).



Now, the *Angular application* should be allowed to turn ON the *VM*.

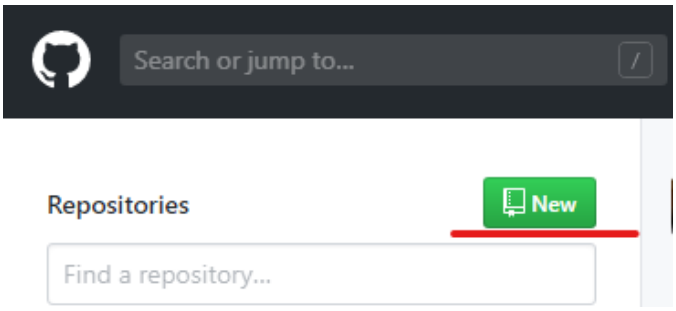
Section 4: DevOps Project configuration.

Awesome! At this point, you have an *Angular application* running locally and capable to turn ON an *Azure VM* just by clicking a button. However, it would be better if you *deploy* this project to a *Web App*, so you can have access to it from the *cloud*.

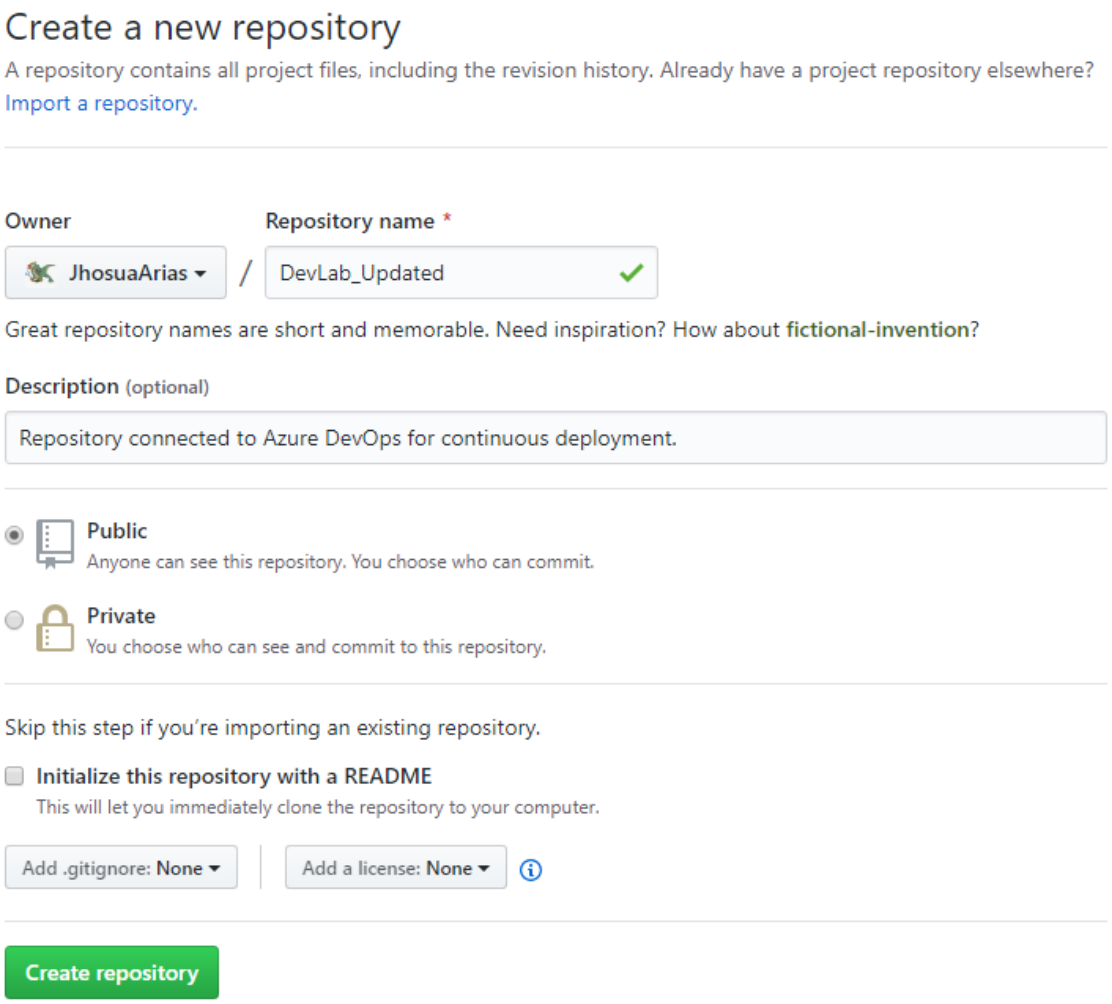
A. Publish your code in a Github Repository.

For this, it is highly suggested to *publish* the code by using *Git* commands. If you don't have *Git* installed on your computer, you can download it from https://git-scm.com/downloads.

Once it is installed, go to Github (https://github.com/), Sign in and *create* a new *repository*.

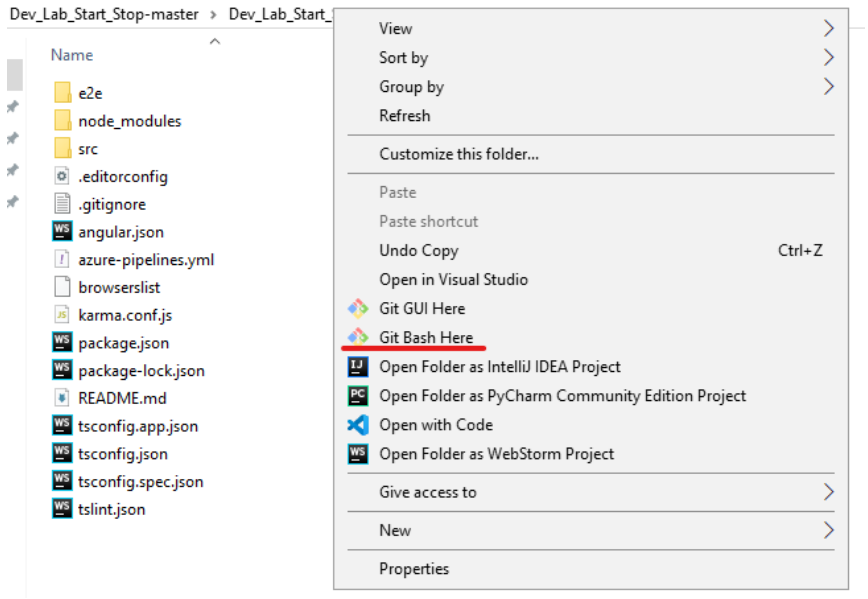


Give it a name, a description, keep it public and do not add any extra files.



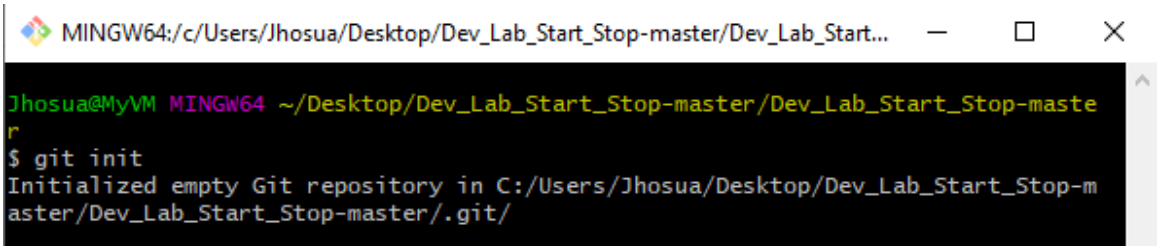
The *repository* will be completely empty at this point. Also, you can find some advice for *pushing code* to it. Keep in mind these *commands* and its descriptions, you are going to use them later.

Now, go to the project's *root* folder (where you can find the **src** folder) and right-click to open the *Git Bash*.



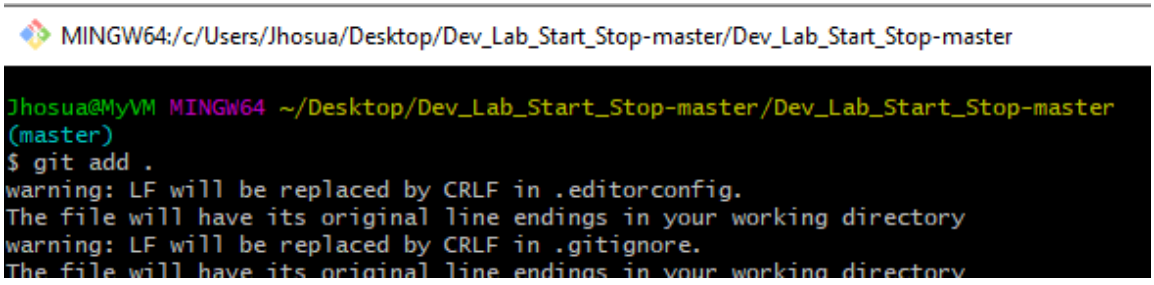
A *Git terminal* will be opened. From here, you can to use the following *command* to create a *local repository*:

- **git init**



In order to *add* all the files and sub-folders (located in the project's *root folder*) to the *local repository*, use the next *command*:

- **git add .**



Now, you can *commit* all the *added files* to the *local repository*:

- **git commit -m "<Your message here>"**

```
MINGW64:/c/Users/Jhosua/Desktop/Dev_Lab_Start_Stop-master/Dev_Lab_Start_Stop-master
Jhosua@MyVM MINGW64 ~/Desktop/Dev_Lab_Start_Stop-master/Dev_Lab_Start_Stop-master (master)
$ git commit -m "first commit"
[master (root-commit) 94c0dd9] first commit
 35 files changed, 12847 insertions(+)
 create mode 100644 .editorconfig
 create mode 100644 .gitignore
```

Before *pushing* the code to a *remote repository* (GitHub), you must add the *remote repository's reference*, with the command showed below:

- **git remote add origin <GitHub Repository's URL>**

```
MINGW64:/c/Users/Jhosua/Desktop/Dev_Lab_Start_Stop-master/Dev_Lab_Start_Stop-master
Jhosua@MyVM MINGW64 ~/Desktop/Dev_Lab_Start_Stop-master/Dev_Lab_Start_Stop-master (master)
$ git remote add origin https://github.com/JhosuaArias/DevLab_Updated.git
```

Once all these are done, you can *push* the code into the *remote repository*:

- **git push -u origin master**

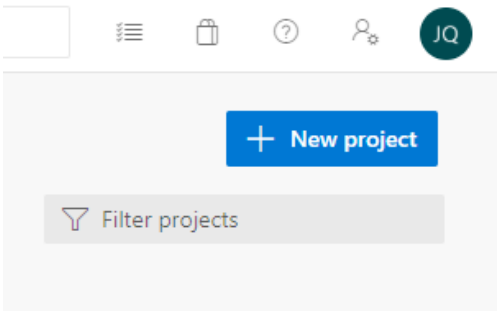
```
MINGW64:/c/Users/Jhosua/Desktop/Dev_Lab_Start_Stop-master/Dev_Lab_Start_Stop-master
Jhosua@MyVM MINGW64 ~/Desktop/Dev_Lab_Start_Stop-master/Dev_Lab_Start_Stop-master (master)
$ git push -u origin master
Enumerating objects: 43, done.
Counting objects: 100% (43/43), done.
Delta compression using up to 2 threads
Compressing objects: 100% (40/40), done.
Writing objects: 100% (43/43), 109.52 KiB | 2.67 MiB/s, done.
Total 43 (delta 0), reused 0 (delta 0)
To https://github.com/JhosuaArias/DevLab_Updated.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

After this, if you check the *GitHub's* website, you are going to find all the code.

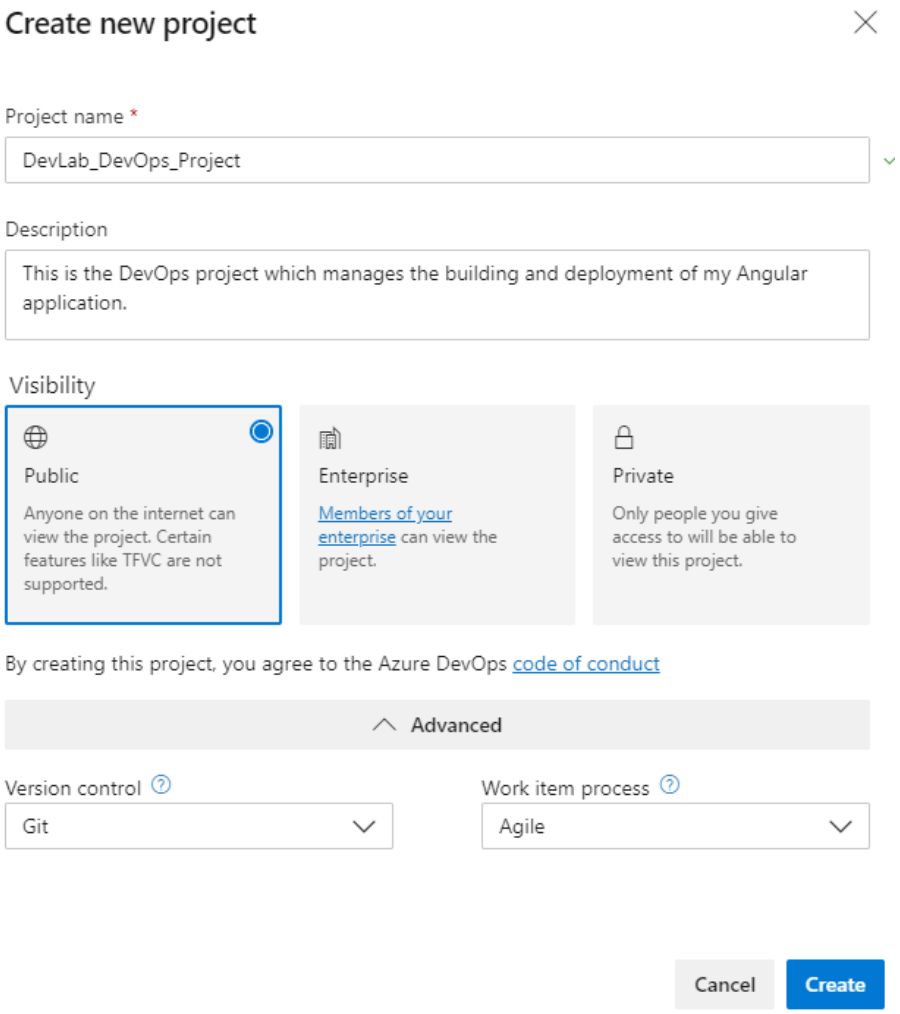
B. Create a new Project on Azure DevOps.

You are going to create an *Azure DevOps project* to manage the *build* and *release* of the *Angular application*. In order to do this, go to https://azure.microsoft.com/en-us/services/devops/ and click on the "**Start free**" button, then register/login to the application.

Once you are in the *Azure DevOps* dashboard, look at the right side, you can find a "**+ New Project**" button. Click on it.



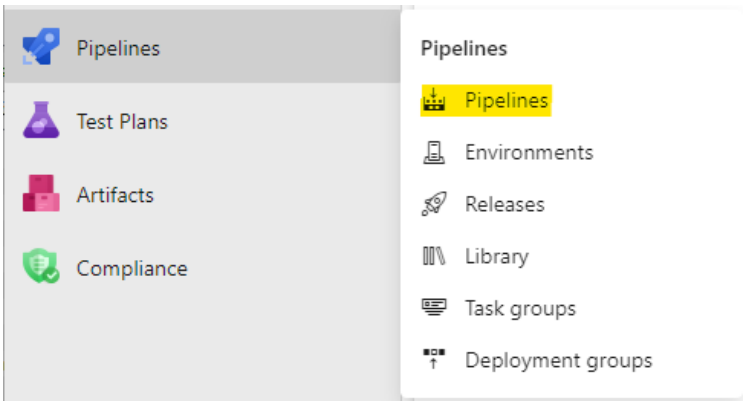
A form will be opened to create your new *project*. Give to the *project* a name, a description, set it to public and use *Git* as version control.

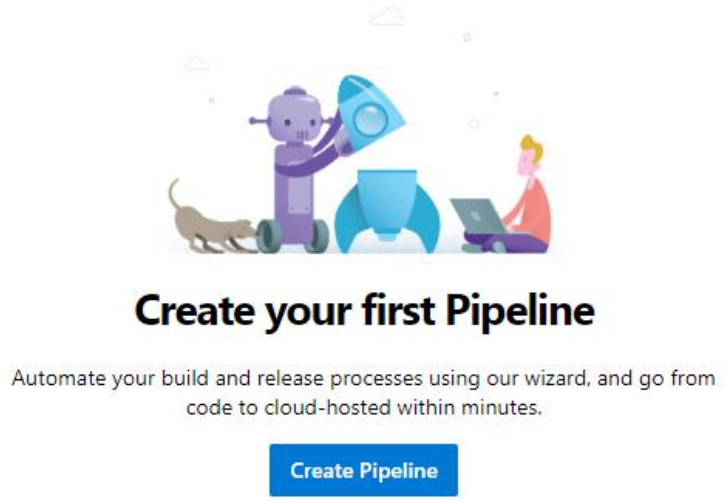


C. Create a Build Pipeline

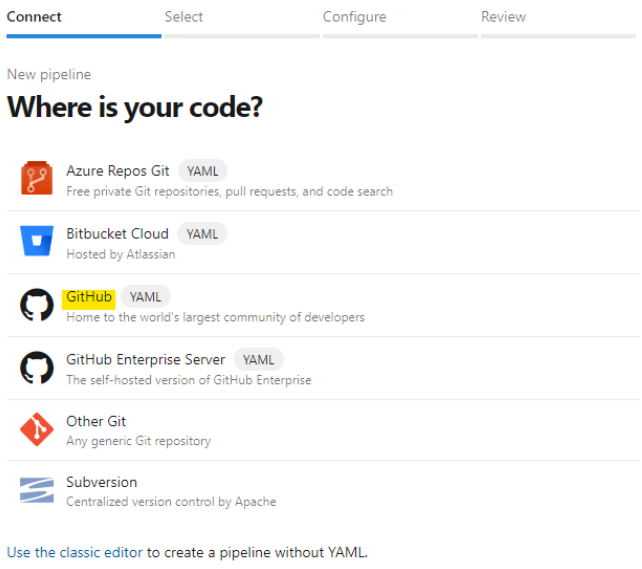
Now that you have a new *project* created, proceed to create a *pipeline* to build your project. Basically, the *pipeline* will be attentive for any *push* made into the *GitHub's repository*. If there is any new code *pushed* in the *repository*, the *pipeline* is going to *clone* it, *build* the *code binaries* and *create* an *artifact* from those *binaries*.

First, go to the *project's dashboard* and click on "**Pipelines**", which can be found on the left side. And then, click on "**Create Pipeline**".

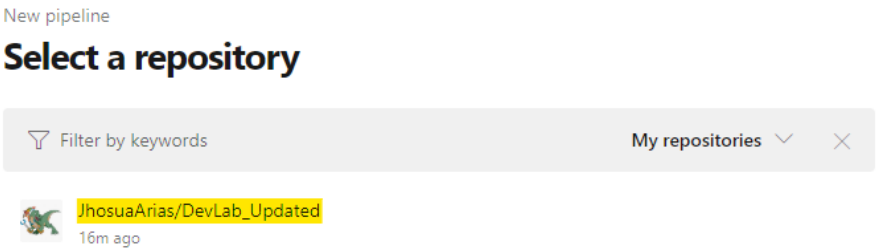




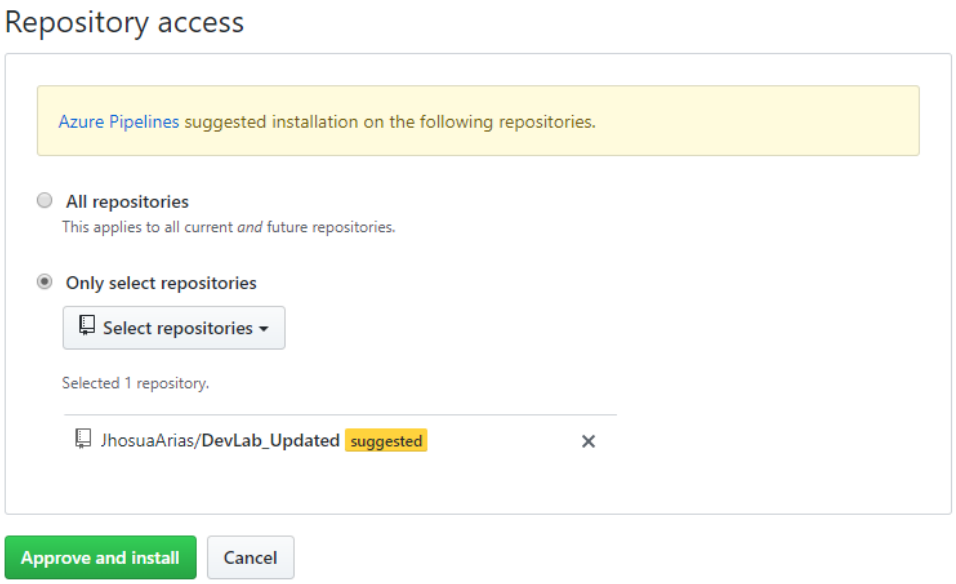
A form will be shown for configuring the *pipeline*. First, you need to set the *external repository*. Click on the *GitHub* button and login to your account.



Then, select the respective *repository*. The *pipeline* will configure a *listener* on the selected *repository*.



Azure DevOps requires permission from *GitHub* to access this repository. Proceed to approve these permissions.



Once this is done, a text editor will appear with a premade *script*. These *commands* are going to *run* after *cloning* the *GitHub's code*. After that the cloning, you want the *pipeline* to *build* the *binaries* and *create* an *artifact*. Please, refer to the following *script*:

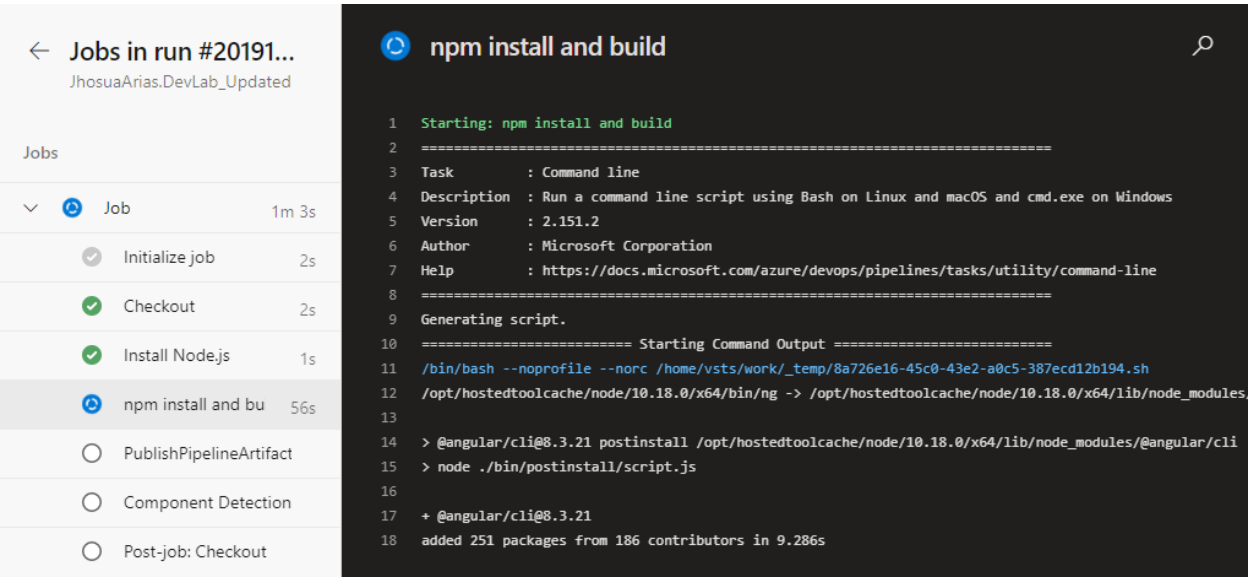
```
#
Node.js
with
Angular
          # Build a Node.js project that uses Angular.
          # Add steps that analyze code, save build artifacts, deploy, and more:
          # https://docs.microsoft.com/azure/devops/pipelines/languages/javascript

          trigger:
          - master

          pool:
            vmImage: 'ubuntu-latest'

          steps:
          - task: NodeTool@0
            inputs:
              versionSpec: '10.x'
            displayName: 'Install Node.js'

          - script: |
              npm install -g @angular/cli
              npm install
              ng build --prod
              ls -al
              pwd
            displayName: 'npm install and build'

          - task: PublishPipelineArtifact@0
            inputs:
              artifactName: 'angular'
              targetPath: '/home/vsts/work/1/s/dist'
```

(If you want to copy and paste, right-click on the image-> Document Object-> Open)

With this *script*, you are "telling" the *pipeline* to:

- *Listen* to the *master branch* of the *remote repository*.
- The *application* will *run* on *Ubuntu's latest version*.
- Install *NodeJS*.
- Install *Angular*, the *project's dependencies* and *build* the project *binaries*.
- *Create* an *artifact* and *store* it at the '**/home/vsts/work/1/s/dist**' path.

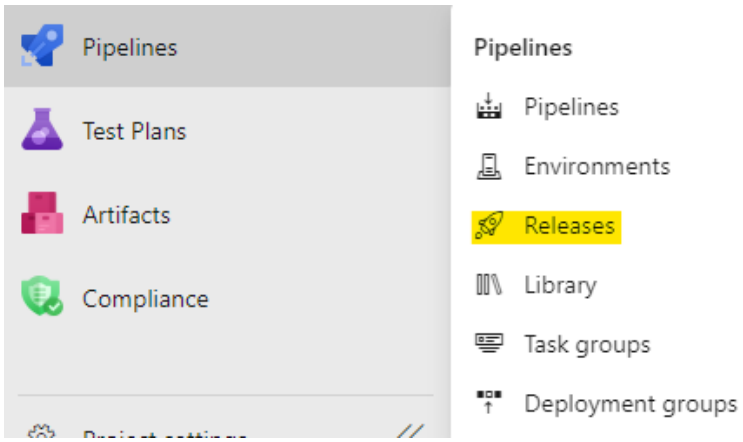
Once this is completed, click on "**Run**". You are going to be able to see the *Job* running to *process* the above *script*.



D. Create a Pipeline to Release.

In the previous step, you have created a *pipeline* to *get* any new code from a *remote repository*, *build* its *binaries* and *create* an *artifact*. Nonetheless, the *artifact* is just stored on *Azure DevOps*. Due to this, you need to *create* another *pipeline* to *release* the *artifact* into a **Web App**.

Create this *pipeline* the same way as the previous one, but making sure to select "**Release**".

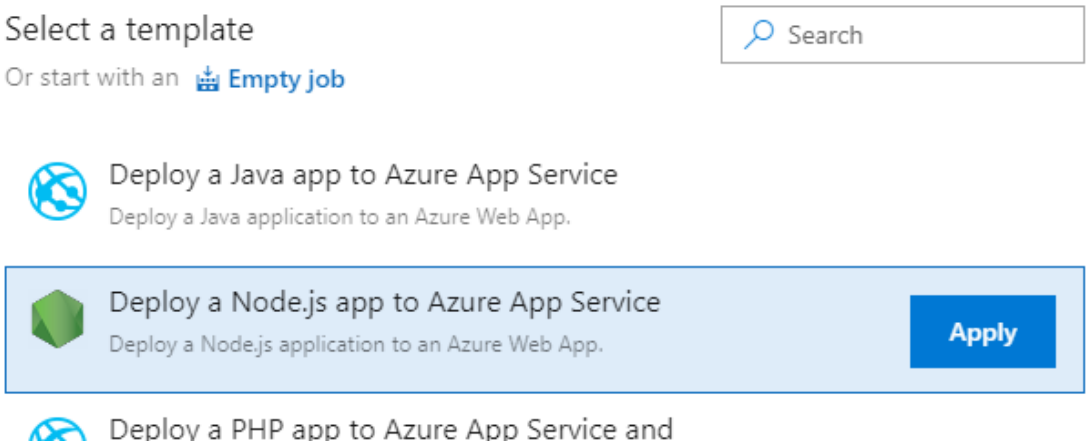


No release pipelines found

Automate your release process in a few easy steps with a new pipeline

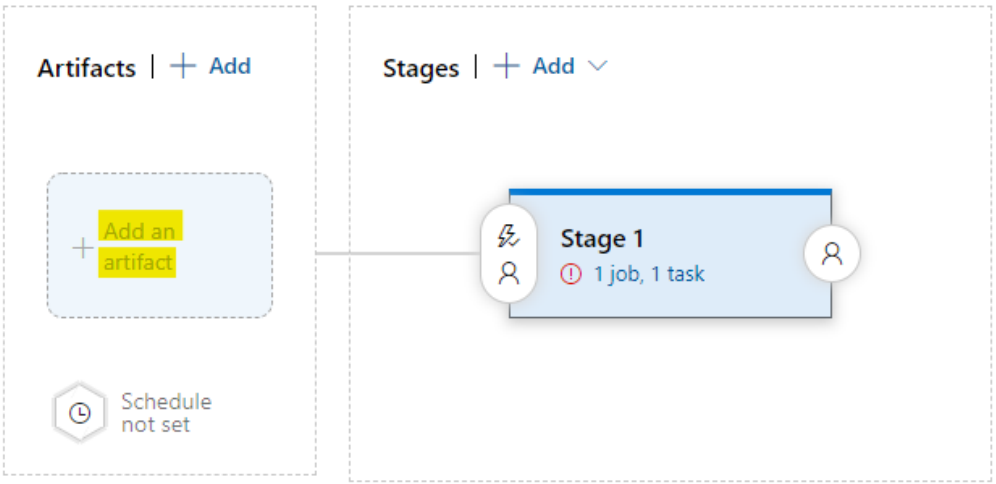
New pipeline

Then, select "**Deploy a Node.JS app to Azure App Service**" template and click on apply.

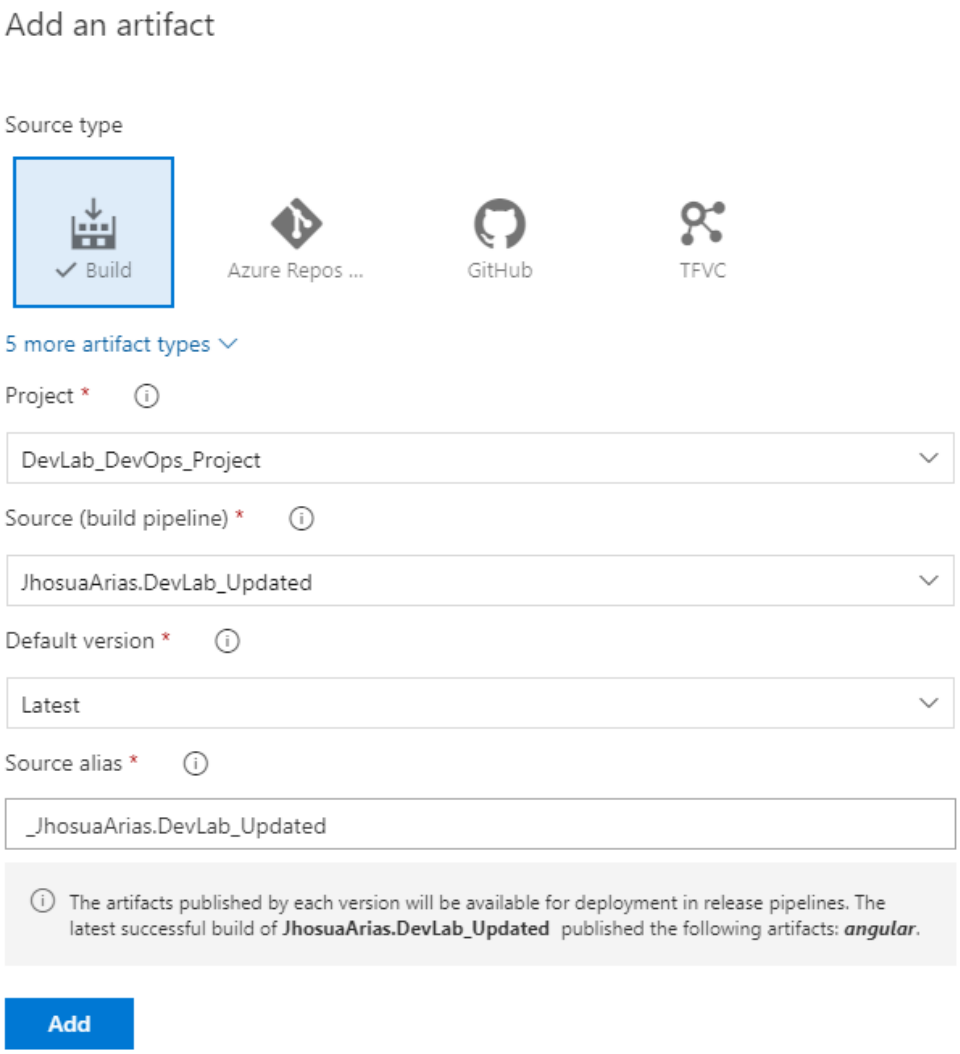


This is going to create a *task* on Stage 1 for the *release pipeline*, which is meant to pick the *binaries* from the already created *artifac*t and *deploy* it to an *Azure Web App*.

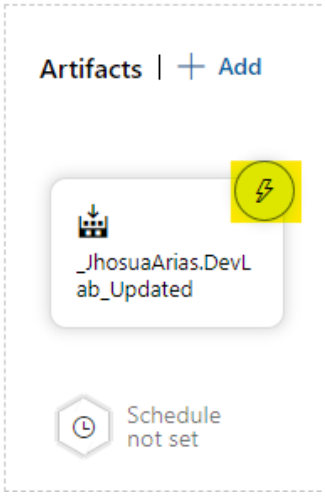
Now, click on the **"+ Add an artifact**" button.

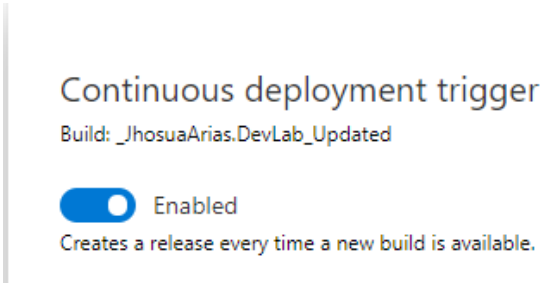


A menu is going to show up, on there select the respective "**Source (build pipeline)**". This source is the same *pipeline* that we made in the previous step.

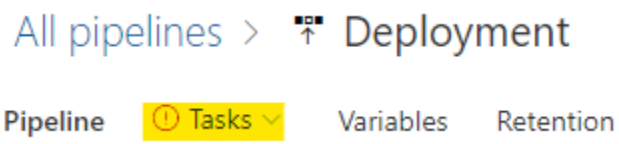


After that, click on the thunder icon and set the *continuous deployment* configuration. This is going to *release* your *application* each time there is a new *build* available.



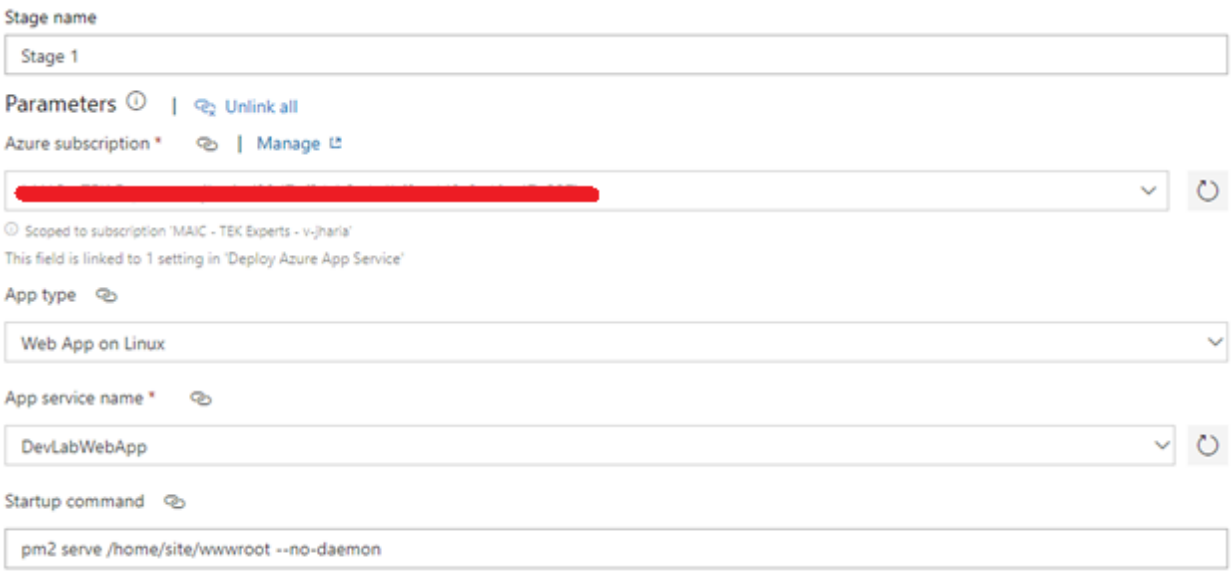


Now, you must configure the tasks. In order to do this, click on the “Tasks” tab.

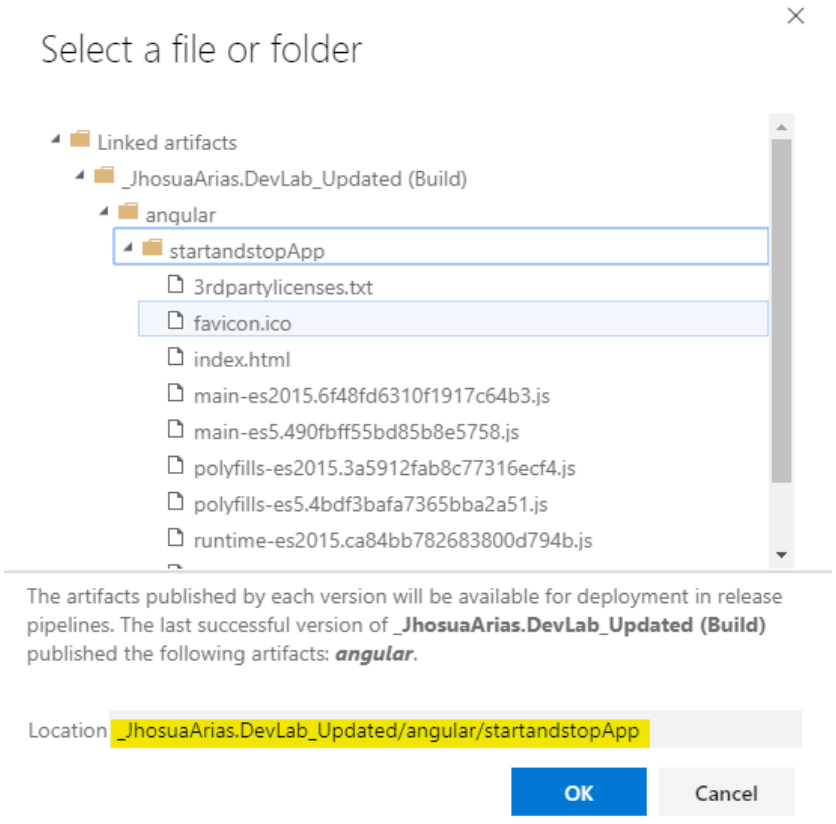


Then select and configure:

- An Azure Subscription.
- Web App on Linux as a Web App Type.
- The App Service name.
- And this startup command: pm2 serve /home/site/wwwroot --no-daemon. (This is really important to start a server to run the NodeJS application).

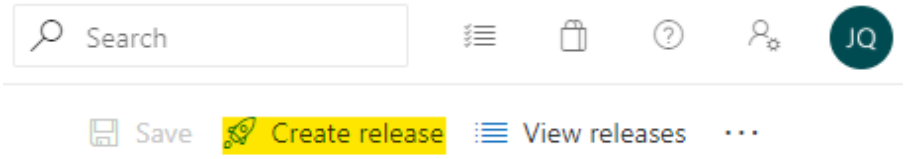


Now click on the "**Deploy Azure App Service task**" and change the *package* or *folder* field to the folder where the project is.



When you are about to save this configuration, select the root folder (/).

Click on “**Create release**” and then “**Create**” and it should *deploy* the *application* to the new *Azure Web App*.



Check if the Web App is running your application.

E. Testing all the configuration.

At this point, if you make any changes to the code, *commit* and then *push* it to *GitHub*, *Azure DevOps* will automatically *build* the *project* and *deploy* it to the *Web App*.

Please, complete this laboratory by adding the “**Stop**” button and its logic to turn OFF the *Azure VM*. Once you have finished it *locally*, proceed to *deploy* the code to the *Linux Web App*.

**** Bonus: Create a function to get the current status of the VM and its IP address. So, you can share that information in that Angular application****

Thank you. 😊