Web Development: Module 2, Lesson 3  
Deploy Node Script to Azure Hands-On Lab

## Overview

Building on [Module 2 Lesson 2](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module2/Lessons), it's time to make the application public. This application will be a huge success. For this reason, we need a scalable solution which start small and cheap but rev up as the traffic increases. Which leads to deploying to the cloud.

## Objectives

In this hands-on lab you will learn how to:

* Log in to Azure Portal, and create and manage App Services there
* Deploy a Hello World HTML page using OneDrive
* Prepare your Node CLI script for deployment
* Deploy your project to cloud using OneDrive

## Prerequisites

The following are required to complete this hands-on lab:

* A code editor
* Windows PowerShell, Mac Terminal, or some other shell with node.js installed
* You should have completed [Module 2 Lesson 1, 2 and 3](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module2/Lessons) as well as the [corresponding labs](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module2/Labs)
* Azure account. [See Module 1 Lesson 1](https://github.com/MSFTImagine/computerscience/blob/master/Complimentary%20Course%20Content/Module1/Labs/) for information on getting an Azure account.
* OneDrive (or Dropbox) account

## Exercises

This hands-on lab includes the following exercises:

* Exercise 1: Deploying a Hello World HTML page to Microsoft Azure App Service via OneDrive and Web Portal
* Exercise 2: Deploying a Node.js script to Microsoft Azure via web interface (Portal)

## Exercise 1: Deploying a Hello World HTML page to Microsoft Azure App Service via OneDrive and web Portal

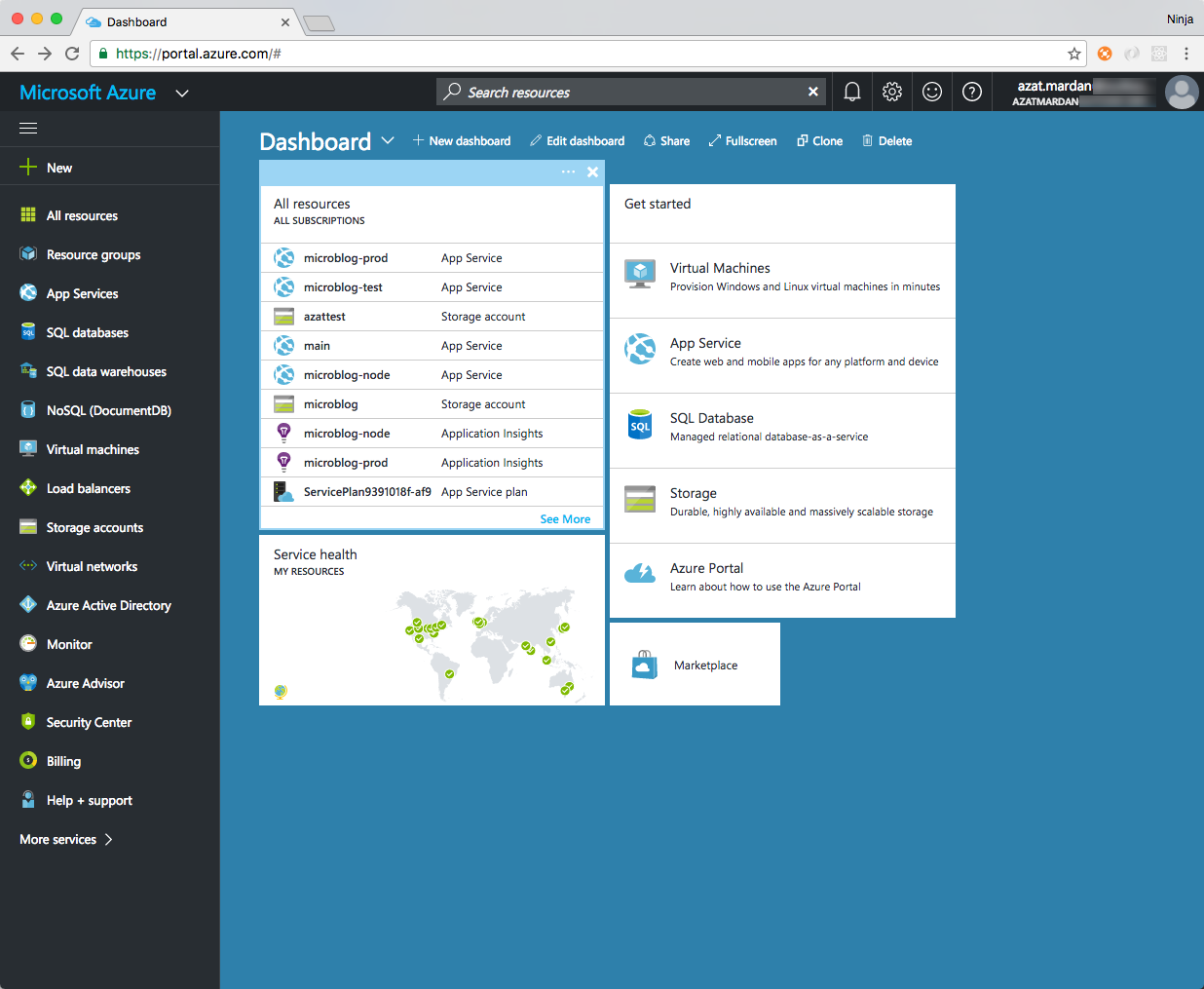
In this exercise, you will use the Microsoft Azure portal to create and configure an App Service instance, and deploy the code via Microsoft OneDrive. In addition to, you will learn how to prepare your code for Azure and how to inspect, modify and monitor the app in the cloud using the Azure portal and web tools.

These are the high-level steps to complete this exercise:

1. Open Azure portal and create App Service
2. Configure OneDrive as a Deployment option (must have OneDrive account)
3. Copy files into your folder in [OneDrive](https://onedrive.live.com/) using location Apps -> Azure Web Apps -> name of your app (e.g., microblog-prod-2).
4. Go to Azure portal, to your app, to Deployment options and ensure that the code has been synchronized. Click on Sync if needed.
5. Open the application at https://microblog-prod-2.azurewebsites.net and test it by verifying that you can see the HTML.

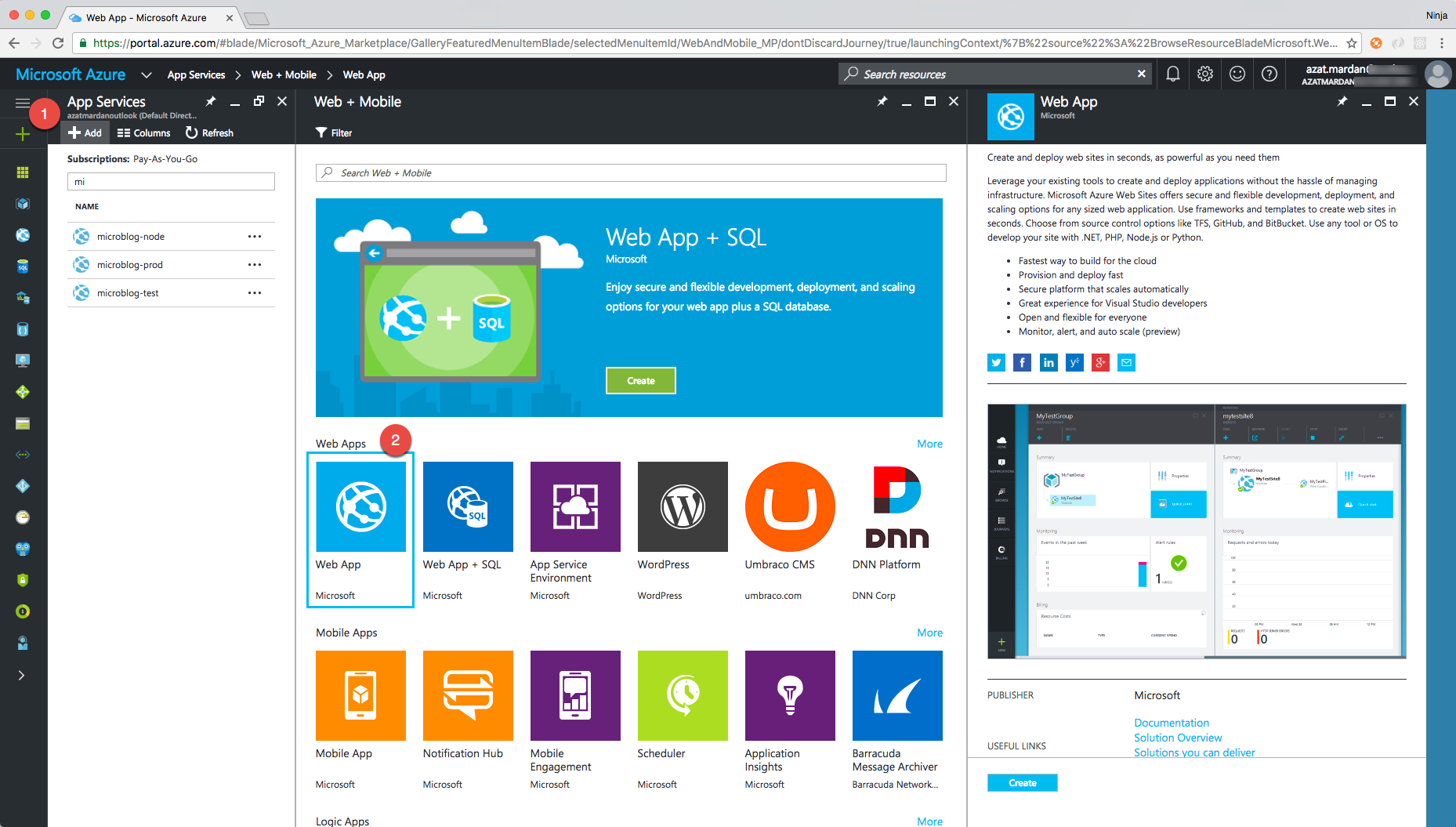
Now we will go over these steps in more details.

Open [Azure portal and sign in with your Azure subscription](https://portal.azure.com/). You can sign up for a free trial [here](https://azure.microsoft.com/en-us/free/). This is what your dashboard will look like with no apps:

****

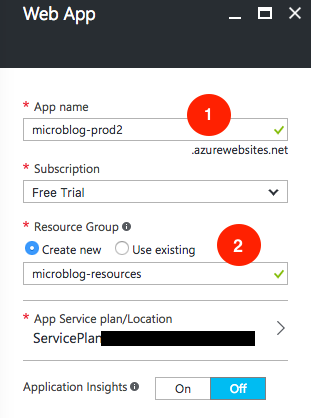
Create your App Service.

1. Select the menu option **App Services** in the left side bar. This is our PaaS which we use to deploy an HTML page. When you select App Services from the menu, you see a list of app services. If you just signed up for a new Azure subscription, this list will be empty.
2. Create a new service by selecting **+Add** from the menu and then selecting **Web App** as shown in the screenshot below .

****

When creating a new app you must specify an App name and create a resource group.

1. The app name must be unique across Azure. The name you enter for your app will determine the URL used to access your site. e.g an App name of amazingsite will be deployed to amazingsite.azurewebsites.net. A green checkmark indicates a valid app name, a red x indicates an invalid app name.
2. The resource group groups together the resources used to create your app service. Select Create New and enter a name for your resource group.

****

Creating a simple HTML web page

Create an index.html file with some basic hypertext.

<html>

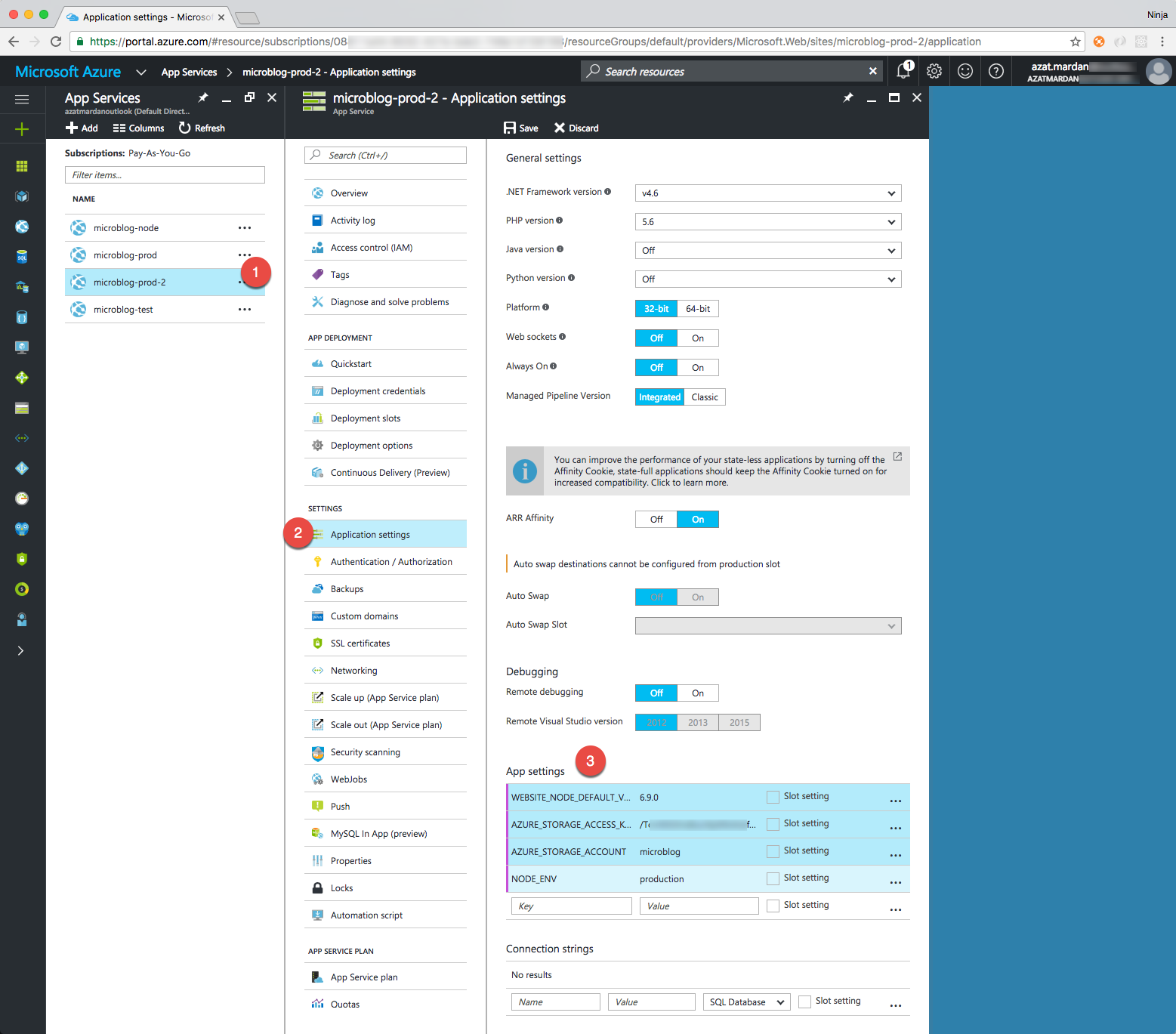
<body>

Hello from San Francisco!

</body>

</html>

Save the file on your local machine. We'll need it in a moment. For the next step you will need an account with OneDrive, you could also use Dropbox. If you have an outlook.com or live.com email, then you have OneDrive . Access OneDrive at https://onedrive.live.com. You should see an interface similar to the one shown below display your folders and files.

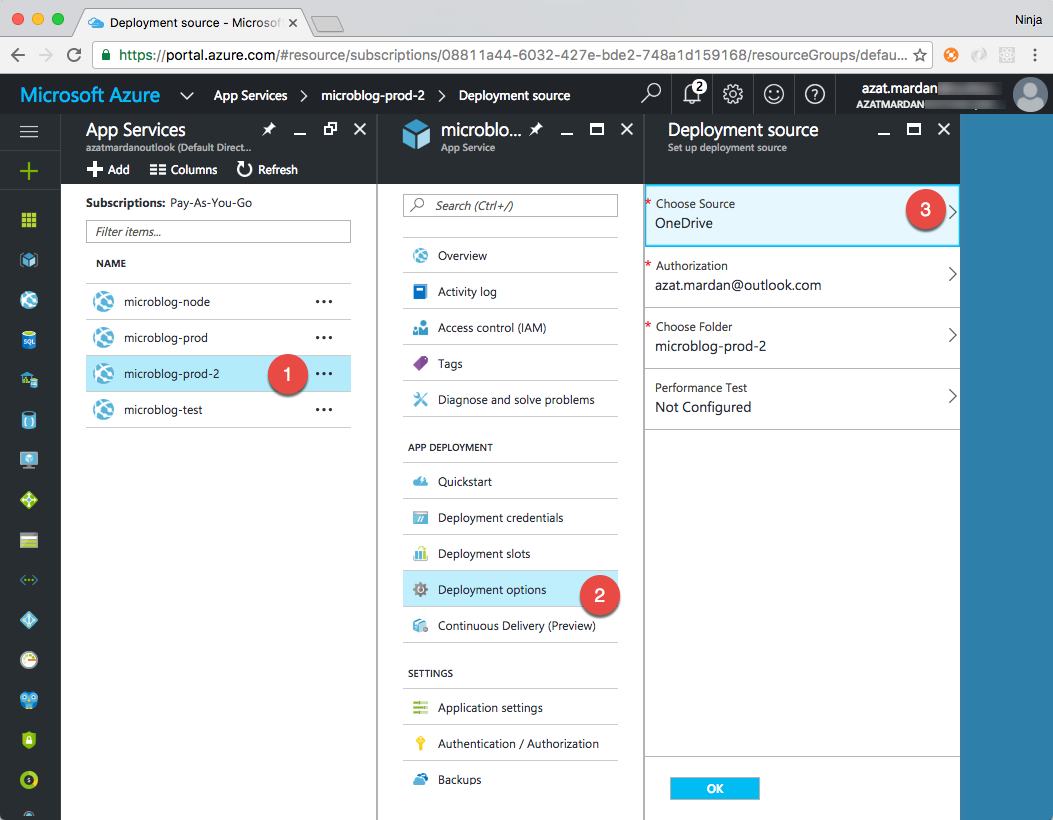


OneDrive provides file storage in the cloud. It provides and easy option for deploying to Azure App Services.

Configure OneDrive as a Deployment option

In the Azure portal, select **Deployment options** for your App as shown in the screenshot below

* **Choose Source** select **OneDrive** (if you are using a DropBox account you would select DropBox)
* **Authorization** enter your OneDrive account name
* **Choose folder** specify the folder where you are going to store your website files
* **Performance Test** leave as the default of Not Configured

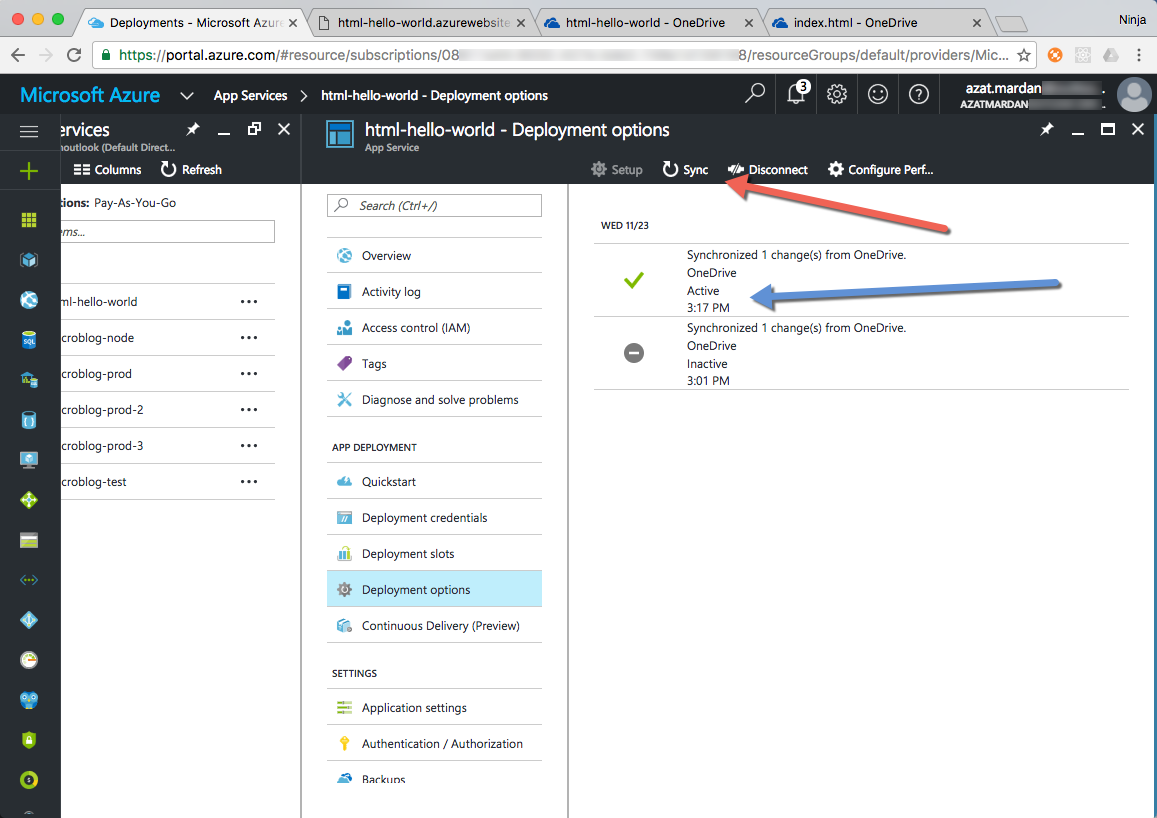
****

Deploy your website

Copy your index.html file into the OneDrive folder you specified when you configured the Deployment options of your app service. You can use upload button from OneDrive or drag and drop the file from Windows Explorer to OneDrive.

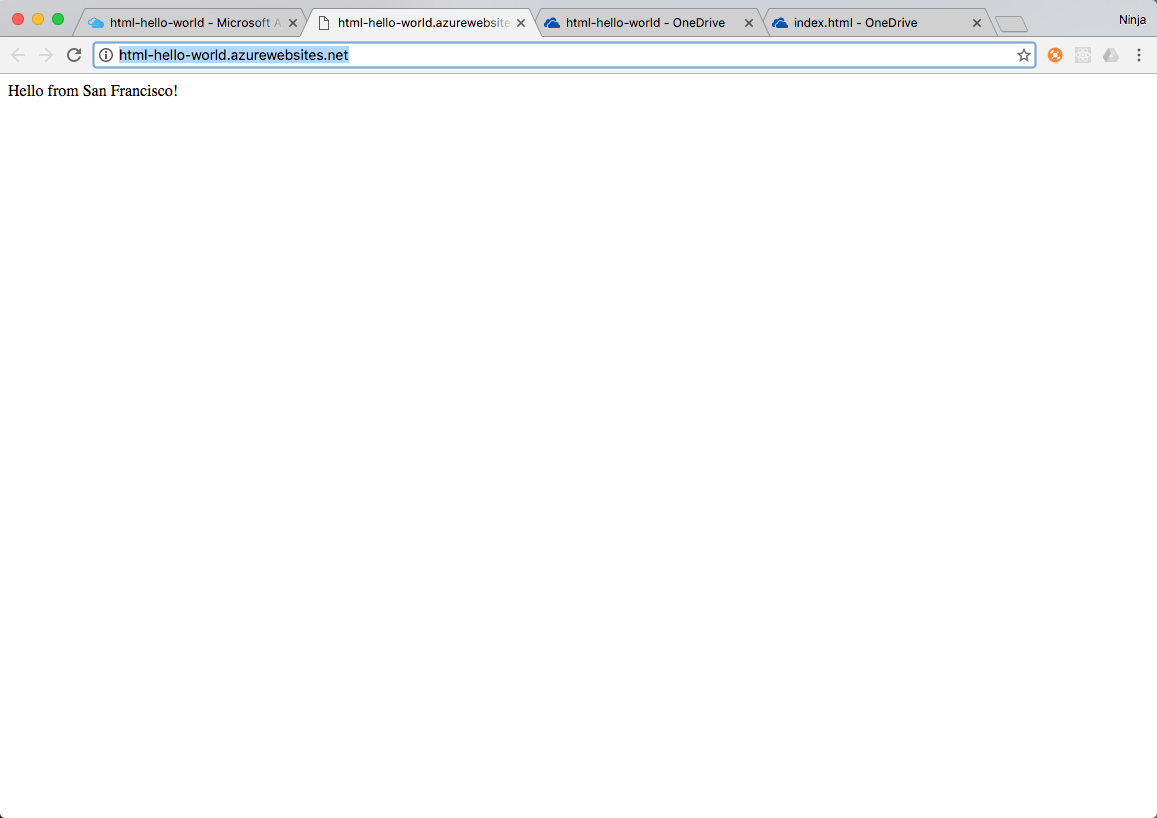
Synchronize your code

Go to Azure portal, located your app in App Services. Select Deployment options to see when the code was last synchronized, check the date and time of the last sync. If your code on OneDrive has been updated since the last synchronization and needs to be refreshed, select Sync on the top menu.



Test your website

Your page is deployed. You can open it by going to your browser and navigating to http://html-*yourappname*.azurewebsites.net/ . You can also navigate to your website by selecting the **Overview** tab in your app service in portal and selecting the URL on the overview tab. Your deployed website should look similar to the screen shot below.



Whenever you want to update the website, just update the files in OneDrive and those changes become available to Azure. You can also set up Continuous deployment to automate the synchronization.

You are not limited to HTML files. You can also deploy CSS, JavaScript and many other types of files. Even server-side code such as Node or PHP will work.

## Exercise 2: Deploying a Node.js script to Microsoft Azure via web interface (Portal)

In this exercise, you will use the Azure App Service settings to specify Node.js. Then, install a Node.js Timer app and run it in the cloud. You will use the OneDrive and App Service Editor to deploy the Timer Node.js app developed in [Module 2 Lesson 2](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module2/Lessons) (be sure to refer to lesson 2 throughout this lab).

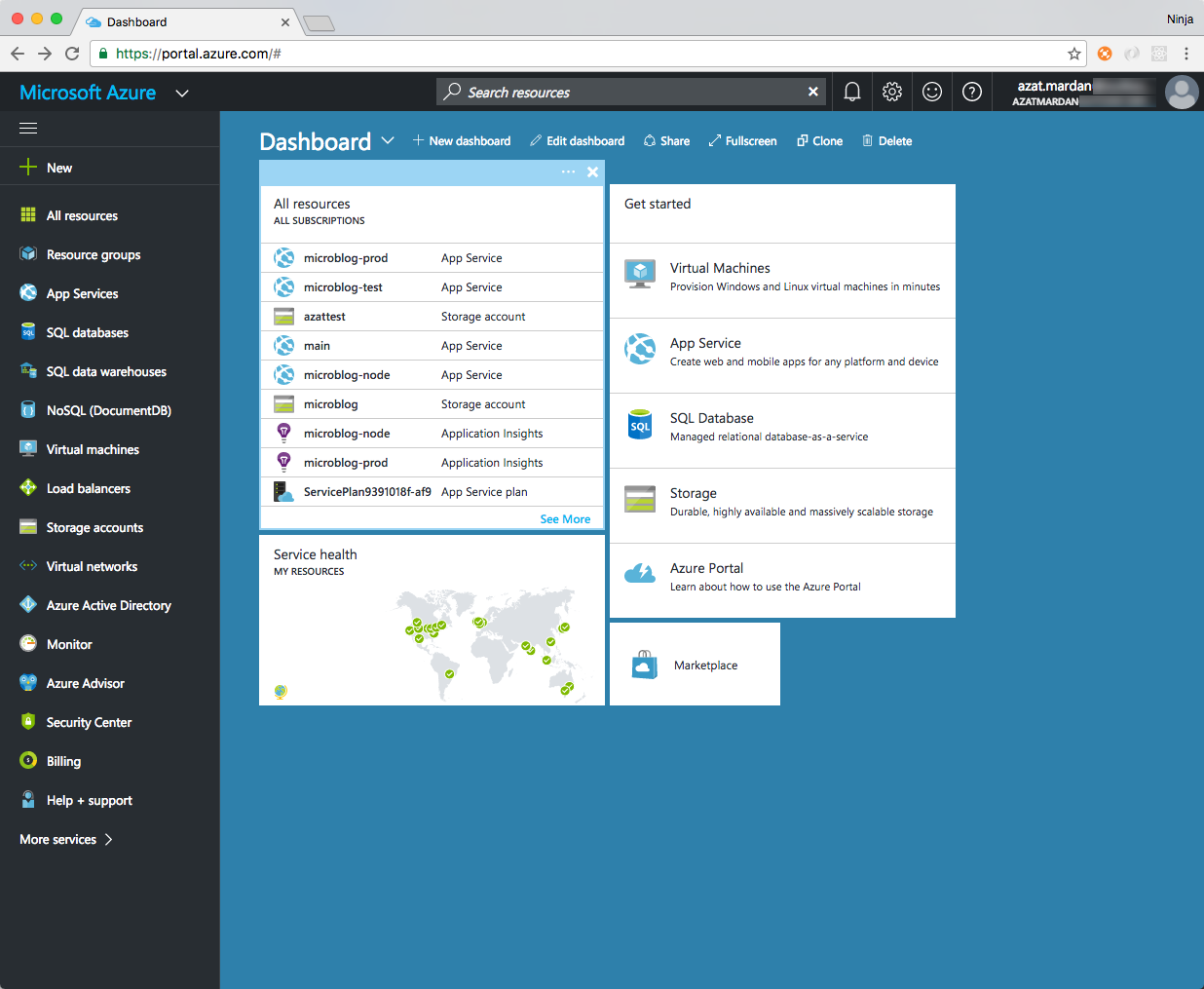
These are the high-level steps to complete this exercise:

1. Open Azure portal and create App Service by selection Web App (or use the service created in exercise 1)
2. Configure OneDrive as a Deployment option (must have OneDrive account)
3. Copy Timer app script (timer.js) into your folder in OneDrive (https://onedrive.live.com)
4. Go to Azure portal and ensure that the code has been synchronized.
5. Open the console and run Timer in the cloud.

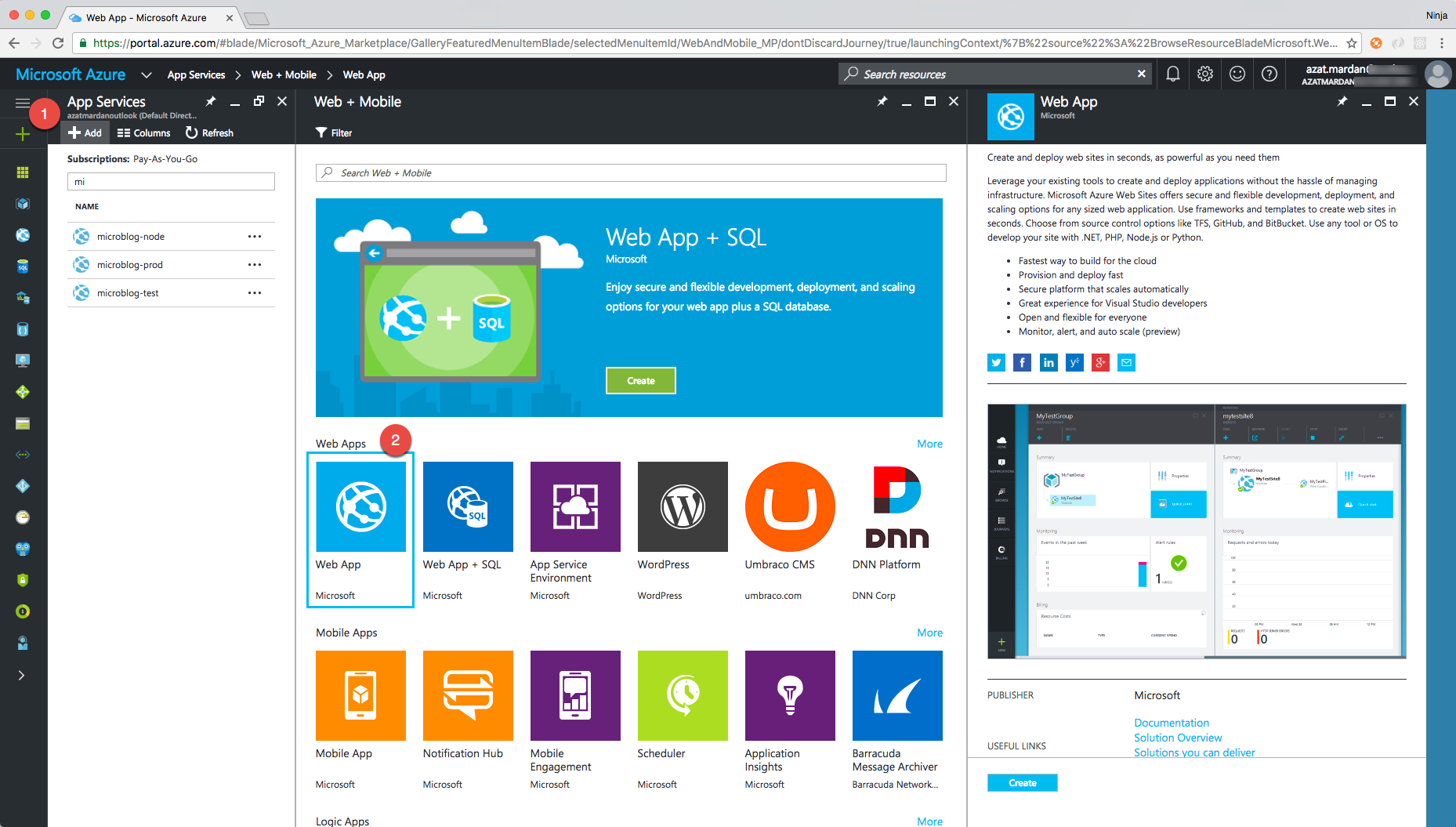
Pre-requisites

* Azure account
* Timer app (timer.js from the lesson 2 lab).

Open Azure portal. [Sign in to the Azure portal](https://portal.azure.com/). If you do not have an account you can sign up for a [free trial](https://azure.microsoft.com/en-us/free/)

****

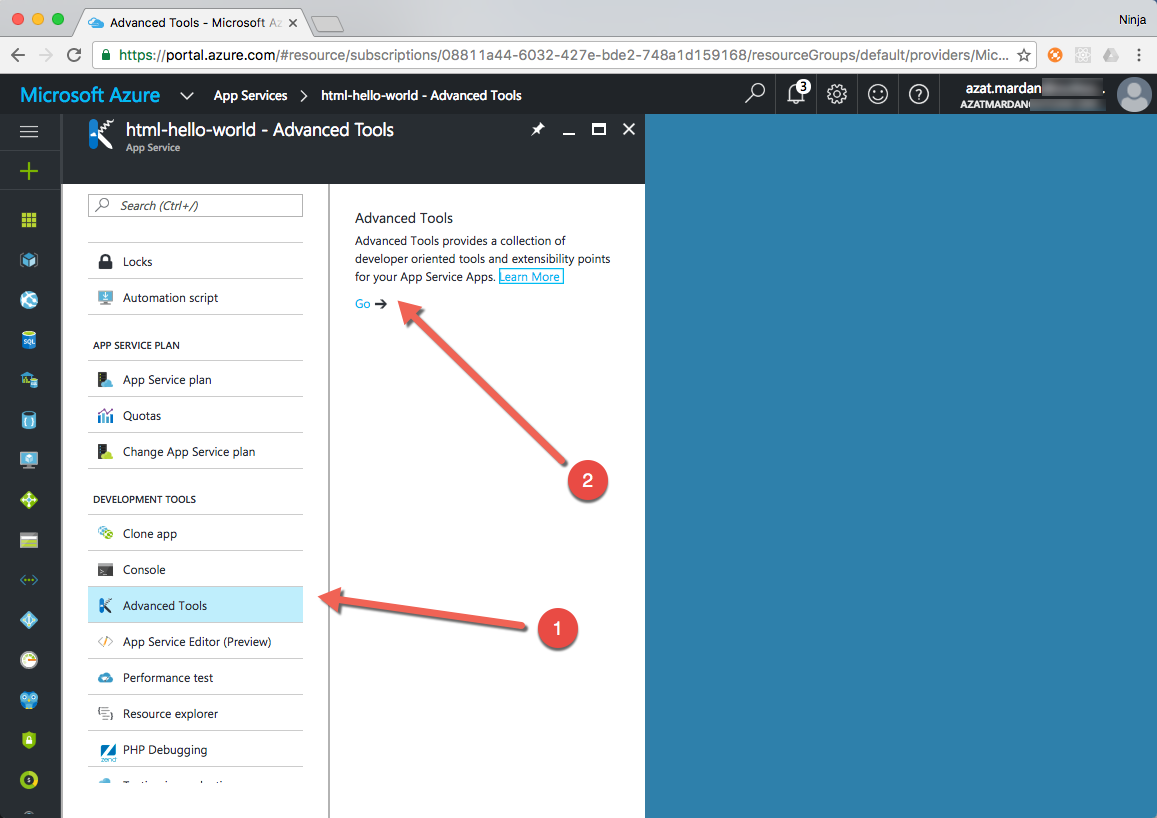
Select the app you created in Exercise 1 or create a new App Service. To create a new App Service select **App Services** from the left menu. Create a new service by selecting **+ New | Web App**

****

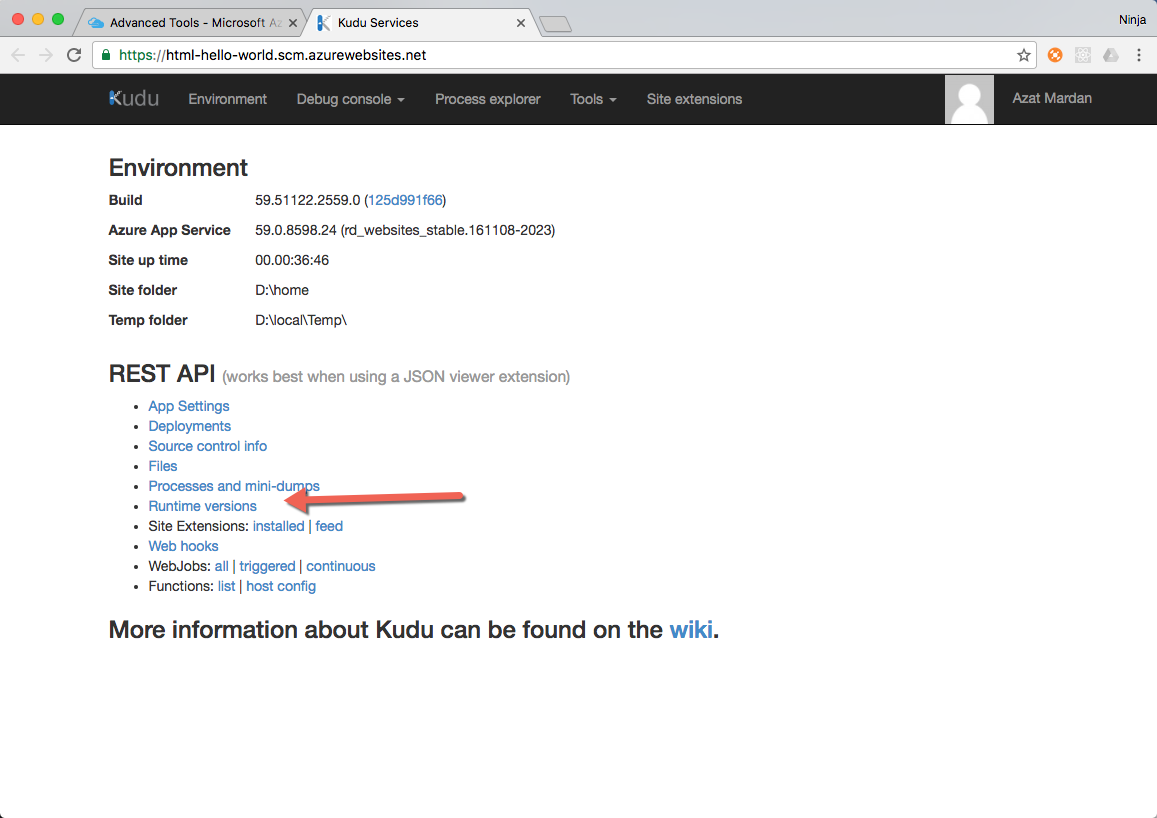
Specify Node.js version for our App Service

We need to specify the version of Node.js version we want Azure to use when we run our script. Azure App Services is a PaaS which means developers have some control over the environment. You can control the versions of Node.js to run. Let's check what versions are available. This way you can find out about new versions in the future.

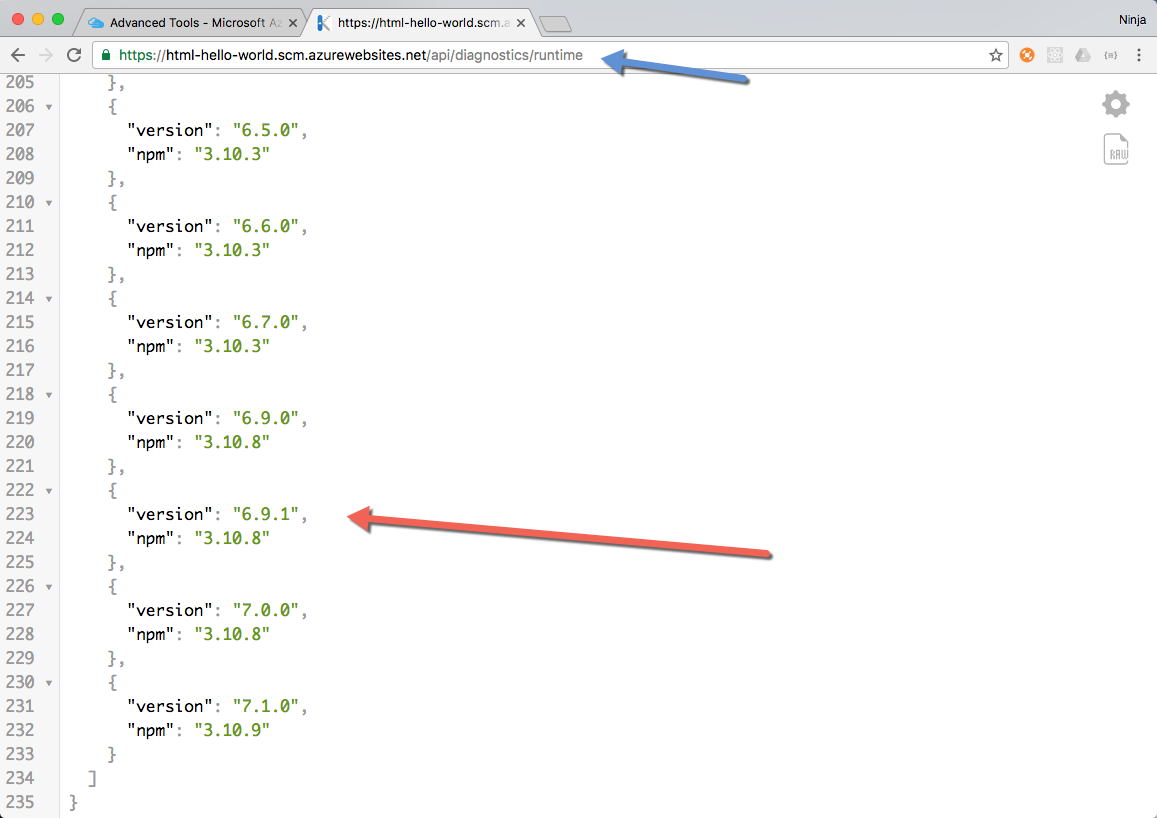
In the Azure Portal, navigate to your app. Select **Development Tools | Advanced Tools**. Select **Go** on the Advanced Tools blade.



You will be brought to advanced tools for your service ( https://*yourappname*.scm.azurewebsites.net)



Select **Runtime versions** or you can navigate there directly https://*yourappname*.scm.azurewebsites.net/api/diagnostics/runtime. You will see a list of available Node and npm version pairs in JSON format.

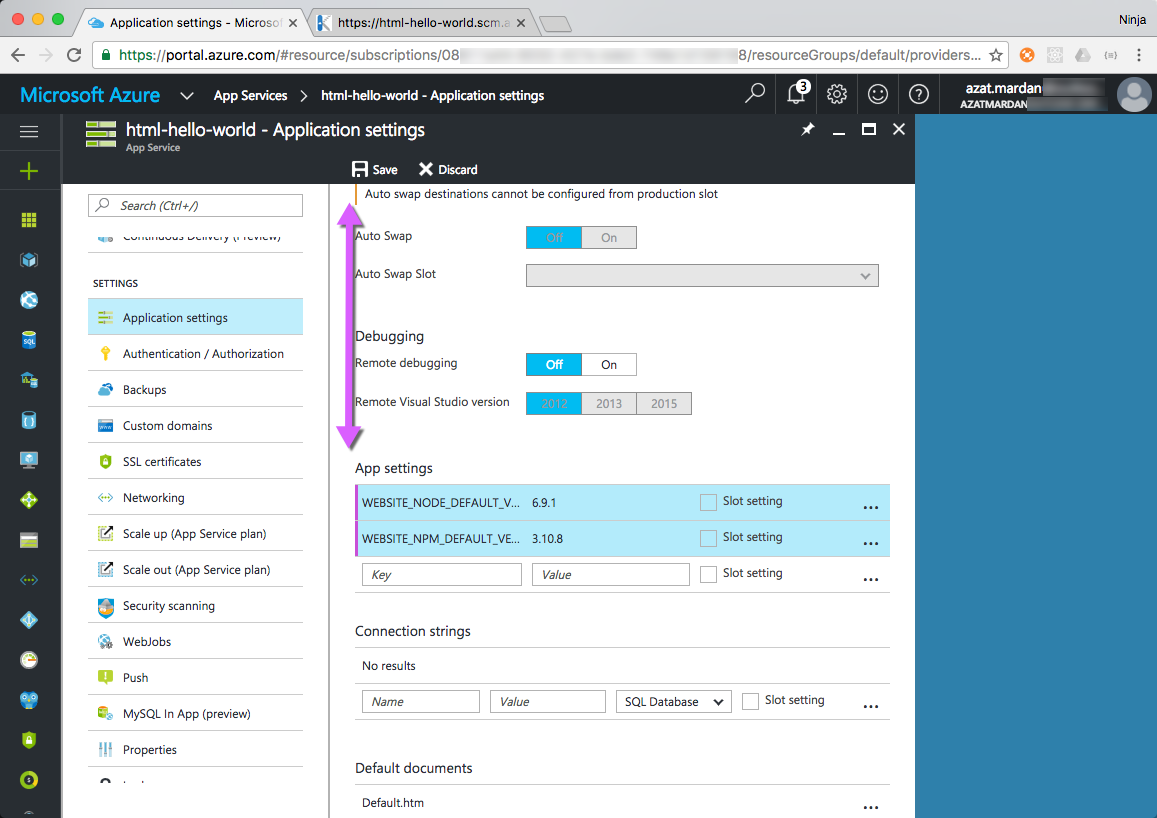


Now we know what versions are supported, we can return to the Azure portal. We are going to use versions 6.9.1 for Node and 3.10.8 for npm. In the Azure portal navigate to your app service.

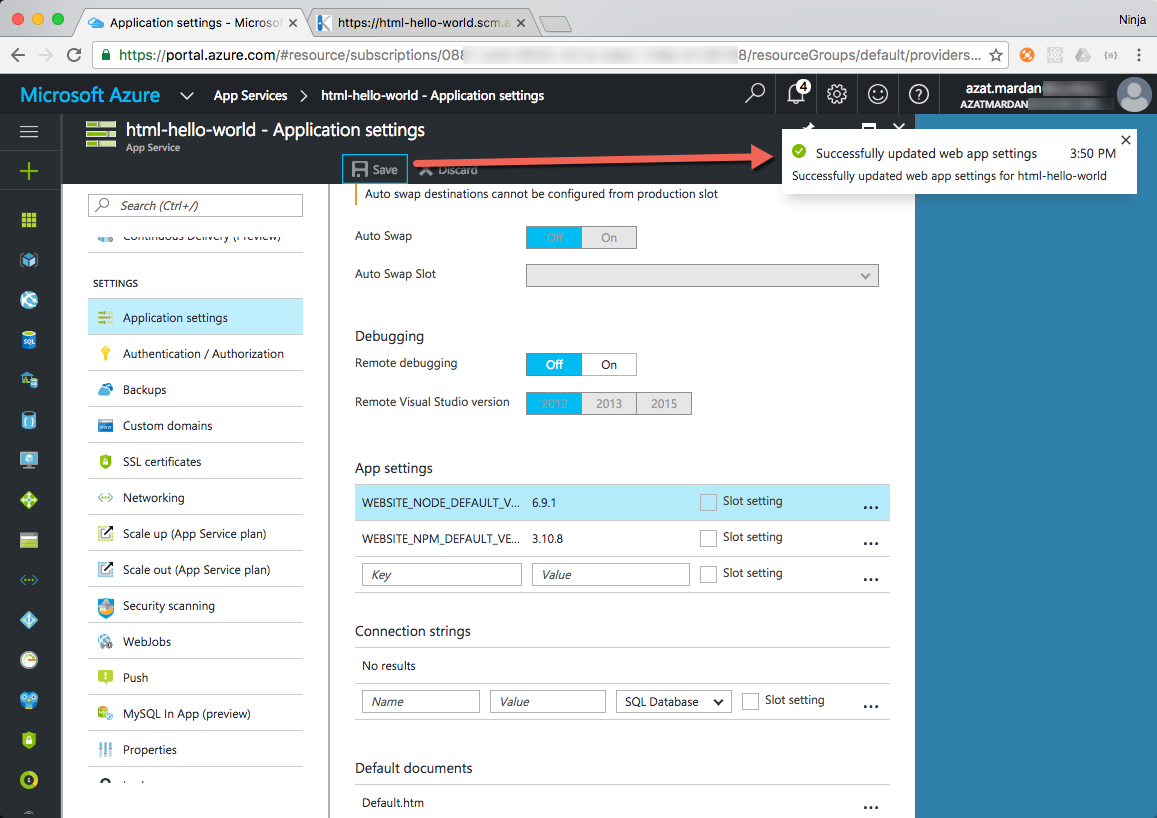
Select **Settings | Application Settings**

Scroll down to App settings.

These are the environment variable which Azure will set for this service. We can control Node.js and npm versions by setting the values for WEBSITE\_NODE\_DEFAULT\_VERSION and WEBSITE\_NPM\_DEFAULT\_VERSION respectively. Set WEBSITE\_NODE\_DEFAULT\_VERSION to 6.9.1 set WEBSITE\_NPM\_DEFAULT\_VERSION to 3.10.8 as shown in the screenshot below.



When you update the app settings to specify new versions, you will see a purple bar beside the modified app settings. This indicates the changes are not saved. Select **Save** in the top menu to save your changes.



Upload your node.js app

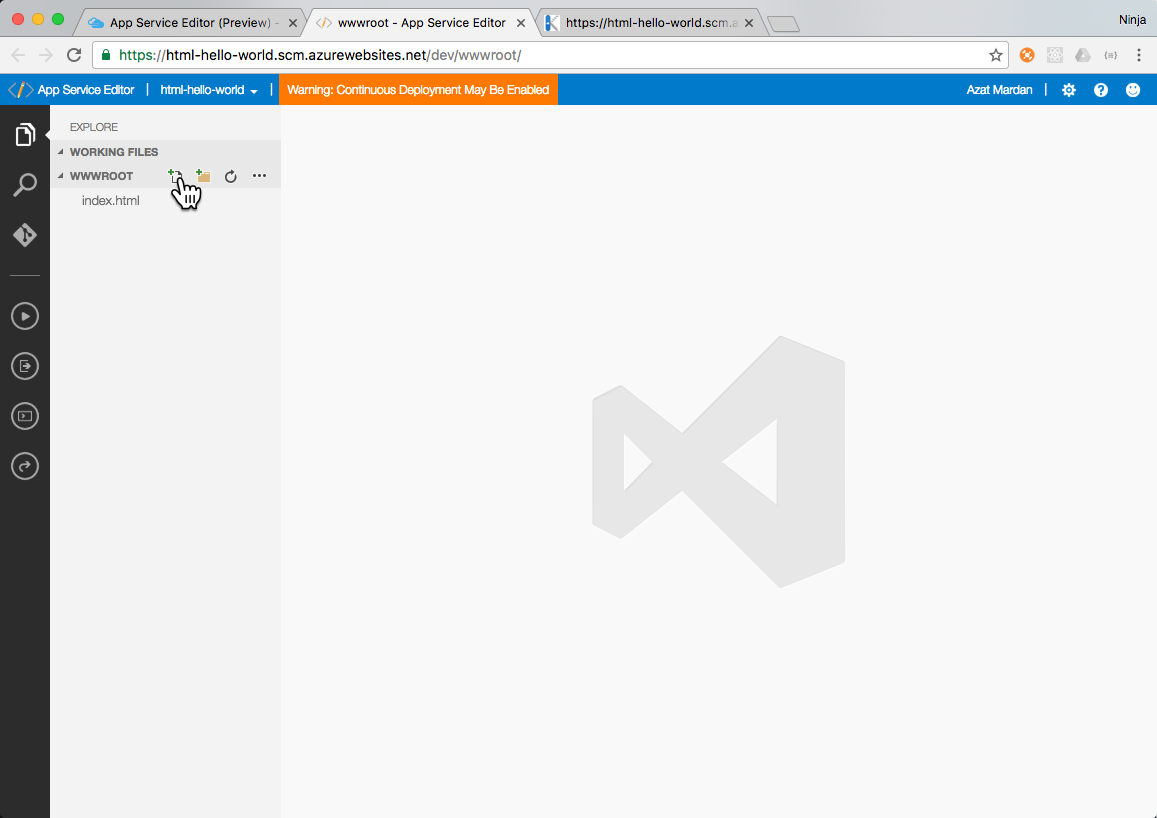
Upload your timer app to OneDrive and synchronize the changes to Azure using the steps you followed in Exercise 1 for your HTML file or use App Service Editor as described below.

If you want to try App Service Editor follow the steps below

App Editor looks similar to VS Code and is a great tool to make changes and run code in Azure directly. When using other deployment options (Git, FTP, OneDrive), keep in mind that your manual changes completed with App Editor might be overwritten when you synchronize files.

To create a file in App Service Editor, navigate to https://*yourappname*.scm.azurewebsites.net/dev/wwwroot/ or select **Development Tools |** **App Service Editor** in the Azure Portal.

In App Service Editor, you will see the wwwroot folder in the left menu. You can hover over the folder name which will pop up the icons to create a new file. Select Create New File as shown in the screenshot below.



If you do not have the completed timer.js file, you can use the code; create your new file in the wwwroot/ directory, and name it timer.js. The only difference between this code and the code implemented in lesson 2 lab is that this code uses ES6/ES2015 features which are not available in by default in Node 4 which Azure has by default, only in 6 and higher. This allows us to test we correctly configured Azure to use Node 6 for our App Service.

let timeLeft = 0

const argv = process.argv

if (!argv[2]) {

console.log('Please provide an argument, e.g., 5s or 1min.')

process.exit(1)

} else if (argv[2].indexOf('s')>-1) {

timeLeft = parseInt(argv[2].slice(0, -1), 10)

} else if (argv[2].indexOf('min')>-1) {

timeLeft = parseInt(argv[2].slice(0, -3), 10) \* 60

} else {

console.log('Please provide a valid argument, e.g., 5s or 1min.')

process.exit(1)

}

let interval = setInterval(()=>{

timeLeft --

console.log(`Left: ${timeLeft}s`)

if (timeLeft == 0) {

clearInterval(interval)

process.exit(0)

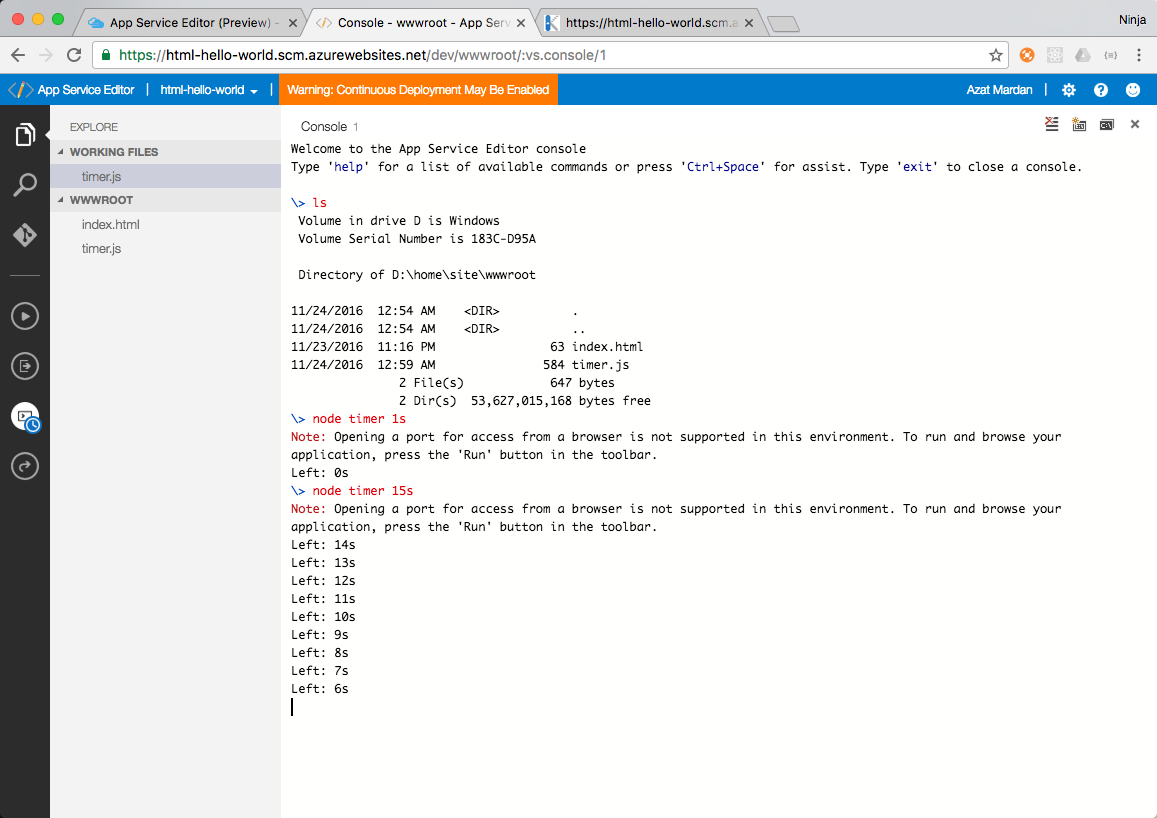
}

}, 1000)

Save the file and select the console icon from the right side menu (Open Console).

Type ***ls*** to see the list of scripts

Type ***node timer 1s*** or ***node timer 15s*** to run your script



Running Node in Azure App Service is straight forward and will give benefits such as:

1. Location and device independence: You can use a browser in any OS. You don't even need to have Node installed on your dev machine.
2. Cost: when running CPU-intensive operations it's more cost effective to use cloud because you can turn it off when you don't need it and turn it on when you need it. You can also scale it as needed when needed.
3. Fast time to deploy: It took us just a few moments to deploy our code to Azure and make our service accessible via a public URL and deployed code.

## Summary

In this hands-on lab, you learned how to:

* Use Azure App Services in Azure Portal: create, deploy, configure access advanced development tools.
* Deploy code via OneDrive
* Modify and launch code from App Service Editor