Use of Microsoft's Platform as a Service

Demo

Instructor: In this demonstration we are going to take advantage of Microsoft's Platform as a Service capability to install a simple application, and configure a supporting Web Application Firewall. Along the way we will address some other cloud security best practices in Microsoft Azure.

Let's go ahead and create a new app service. We have to select the Azure subscription this will be tied to, and identify the Resource Group it will belong to. Since there are none currently available, a new one will need to be created.

We will enter an application name. Note, this must be unique as the

shared domain suffix will be azurewebsites.net by default.

The runtime stack will be .NET Core 3.1 and we'll see where that comes in to play a bit later.

The region our application will be deployed in is set to Central US.

Within our app service plan it has defaulted to a standard size configuration. We can change that, and different options are available to support compute, memory, storage, and other scaling requirements. We are going to select the smallest free tier possible for our first application.

When we proceed to configuration of our application insights, we run into a problem. Our subscription already has an insights configuration for US East, so if we want to leverage this, we must deploy our application there, or create a new one for central.

Let's reconfigure our app settings. When we return to reset the region, notice that it also reset my selection for Sku and size. I'll need to change that back to F1, the smallest, shared infrastructure option available.

Now we can enable application insights. We are not going to add any tags. And now, review and create.

Once deployment is complete, we have a new dashboard available that presents detailed information aout our new app service. And we can

open the URL that has been setup to see our default web site. Also notice that this has TLS enabled by default using a Microsoft signed certificate. We could choose to replace this with our own certificate later on, but this is a nicely integrated feature.

Microsoft does have a version of Visual Studio for the Mac. The install is quite painless, and I already have ready to support Azure publishing. We will also take advantage a some freely available source code from Microsoft to setup our first Azure app.

Inside of visual studio we will create a new web application project, no authentication, and HelloWorld-Test as the name.

Now we want to grab that source code from its Git repository. Click Version Control, Clone Repository. Enter our Git URL, and click clone.

A quick look at the application options shows us .Net Core 3.1 target framework, which is why we previously selected that when setting up our app engine in Azure.

We are going to make a small code change, just so we can verify this is our code being deployed.

Let's do that now. Build, Publish to Azure. I'm already logged in through visual studio so my list of available app services shows up. I will select the only one I have available and click publish.

Once complete, and with a browser refresh, we see our default Azure page has been replace by our new code.

Now we are going to setup some developer permissions. In our Azure Active Directory I already have a dev2 user standing by. That user does not have any assigned roles or groups at this time.

We will jump over to our webapp and verify that dev2 is not currently assigned. Now, let's add dev2 to this service. Add. Add role assignment. You can see the descriptions of each role pop-up as we mouse over each. We are going to go with the contributor role. We will select our Azure AD account to pull our user list from, and then click on dev2 to get assignments setup. But, before I click save, let's see if dev2 can publish right now.

I need to go back to visual studio and sign out, and then sign in with dev2. We will change our web page a little, just so we can easily confirm the update after we publish. But when we go to publish to Azure now, we don't have any app services available to us.

Returning to Azure we will save our new role assignment, and confirm the configuration in the Role Assignment section.

Now we can return to visual studio, refresh our list of available app services, and here we go, there is

now one available. We will select it, and click publish. Upon completion and refresh, we see that our code update was pushed.

Next we are going to configure a Web Application Firewall or WAF to provide additional protection. The WAF will evaluate the request headers and other elements against a set of policies to allow or deny the request. If permitted, the request will get forwarded to our actual web application.

When creating our policy, we will assign it to our global front door.

Assign it to the resource group that our web app is also assigned to. And give it a name.

We can set our policy to detect threats or to block them. We are going with prevention today. And we will customize our response message to verify it is ours, and not the default from some other service. We can take a peek at the default rules which are setup to prevent SQL injection, cross site scripting, and other typical web attacks.

We are going to add our own custom block rule. We'll give it a name, ensure it is enabled, and then set some conditions. There are several things available to evaluate; Such as request headers, cookies, the request body, etc. We are going to do a string match on a query string. We will use blockme as the string we will consider bad, and ensure that we

have deny traffic as our action. We could choose to allow or redirect to another URL also. And our priority number listed above would come in to play if a request matches multiple conditions.

We need to add a frontend host, but unfortunately, we don't have one available yet. We'll have to come back and assign this in a minute. No tags, review, and create.

Let's do a couple of quick tests. First a simple SQL injection string that should match one of the rules we saw. Our page loads fine. A similar test with our blockme text still works. This is expected because we haven't actually finished our WAF configuration yet.

Navigating to our WAF policy, we still do not have a frontend assigned. We need to do that now. So, Front Doors. Add. Select our subscription and resource group. And next is a configuration wizard.

We need a unique name for azurefd.net. And ensure we enable the WAF.

This front end needs to point to a backend application that exists within the resource group we previously identified, so we need to select app service and then our webapp08162020. Next is the backend pool configuration and we will accept the defaults on this.

Almost done, we need to add a routing rule. This last step directs requests that hit this front door to the specific WAF backend pool we just configured. Review and create.

If we revisit our WAF policy we see a frontend host identified along with a URL. To leverage the firewall, all requests must hit this URL for inspection, and upon success, they will get forwarded to our app engine. So, let's open that URL.

We will attempt our sample SQL inject text on the URL. We know the WAF blocked this request because of our custom error message. One more test using our blockme query sting. And that too is blocked. Perfect.

There is just one more thing to take a quick look at. Let's go to settings for our new application. Then configuration, and platform settings. Here we can see our stack definition and other parameters. I want to disable FTP state. We won't be using FTP to deploy our application, and therefore we should ensure this unnecessary service has been disabled.

We also see in here where we can configure our default web documents, set snapshot and backup schedules, and even import your own SSL or TLS certificates.