App Service Demo Script

# Intro

## Setup

Run Initialize.ps1 script that creates a Resource Group with all the needed resources for this section:

* 3 App Services
* 2 App Service Plans in different regions

The script takes a single parameter called $prefix to ensure uniqueness on all the resources created.

/SpeakerResources/Intro/Setup.ps1 -prefix nunos

# Configuration

In this area we cover the following topics:

* Creating App Services, App Service Plans and Resource Groups
* Configuration features of App Services such as Application Settings, Custom Domains, SSL and Backups
* A Powershell primer on how to automate these actions through scripting
* Creating a Traffic Manager profile

## **Demo Script**

### Resource Groups

**Go into the portal and expand All Resources -> Explain how tricky it can be to manage all this as isolated components**

**Expand Resource Groups and explain how Resource Groups help on this regard. Explain the Location is irrelevant.**

**Expand the WorkshopIntroRG resource group. Go through the several resources inside.**

**Show that you can Delete an entire group**

**Show that you can Automate its creation through an ARM template. Show all the different options and then ”Add To Library”.**

**Browse to “Templates” and try to deploy the new template. Show the workflow without completing it.**

**Create a Resource Group MyTestRG**

**Go into previous Resource Group and click Move to show you could move all resources to a new Resource Group (but don’t do it).**

**Click the little “Edit” button near the Subscription name in the Overview blade and show how you could move this to a different subscription.**

### **Service Plans**

Explain that while Resource Groups are generic containers, App Service Plans are specific to App Services.

Expand **“App Service Plans”** (Browse -> App Service Plans)

Use the **Columns** button to add Location and Apps to the table.

Open the App Service with 2 apps and choose the “**Apps”** blade to show which apps are running on it.

Open one of the apps inside the Service Plan

Explain that **all scaling operations happen on App Service Plan level**

Click on **App Service Plan**

Open **Scale Up** and explain the several tiers – scaling will be covered in more detail later

Point out that we are running on a **S1** Plan.

Look into the App Service Plan to show that indeed it’s a **Standard: 1 Small** (explain that the instance count is actually done through scale out).

Explain that this app might not require as much resources but we don’t just want to scale down the entire plan because other apps are living in it, so we’re going to create a new one.

**Create** a new App Service Plan **<prefix>-FreeWestEurope** using **Free** and **West Europe** and put it in the **WorkshopIntroRG** resource group

Open the **<prefix>-LabBackoffice** using the **App Services** menu

Click the **Change App Service plan**

Move to new **<prefix>-FreeWestEurope** service plan.

**Explain that we are charged per Service Plan**. In this case, the old service plan is free, but if it was not we would still be charged for it even though no application is running on it.

### App Service

**Create** a new App Service

* Try **“coolwebapp”** – notice DNS validation saying this name already exists
* Use **<prefix>-coolwebapp**
* Choose the **WorkshopIntroRG**
* Choose the **StandardWestEurope** plan you created (Standard, West Europe)
* **Pin to Dashboard**

**Browse** to it and see the default hostingstart.html.

Explain that we need to deploy code to it – covered in a later session.

**Create Web App** from Marketplace Template

* New -> Web + Mobile and then see All on that blade -> click More on Web Apps and see the entire list for web apps
* Point out the **Web App + SQL**
* Create a **Wordpress** app and click create - put it on our resource group, let it create a new service plan, configure database quickly -> Pin and create
* This may take a little more time, when we browse to it we go to the standard wordpress setup

### Config Lab – part 1

Open Powershell ISE

On the console navigate to the **Labs/Configuration/Scripts** folder

Open Login and sandbox scripts

Go through Login and make sure you are assigned to the correct subscription

Run Initialization script and show things in the portal after it’s done

/Lab/Config/Initialize.ps1 -prefix nunos

Go through the Powershell lab.

One this finished you will have a new app in a Standard hosting plan:

* **<prefix>-labwebapp**
* **<prefix>-StandardWestEurope**

### WebApp configuration

Explain that we created an app on a Standard hosting plan to enable all configuration options.

Show that some options are disabled on the regional sites but are available on the **<prefix>-labwebapp.**

Go to one of the Free apps and show **Custom Domains** and then click **quotas – compare it to the standard one**

Go into overview tab of new app and explain the general information there (App Service, hostname, etc…)

Go through menu on the left and explain the several areas for tools and information

Open Application Settings and go through main settings.

#### Default Documents

* Notice the **hostingstart.html**
* Open the Console (under Development Tools), do **dir** and then **move hostingstart.html somepage.html**
* Update Default document

#### Explain AppSettings using **eu-prefix-labwebapp**

* Browse app and explain 2 settings being displayed
* Use App Service Editor to show web.config
* Set Application Setting called **AppName**
* Add PHP file to it and show that it works too.

#### Access Control

* Add my outlook account as a website contributor
* Login using InPrivate browsing and notice I can’t add users or look into service plans

#### Custom Domains and SSL

Configure **Custom Domain Name** on new web app

* Go through the wizard and just add nunostechstuff.com and [www.nunostechstuff.com](http://www.nunostechstuff.com)

Add **SSL** support

* Use imported certificate and create bindings

### Config Lab – part 2

Introduce the topic of complex configuration and consistent deployment.

Explain Azure Resource Manager

On **WorkshopIntroRG**, expand **Deployments** and look into the deployment operations and templates

Open **resources.azure.com** and explain how those templates correspond to items on the API.

Go to our new website

Show that we can’t do everything directly through scripting, like setting up **alwaysOn (**show lack of script option)…patch it in resource explorer

Show how you could do it in powershell as well (in the commands tab)

Show **Powershell** tab on resource explorer and walk through a couple of examples showing that the Get and Set are always the same parameters.

Open **Fiddler** and use the Set-AzureRM command to set application settings. Notice how 9 requests are made, notice the methods and the actions.

Navigate to the appsettings under your website and just run the powershell command to patch it (with different values)

Go to <http://github.coM/davidebbo> and show the script examples

Show the point is that, **while you can definitely craft everything in script, there are better ways to handle the provisioning of your website.**

Open Visual Studio Code on **ARMTemplates** and show SimpleWebSite

Explain you can write templates in a specific format. Go through the Empty template and the SimpleWebSite.

**How do we create these?** Mention Visual Studio but create a new **Template Deployment** in the portal.

Add a **WebApp** -> show that it also creates a service plan

Go through variables and parameters

Add a **second** **WebApp**

Add a **StorageAccount**

Save and show how it looks if we try to provision

Back in Edit mode show selection for Quickstart templates -> choose web app and sql template

Go to the GitHub repository azure-quickstart-templates

Notice the Deploy to Azure button and click it -> Do not deploy

Back on template deployment just save template -> Open templates in Portal and show how it works. Mention that it is very similar to the experience of creating a Web + Sql site

Open **WebSite.json** and walkthrough the template

**Run powershell script to install it (OPEN WITH ISE)**

Make **changes to template**

**Run powershell again with NEW LINE ADDED**

Leave people to see **ToDo** and explain we don’t need to parametrize everything. Explain single prefix parameter.

**Show my ContactManager template and connection to powershell script**

**Explain Backup stuff**

# Deployment & Scaling

### Traffic Manager

Create Traffic Manager Profile on **WorkshopIntroRG** using **Performance**

* *Optionally, run on the* ***<prefix>-LabRG*** *but make sure you up-scaled the plans to Standard using the provided script*

Explain that is DNS based routing

Explain the 3 different methods

* Priority = Failover
* Performance = Latency
* Weighted = Round Robin with equal weights
  + Gradual Application Upgrade
  + Migration to azure using both external and internal endpoints
  + Cloud-bursting an on-premises deployment by putting it behind a TM profile

**Add Endpoints** – EU-<prefix>-LabWebApp and US-<prefix>-LabWebApp

Open **Configuration** and reduce the TTL to 30 seconds and explain monitoring

Now open **nslookup** and test the traffic manager CNAME

Go to **digwebinterface.com** and test with both Europe and US datacenters

Change Traffic Manager to **Priority**

Show Endpoints priority changes

Do nslookup again and browse to the site

**Stop** priority website and wait 30 seconds

**Browse** again and notice we switched to the other one

### Scale Up

Use ContactManager app

Say that webApp is using more memory than I expected - more users, etc...

When you scale up, you scale the entire App Service plan so you affect all apps inside that plan (go from app -> to app service plan -> apps)

Click **Scale-up and explain several tiers** (from free to up)

Explain we're staying in Standard because it has all the features including auto-scale but we're going to move up the tier to **S2**

Change to **S2** and show how fast it was -> it changed to a different virtual machine -> although if you have any start up code or warmup code, it needs to run again because it is a different machine inside the Azure datacenter

### Scale Out

**So far, we saw a bunch of websites on only one instance** -> which can cause slowness but also doesn't offer a lot of reliability

**Let's scale out to more instances** -> Again, this is at the app service plan level, so it will affect all apps running on this plan

**Just drag it to two ->** the maximum comes with the app service plan -> Premium allows 20, Standard is 10 and Basic allows for 3.

This was a manual scale operation, and my app is now running in 2 instances

Go Into **Monitoring -> Process Explorer** and see the two instances running

**Auto Scale - Use CPU Percentage**

Explain dials and set from 65 to 90 - go to activity log and **show scaledown operation** (mail should arrive too) - there is some latency

What if we have more complicated rules for scaling??? **- schedule and performance** rules

Add rule based on **Memory** and go through the process

Explain we need to **add the scale down rule** as well

Explain limits on instances on profile

Change this to be the **weekend** profile only

Also explain what a **fixed date** could be (like a specific match)

### Deployment

**Run Install on Lab 2**Explan Deployment Credentials

#### FTP

* Get FTP hostname - just talk don't actually deploy
* Open FTP client and show how to connect - just drop one file with some html in and call it a day

#### WebDeploy

* Publish from Visual Studio using Web Deploy
* Show Cloud Explorer
* Open VSApp solution
* Publish Web App -> Choose App Service and go through the process
* Show we could create a new app service (look at the export feature to show it's an ARM template)
* Connect to VS-APP and publish
* Show the publish profile -> explain we could download it and just create a publish profile from this
* Explain it's incremental, so make a change and show Preview -> Publish and notice changes

#### Web Deploy LAB

* Mention we can also generate the webdeploy package
* Open lab folder and Edit script
* Notice the Publish package command let
* Explain we are using an old system and we should move to new process of deploying through template

#### Web Deploy + ARM

* [SHOW ARM TEMPLATE]
* Or, we can publish through ARM -> look in Speaker installWithArm template and explain the package needs to be accessible publicly

#### VS Publishing scripts and ARM Building tools

* Or, we can publish through ARM -> **look in Speaker installWithArm template** and explain the package needs to be accessible publicly
* Add new **Azure Resource Group project** -> choose **web app**
* Remove all unecessary components (app insights)
* Add **reference to vs project**
* **Add WebDeploy resource**
* Change website variable (change also in appinsights)
* use **VSApp as folder and package.zip as** **file**
* **Don’t run it in the interest of time**

#### GitHub – speaker only

* **Use AppToGitHub**
* GitHub **- login on github** and setup continuous deployment - that's cool
* Choose **Deployment Source on portal and point to GitHub**
* Go back **to Deployment Source** and see the deployment happening
* Change **content on page with Visual Studio Code and commit/push to GitHub**
* We should see **deployment happening again**
* Open **WebHooks** on repository and choose the hook to see the json payload (awesome)
* We can do this with VSTS also

#### Local Git

Do Local GIT Repository using Lab -> clone of the repositories (like the ASP.NET) and add the remote git as source on your local folder

### Deployment Slots

* Deployment Slots - use VS Site....
* Add new Deployment Slot - Staging (copy stuff which is awesome)
* It is another web app actually - its own url, its own config settings
* Use VS or script to Upload altered version of the app to the staging site (change a color)
* **Change Version parameter on Staging slot to 2.0**

#### Swap

* Choose Deployment slots and click **Swap**
* Swap with **Preview**
* **Look in the staging slot** to see what changed...notice the WhatAmISlotSetting changed because it is running the staging environment with production settings
* **Notice that production** site hasn't changed at all
* **Complete swap a**nd show changes...very cool
* Click Complete Swap

#### User Testing in Production - NEEDS TO BE DONE HERE

* Allow a small portion of your users to test your staging site
* Look at staging environment -> go to development tools and choose Testing in Production
* Select staging slot and ask for 50% of traffic to there
* Notice 50% coming to production
* Browse, F12, delete TPMix cookie and try again

#### Auto-Swap

* only available in non-production slots
* Application Settings "Auto-Swap: On" - production
* It's interesting to avoid cold start

#### Continuous Deployment

**Deploy the CD template immediately and explain after**

* Go to website -> show its empty
* Show staging website
* Swap them
* Turn on **AutoSwap** and show how that would be done in the template
* **Go to deployment options on staging**
* Go to GitHub and change index.html file and commit
* Show operations being done -> swap being completed and version reseting on Staging

#### Advanced Continuous Deployment

Show Continuous Integration Scenario with VSTS and follow the flow

# Monitoring

### Intro

Start by opening the email showing the server error alert

**Open Web App and** see that we had our webapp running for a while

Open Web app in Portal and look at about **20k requests and 249 http server errors** (last hour)

**Click on the graphic and look at data**

Default shows last hour of activity -> edit -> **show today -** we need to check those http server errors

**Configure to see 4xx errors**

Let's see if they are **404** - all of those (broken link somewhere)

This is across all the VMs running our web app

How to look into **instance specific** data????

Metrics per instance- **Apps vs App Service** (includes the entire service plan with all the apps that are running)

#### Metrics per Instance -Apps

This is **Live Data - we** can stop the updating in real time by clicking the button

We can select a **time period** and then go forward and backward

Look at 1 hours and we see the instances on each counter - we **can toggle them on or off**

**We're looking into the .NET Process** which applies to this app because it's asp.net

Collapse .NET and look at ASP.NET

-> **Requests/sec i**s interesting

-> the other one is **Requests in Application Queue**

Talk about **ASP.NET Request Queue**

**Choose Site Metrics** on Top

This is historical data

In this case these are toggle buttons (the instance buttons)

**See average response time and play around with metrics to look at the 4xx and errors only**

#### Metrics Per Instance - App Service Plans

Show that CPU on one instance is similar on both – but point out you could see differences

**Scroll down to see different sites and see who is causing problems** - only one site but if we had several we could get which app was causing this?

Scroll down more and look at **Http Statistics and Network statistics**

#### Monitor Live Traffic

We can see live traffic coming in and errors happening in real time - test should still be running

#### Using Alerts

Obviously we want to be able to be notified in case something is happening, or better yet, be about to happen because the chance that we are looking at the portal when a problem happens is actually slim

Show Alerts -> configure one

### Diagnostic Logs

So we are having some **404 errors and some server errors** -> we need to **understand which urls** can be causing the problem

#### Detailed Errors

Go To **Web App-> Monitoring -> Diagnostic logs**

Application Logs -> usually by code -> application exceptions and tracing information

Web Server Logs both to storage or File System

Detailed Error Messages (TURNED ON BEFORE) - you get html file for every error - always on file system

Failed Request Tracing is very detailed analysis of how a request was handled - always on file system

You can get them through FTP but there are better ways...**for now let's use FTP**

Go to **logs and open Detailed Errors** and choose a html file -> we can see which url it is for the 500 errors

#### Application Logging

**Stop Test**

One of the tools we can use is **Application Events** (under Monitoring)

Off by default, requires **Application Logging** to be turned on

See events -> see in RealTime using **Log Stream**

Mention you can get these with **powershell** as well and **Azure CLI (**including some **filtering**)

#### Web Server Logs

Web App with a staging deployment slot and we **setup Testing in Production to** send 20% of traffic to staging site

I have **a new build in staging**

Go **into staging blade and click Monitor chart and change to past week**....look at server errors

Go look at **Metrics per Instance** -> and **choose Site Metrics -** change timeframe and turn OFF http 2xx...

See the **spike in errors, turn it off and see a spike to Avg Response Time**

Go to **Diagnostic Logs and show that web server logging** is on

Now go to **Advanced Tools and click Go, then choose Tools -> Diagnostic Dump**

Once we open the zip file, we see the **LogFiles** directory and we see the **http directory** -> **Raw Logs** and we see two instances

Open it with a text editor -> but actually open with Excel

Look at error **500x,** look at 404 as well and sort **by time taken**

### Troubleshooting

Remote Debugging with VS is an option, but not going to show that today

#### Failed Request Tracing

Turn it on the staging slot

Go to page that causes the 500 error

Log in through FTP and look at LogFiles -> W3SVC.... -> get the xml file

Navigate to 500 error page and look at the new log generated. Show error, time taken, etc...

Do the slow freb stuff, change path to Categories on the rule and access slow running page

**Point out that you still use the same techniques to analyze an app as you would do on IIS**

#### Diagnostic as a Service

On Staging slot

Diagnostic as a service (under Support & Troubleshooting)

By default is set to ASP.NET (explain options)

Click "Run" - it's pulling data for me, all the event viewer logs, the http logs and a memory dump of the process

Once it finishes each collection it starts analyzing

How to get to different options in DaaS

**Advanced Tools -> Open**

Then **add /daas** to the end of the url

**Click schedule -> Live vs schedule...go through the wizard and show additional options**

Look at the Analysis results on **Event Viewer** Logs -> this is the usual problem we already know about

We scroll down and we find another one related to remote debugging -> we can simulate that by adding two instances of VS and trying to remote debug both

Go to DaaG and get the analysis for **HTTP Logs**

Look at queries for **LogParser**

Go through the queries

**Memory Dump Files**

----------------------------------------

NEED TO RUN BEFORE, DURING SLOW TEST TO HAVE MANY BLOCKED THREADS

Just open analysis, look at the top blocked threads and open one thread detailed

#### AutoHeal

show you a tool that you can use for taking automatic actions based on symptoms

Advanced Tools (or just drop scm)

Tools -> Support -> Mitigate

Turn on AutoHeal

Add Max Request rules -> Actions -> interesting is the CustomAction

Add Status Code rule

Memoy Rule

Show that this is just changing web.config through Visual Studio online Extension

**Talk about Companion App and the waking up at 3am**

#### ADVANCED Tools

Choose Advanced Tools and go to the website (used to be called Kudu)

We can dig in to a lot of information about our web app and access some tools that are not available anywhere.

**Environment**

Click Environment and show all the environment variables and AppSettings, ConnectionStrings, etc, etc and then HTTP Headers, Server Variables

Often useful if you use Environment Variables in your app as this can differ when you deploy to the cloud

**Debug Console -> CMD**

Go through the little buttons

Let's go back to website and download logfiles directory (it's the same of going into Tools -> Diagnostic Dump)

**Open Debug Console -> Powershell**

**get-process**

Create a new directory

Send a zip file in there

**Click -> Process Explorer**

show we have two processes (one is actually running scm)

we can see CPU time, working set, private memory, thread count

Click properties on the web app w3wp -> we can download a Dump but DaaS is better

#### Use Site Extensions

Advanced Tools -> Site Extensions

Application Insights is already there (by default)

Go to Gallery -> some are for management, others are for runtime

Let's add this IIS Manager