Web Development: Module 2, Lesson 8  
Deploying to Azure Hands-On Lab

## Overview

Building on [Module 2 Lesson 7](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:

* Learn how to use Azure CLI tool
* Prepare your project for deployment
* Deploy your project to cloud

## Prerequisites

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

* A text editor
* Windows PowerShell, Mac Terminal, or some other shell with node.js and npm installed
* Completion of all [Module 2 Lessons](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.
* Git

## Exercises

This hands-on lab includes the following exercise:

* Exercise 1: Installing Azure CLI and Deploying a web app to Microsoft Azure via Git and CLI

## Exercise 1: Installing Azure CLI and Deploying a Web app (Recommended)

In this exercise, you will use npm to install the Azure CLI. You will use the Azure CLI to deploy the hello world app developed in [Module 2 Lesson 7](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module2/Lessons) (be sure to refer to lesson 7 throughout this lab).

1. Open a shell and enter the command

$ npm i -g azure-cli@0.10.1

1. Log in to Azure CLI using the command

$ azure login

1. Prepare your project by initializing package.json with all the necessary information
2. Create a web.config file (optional).
3. Create Azure Site (app) with --git (or add Git remote manually)

$ azure site create –git {appname}

1. Get your Azure Git and FTP deploy password (if you don't have it already) through the Azure portal
2. Put Azure Storage env vars into this app's cloud settings (AZURE\_STORAGE\_ACCOUNT and AZURE\_STORAGE\_ACCESS\_KEY)

$ azure site appsetting list

$ azure site appsetting add NODE\_ENV = production

1. Add code to the local repository and deploy by pushing code to Azure

$ git push azure master

Let’s dive deeper into using CLI to install apps. If you are familiar with Git, this approach will be the easiest for you since it won't require clicking around and going to Azure Portal.

Firstly, open a shell (a.k.a. terminal or command prompt) and enter the command to install Azure CLI version 0.10, if you don't have it already. We are using npm for it so you need to have npm (version 3 or higher):

$ npm i -g azure-cli@0.10.1

Assuming you installed Azure CLI successfully, log in to Azure CLI using the command below. Keep your credentials nearby. You'll need to do it just once.

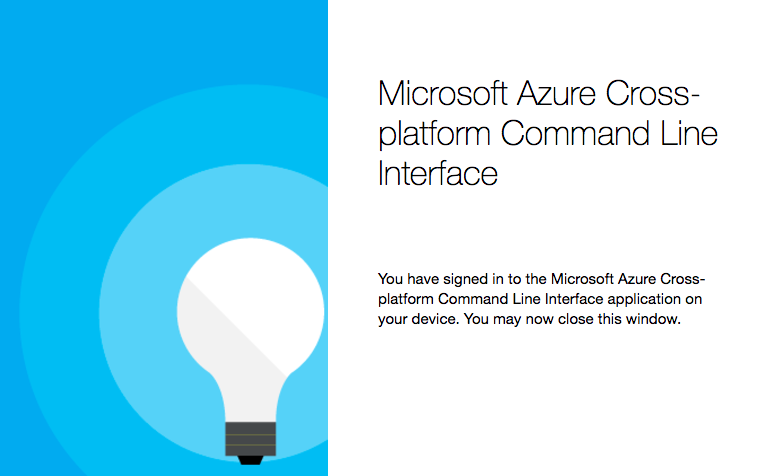
$ azure login

You should see this message:

info: Executing command login

- Authenticating...info: To sign in, use a web browser to open the page https://aka.ms/devicelogin. Enter the code ABC to authenticate.

Open https://aka.ms/devicelogin in a browser and enter the code (yours will be different from ABC). After entering your creds and all the dancing with redirects and entering your creds again you'll see this page:



And the terminal will say something like that depending on your subscription:

\info: Added subscription Pay-As-You-Go

info: Setting subscription "Pay-As-You-Go" as default

+

info: login command OK

Soon after you logged in, we can prepare your project. Navigate to the folder with the source code you want to deploy. Create Azure Site (app) with the --git option (or add Git remote manually) where {appname} name of your app, e.g., microblog-prod-3:

$ azure site create {appname} --git

Answer the question about location. Then pick a your Azure Git and FTP deploy password (if you don't have it already). This can be any silly names. Azure wants you to use different name for Git/FTP than your main login. Pick password and then confirm it. These creds are only for pushing the code (deploying), not for Azure portal. Still, you might want to write them down. At this point the app is created in the cloud and configured for deploys via Git and FTP... but it does NOT have the code yet.

Run git remote -v to double check that Azure CLI added the remote which is these values in my case:

azure https://microblog-git@microblog-prod-3.scm.azurewebsites.net/microblog-prod- 3.git (fetch)

azure https://microblog-git@microblog-prod-3.scm.azurewebsites.net/microblog-prod-3.git (push)

Update package.json to have versions:

"engines": {

"node": "6.9.0",

"npm": "3.10.8"

},

Add this web.config file to the project. It will tell Azure how to route traffic and what Node file to launch, i.e., app.js. The configurations in the comments are just for your references and are NOT needed for this exercise.

<!--

This configuration file is required if iisnode is used to run node processes behind

IIS or IIS Express. For more information, visit:

https://github.com/tjanczuk/iisnode/blob/master/src/samples/configuration/web.config

-->

<configuration>

<system.webServer>

<handlers>

<!-- indicates that the app.js file is a node.js application to be handled by the iisnode module -->

<add name="iisnode" path="app.js" verb="\*" modules="iisnode" />

</handlers>

<!-- Make sure error responses are left untouched -->

<httpErrors existingResponse="PassThrough" />

<rewrite>

<rules>

<!-- Don't interfere with requests for logs -->

<rule name="LogFile" patternSyntax="ECMAScript" stopProcessing="true">

<match url="^[a-zA-Z0-9\_\-]+\.js\.logs\/\d+\.txt$" />

</rule>

<!-- Don't interfere with requests for node-inspector debugging -->

<rule name="NodeInspector" patternSyntax="ECMAScript" stopProcessing="true">

<match url="^server.js\/debug[\/]?" />

</rule>

<!-- First we consider whether the incoming URL matches a physical file in the /public folder -->

<rule name="StaticContent">

<action type="Rewrite" url="public{REQUEST\_URI}" />

</rule>

<!-- All other URLs are mapped to the Node.js application entry point -->

<rule name="DynamicContent">

<conditions>

<add input="{REQUEST\_FILENAME}" matchType="IsFile" negate="True" />

</conditions>

<action type="Rewrite" url="app.js" />

</rule>

</rules>

</rewrite>

<!-- You can control how Node is hosted within IIS using the following options -->

<!--<iisnode

node\_env="%node\_env%"

nodeProcessCommandLine="&quot;%programfiles%\nodejs\node.exe&quot;"

nodeProcessCountPerApplication="1"

maxConcurrentRequestsPerProcess="1024"

maxNamedPipeConnectionRetry="3"

namedPipeConnectionRetryDelay="2000"

maxNamedPipeConnectionPoolSize="512"

maxNamedPipePooledConnectionAge="30000"

asyncCompletionThreadCount="0"

initialRequestBufferSize="4096"

maxRequestBufferSize="65536"

watchedFiles="\*.js"

uncFileChangesPollingInterval="5000"

gracefulShutdownTimeout="60000"

loggingEnabled="true"

logDirectoryNameSuffix="logs"

debuggingEnabled="true"

debuggerPortRange="5058-6058"

debuggerPathSegment="debug"

maxLogFileSizeInKB="128"

appendToExistingLog="false"

logFileFlushInterval="5000"

devErrorsEnabled="true"

flushResponse="false"

enableXFF="false"

promoteServerVars=""

/>-->

</system.webServer>

</configuration>

In app.js itself, slightly modify the way we bootup the server to accommodate Azure's iisnode which import app.js:

if (process.env.NODE\_ENV == 'production' || require.main === module) {

app.listen(3000, function(){

console.log('Express server listening on port 3000')

})

} else {

module.exports = app

}

Finally, we need to put Azure Storage and some other env vars into this app's cloud settings. We don't want to deploy secrets in start.sh file.

AZURE\_STORAGE\_ACCOUNT, AZURE\_STORAGE\_ACCESS\_KEY as well as NODE\_ENV are set in the cloud from CLI similarly. To start, get a list of current vars:

$ azure site appsetting list

Then, set the vars one by one using this command (change the key and value):

$ azure site appsetting add NODE\_ENV=production

Right after that, we are ready to deploy. Add code to the local repository with git add . and git commit -m 'initial commit'... and deploy by pushing code to Azure:

$ git push azure master

Every time you make a change to the code and want to re-deploy, you need to execute only these commands:

git add .

git commit -m 'changes'

git push azure master

Open you app at http://APP\_NAME.azurewebsites.net where APP\_NAME is the name of your app.

## Summary

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

* Learn how to use Azure CLI tool
* Prepare your project for deployment
* Deploy your project to Azure