Djocumenting Bottle and Django to Azure

Thursday, January 21, 2016 4:26 PM

Overview
This tutorial tracks a combination of N pieces of technology ((N+1) if you count the Python programming language) into web-based data services with some other important features included along the way. The original motivation is the work of Parker MacCready on the LiveOcean project and, following suit, Anthony Arendt creating Ice2Ocean from the same template. The latter can be viewed as in-progress work at http://ice2oceans.azurewebsites.net.

Dramatis Personae
- Visual Studio: The Microsoft IDE
- Azure: The Microsoft public cloud
- Python: A Programming Language
- Django: A Web Framework built on Python (complex / powerful) and MVC (see below)
- Bottle: A Web Framework built on Python (simple / powerful) and MVC (see below)
- GitHub: A software repository where we learn how to separate public from private information.

What is MVC?
M is for Model
V is for View
C is for Controller
Read this: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
Now you know what MVC is.

Tactics
The tactical objective of this tutorial is to give a circa January 2016 How To for building out web sites under the **Django** and **Bottle** web frameworks, which are in turn built on the MVC software design philosophy with emphasis on ***separation of concerns***. Briefly this separation into three parts--models, views and controls--is an organizational technique that helps you build complex systems without building correspondingly complex ***bugs***. It's worth knowing about if your programming work goes beyond basic Load-Compute-Save operations on data; and that is particularly the case if you are working in a collaborative and/or web application-focused world.

So we have Django and Bottle as starting points which help us get things done quickly and reliably. The next part is the IDE which in this case is **Visual Studio** (2015 and 2013 are current versions; 2015 is what I'm using.) This not only enables us to put together a Django web app but also helps view it (on the development machine (localhost)) and helps debug it. In fact there are two distinct build paradigms in Visual Studio: Debug and Release. Debug may run slower but it will also 'watch itself' better and help us fix problems. So that is our IDE.

The next piece of technology is the **Azure** cloud platform operated by Microsoft. It includes a web-based portal for creating and configuring resources like web sites; so we can work on the portal to create cloud space for our web application. I try to delineate the jargon (or Djargon) in this manner so we can agree on the terms we use and their meaning.

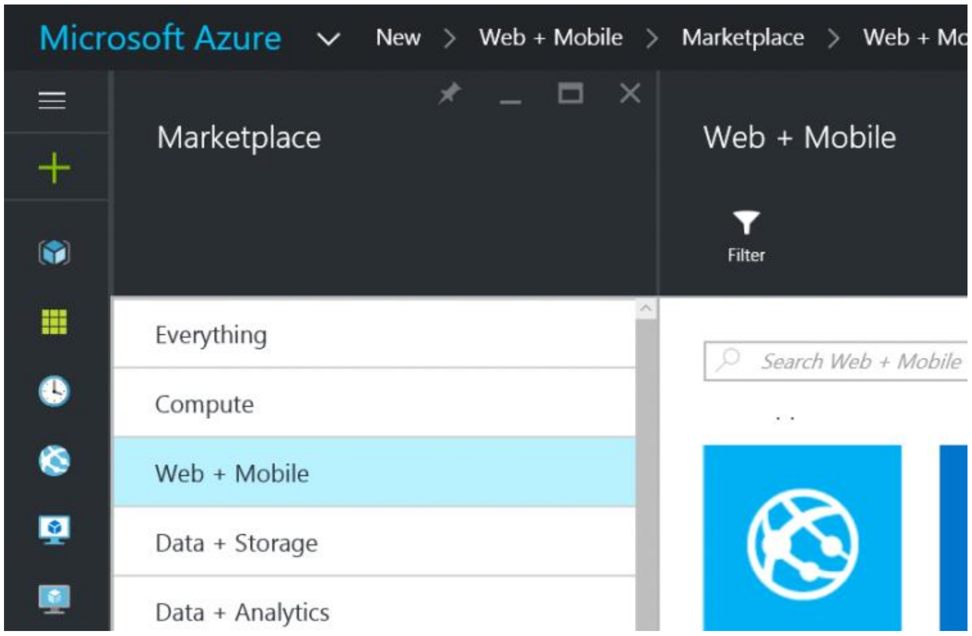
Next is **GitHub**, the code repository. To connect Visual Studio to GitHub is easy… provided you know not to initialize the repo folder on GitHub with anything. Leave it empty and Visual Studio is happy to fill it up. But the really important ***Gotcha*** is to then follow through with a .yaml file that contains private keys. We will use the gitignore feature to make sure we don't accidentally publish any private keys when we Commit code to GitHub. Maybe this will involve YAML (.yaml or .yml extension; see https://en.wikipedia.org/wiki/YAML)

Conventions
This djocumentation uses a few idiosyncratic conventions of mine; so best commit these to memory.

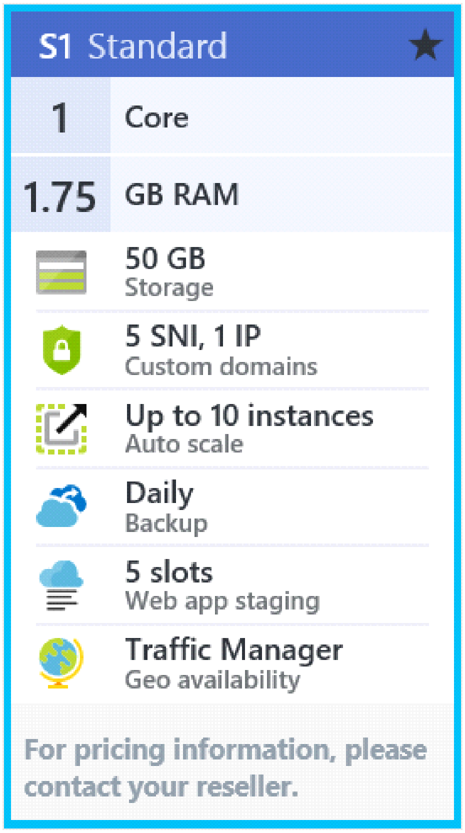
- **Dj** is used to give incorrect spellings for common words entirely for the sake of my own amusement. This document covers the creation of not One but Four web applications, for example, which are named respectively:
    - Djak
    - Djunior
    - Djazz
    - Djargon
- The all-lowercase name '**kilroy**' is used to flag places where the sidewalk ends; where it is unclear what is going on or where errors are occurring or where problems were found and fixed. Searching on kilroy in my text (code especially) is a good way to review what I think is missing in the project.
- **GOTCHA-N** is a numbering scheme for particularly vicious traps; things that should be straightforward and easy but mysteriously fail. N is the N'th GOTCHA but I may fail to adhere very well to the numbering. The GOTCHA flag is the important part.

GOTCHA 1
Sometimes we wind up with more than one identity and it can be difficult to know "who am I logged in as?" Mysterious even. So my Azure account is under my administrative ID but if I try to log in to the Azure portal by providing that ID: I end up logged in under a different ID. The browser is apparently being overly helpful. The fix: I use in private browsing in which case I am permitted to log in as myself-administrator. Kilroy was here.

In the Azure portal (portal.azure.com) I created a 'djak' resource group. I then found Web + Mobile and will add a new Web App that I hope to name djak.azurewebsites.net.



So the portal suggests I create an App Service Plan first; so I do so choosing the Standard 1 tier:


S1 Standard

1 Core

1.75 GB RAM

50 GB Storage

5 SNI, 1 IP Custom domains

Up to 10 instances Auto scale

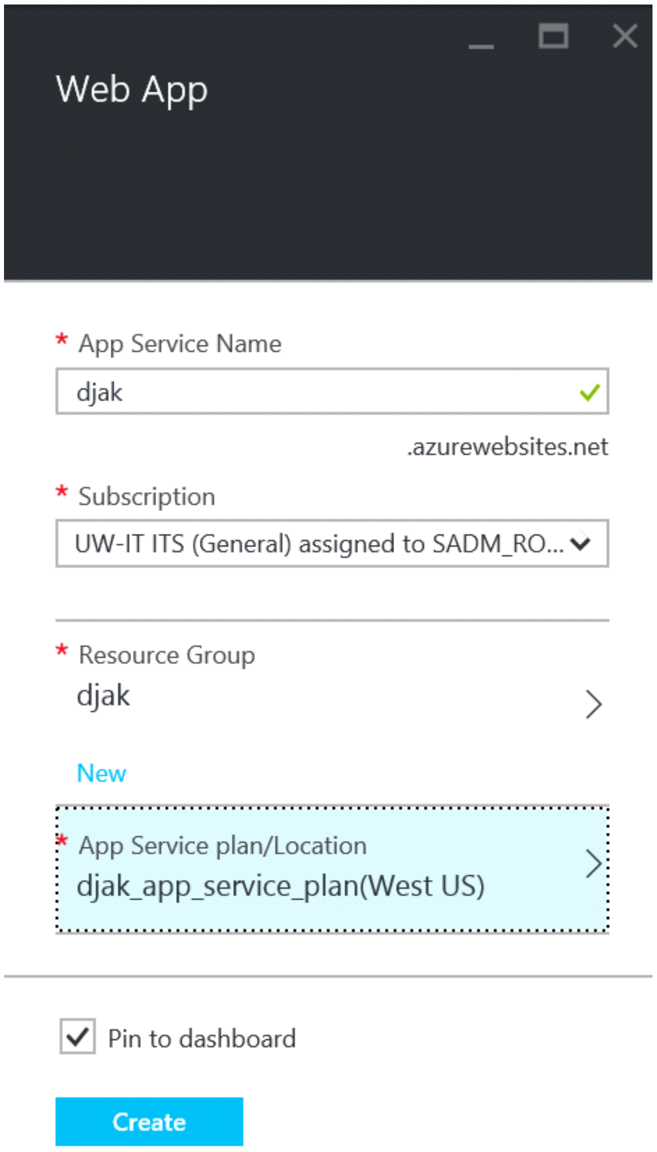
Daily Backup

5 slots Web app staging

Traffic Manager Geo availability

For pricing information, please contact your reseller.

So collapsing all the blades (things that fold out to the right) I hit Create on this configuration:


Web App

* App Service Name

djak

.azurewebsites.net

* Subscription

UW-IT ITS (General) assigned to SADM_RO...

* Resource Group

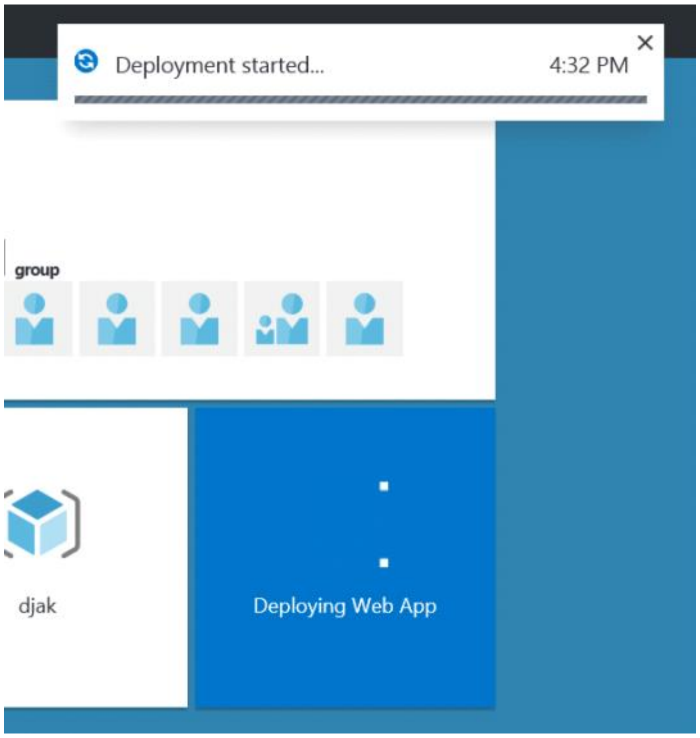
djak

* App Service plan/Location

djak_app_service_plan(West US)

☑ Pin to dashboard

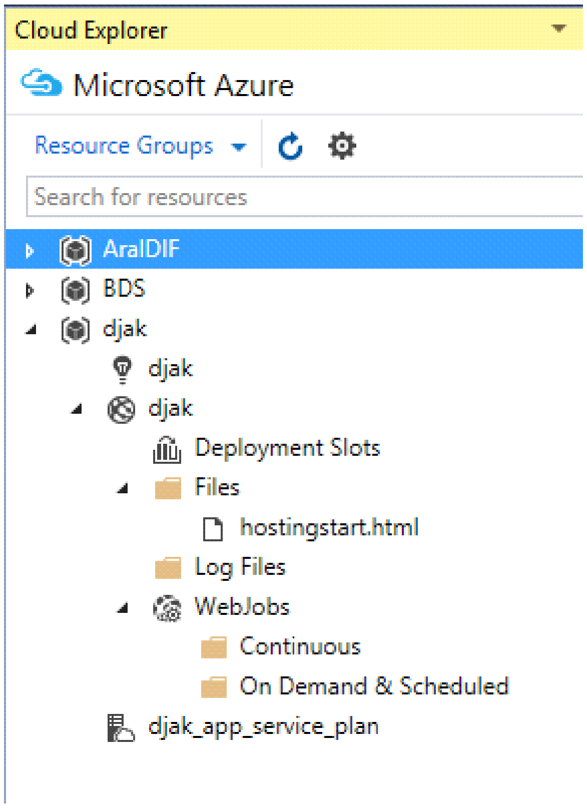
I get a deployment activity page:


Deployment started...

4:32 PM

This runs for five minutes or so.

When it is done I can go to http://djak.azurewebsites.net and there is a nice splash page. But that doesn't mean Djak just yet. Let's go back to VS and see how it views my Azure subscription:


Cloud Explorer

Microsoft Azure

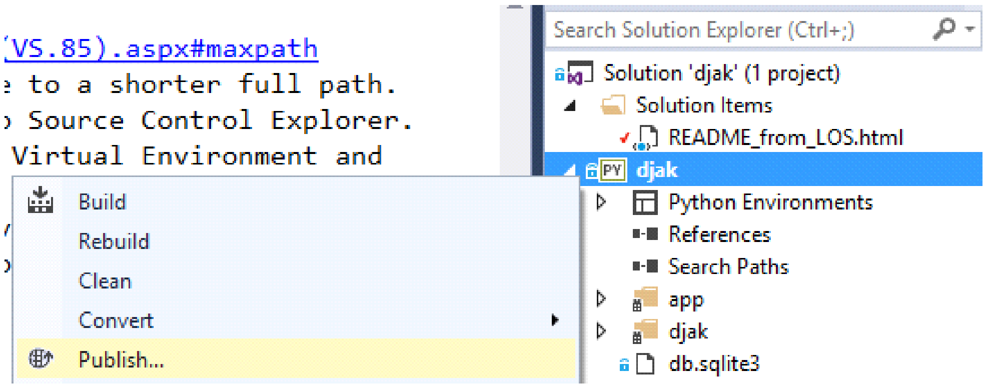
Resource Groups

Search for resources

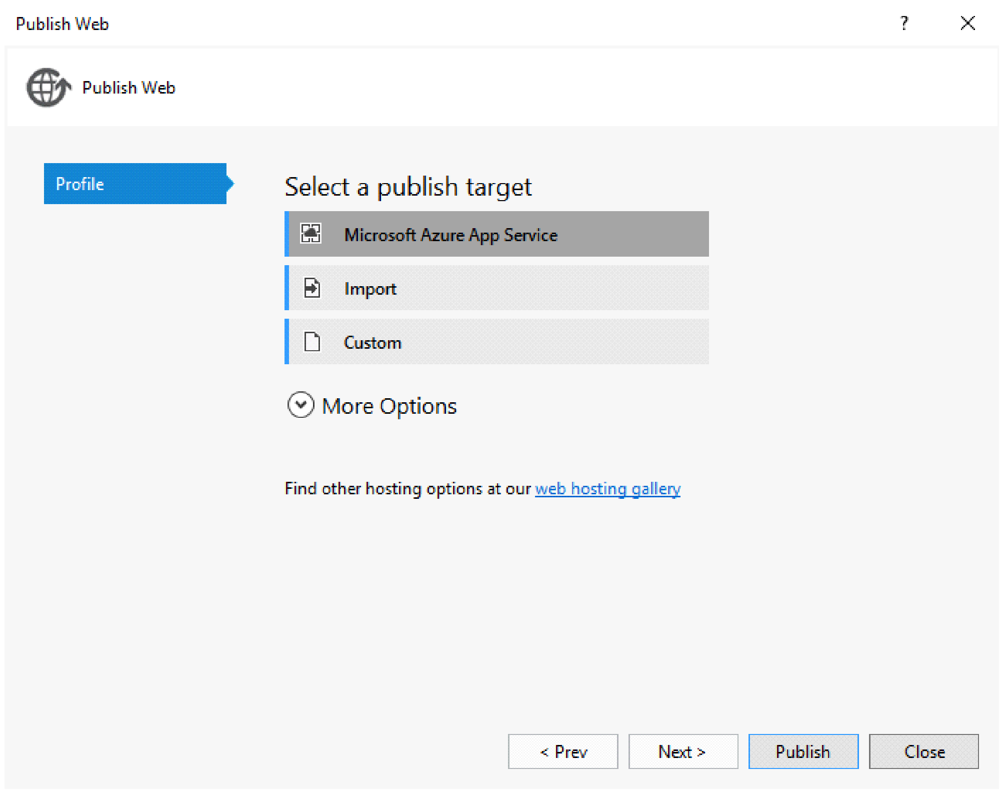
```
▸ AralDIF
▸ BDS
◢ djak
      djak
   ◢ djak
         Deployment Slots
      ◢ Files
            hostingstart.html
         Log Files
      ◢ WebJobs
            Continuous
            On Demand & Scheduled
      djak_app_service_plan
```

Yes! Djak is there and ready for me to publish to. How hard will this turn out to be??? I don't know!

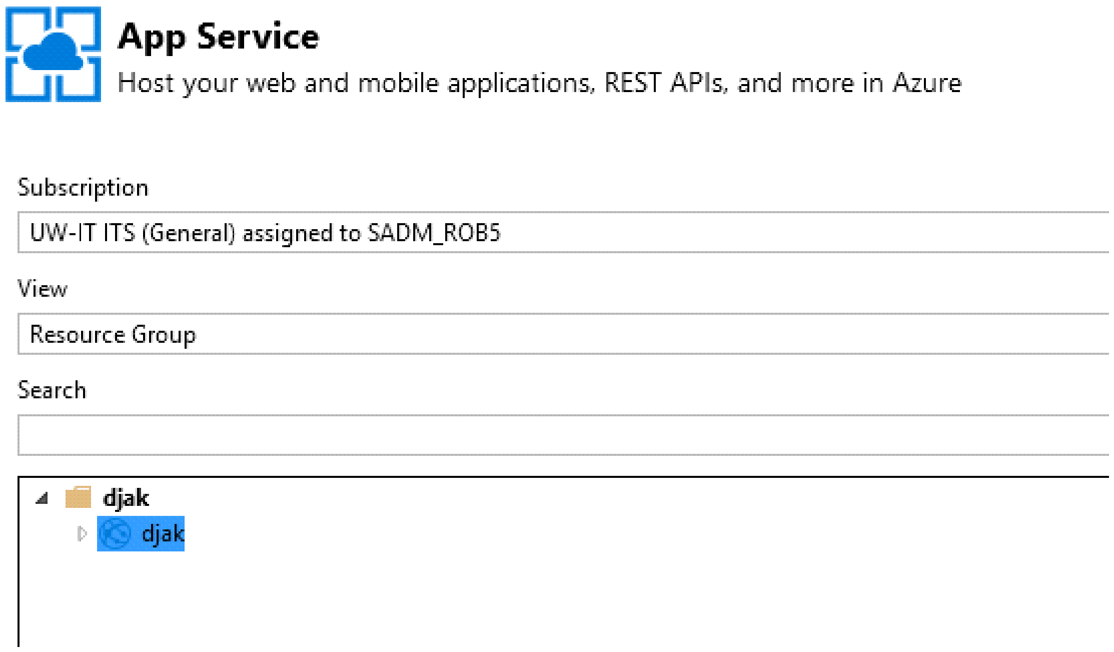
Right click on the Djak project in Solution Explorer:



And sure enough we get the option to go to Azure:



Now we hit an App Service page that seems to have the right stuff… on the App Service pop-up the Ok is greyed out until I select the djak Web App within the djak folder. So here we go:



App Service

Host your web and mobile applications, REST APIs, and more in Azure

Subscription: UW-IT ITS (General) assigned to SADM_ROB5

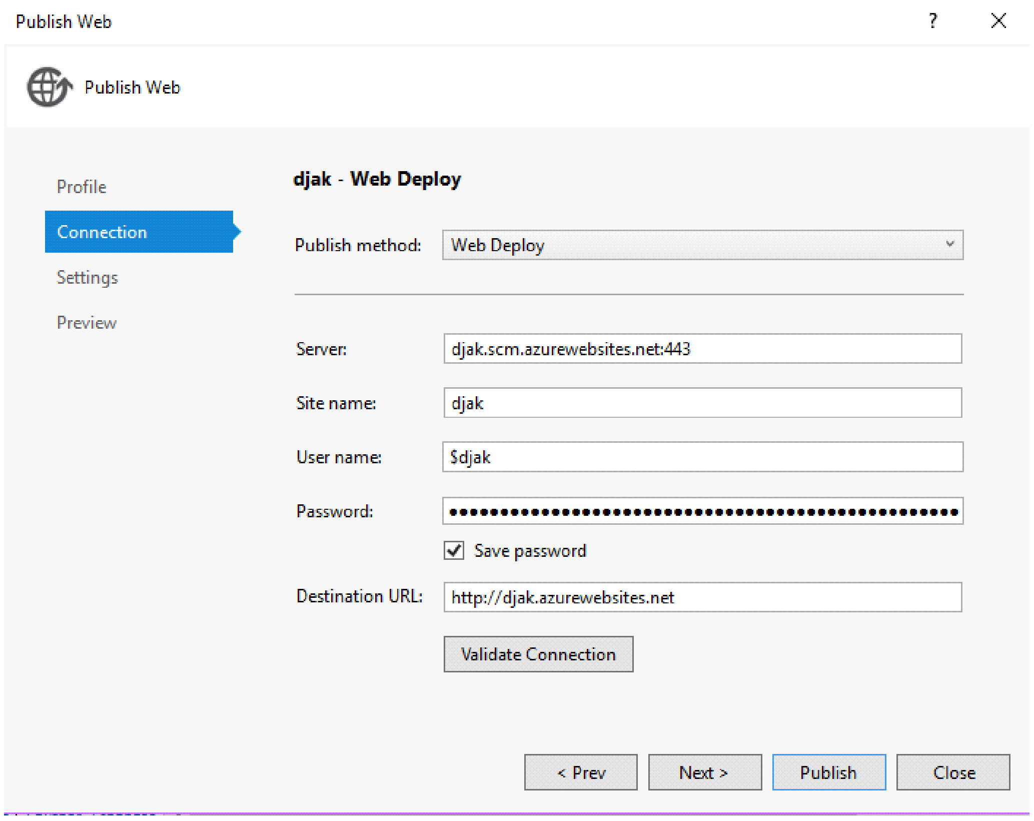
View: Resource Group

Search:

- djak
  - djak

Notice that this dialog also has a New... button which is the Visual Studio way of allocating new resources in Azure; but I'm skipping this here and just running with the web app as shown.

From here we get some "automatic population" of the subsequent form:



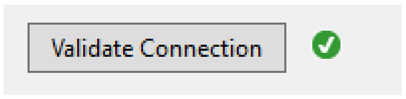
Publish Web

djak - Web Deploy

Publish method:	Web Deploy
Server:	djak.scm.azurewebsites.net:443
Site name:	djak
User name:	$djak
Password:	••
	☑ Save password
Destination URL:	http://djak.azurewebsites.net

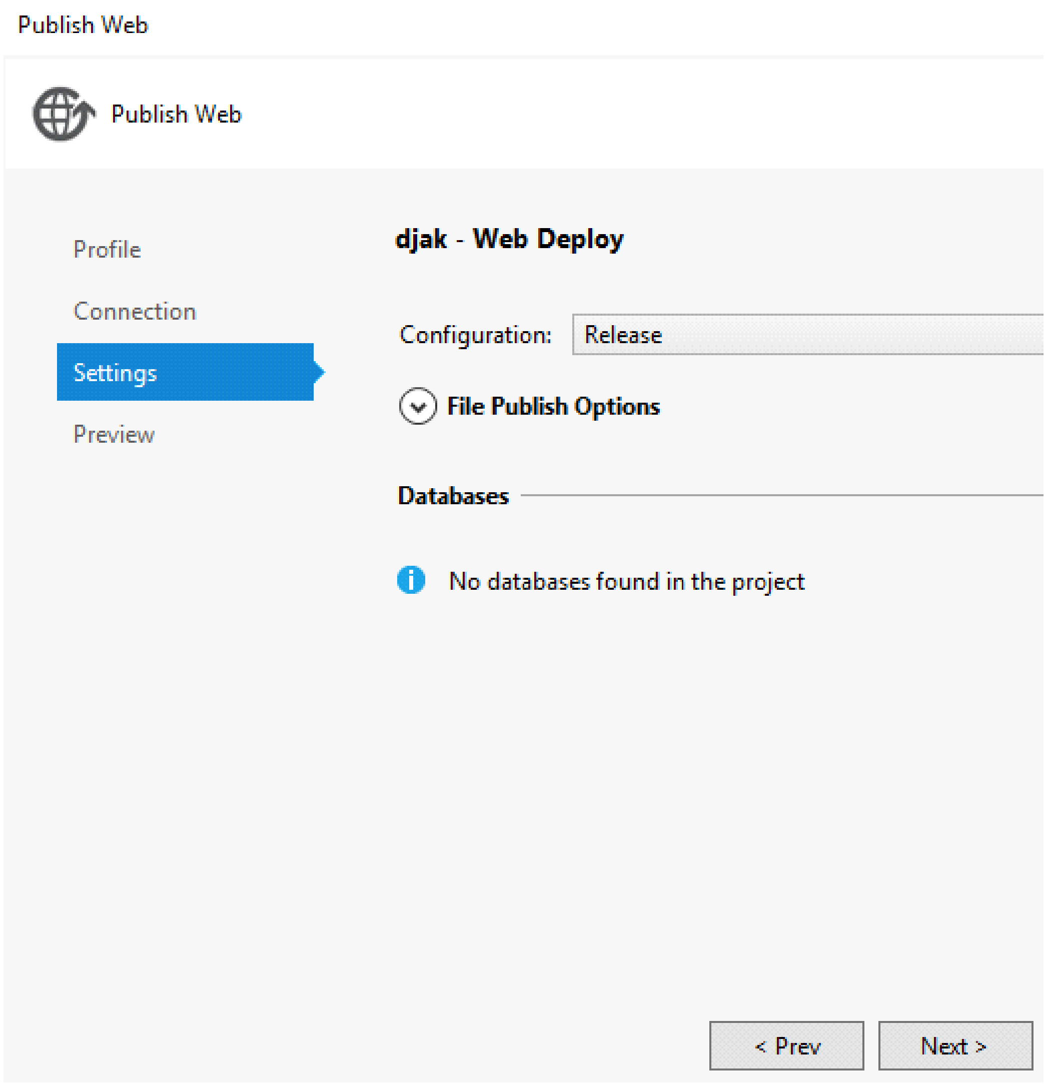
Going forward my only concern is that I have not designated a super user (which was, I believe, supposed to be happening in the DB Synch step... hmmm....)

I click Validate Connection and in a few seconds:



Notice that at upper left there are four steps here: Profile, Connection, Settings and Preview. To be thorough we should step through all of them using the Next> button at lower right... but in principle one might cut to the chase by clicking Publish.

Interestingly on the Settings page we find 'No databases found in the project' and that's probably not good; so To Be Fixed. But let's keep rolling with Next> Incidentally I am fine with a Release (rather than a Debug) Configuration; but Debug is worth coming back to.



djak - Web Deploy

Configuration: Release

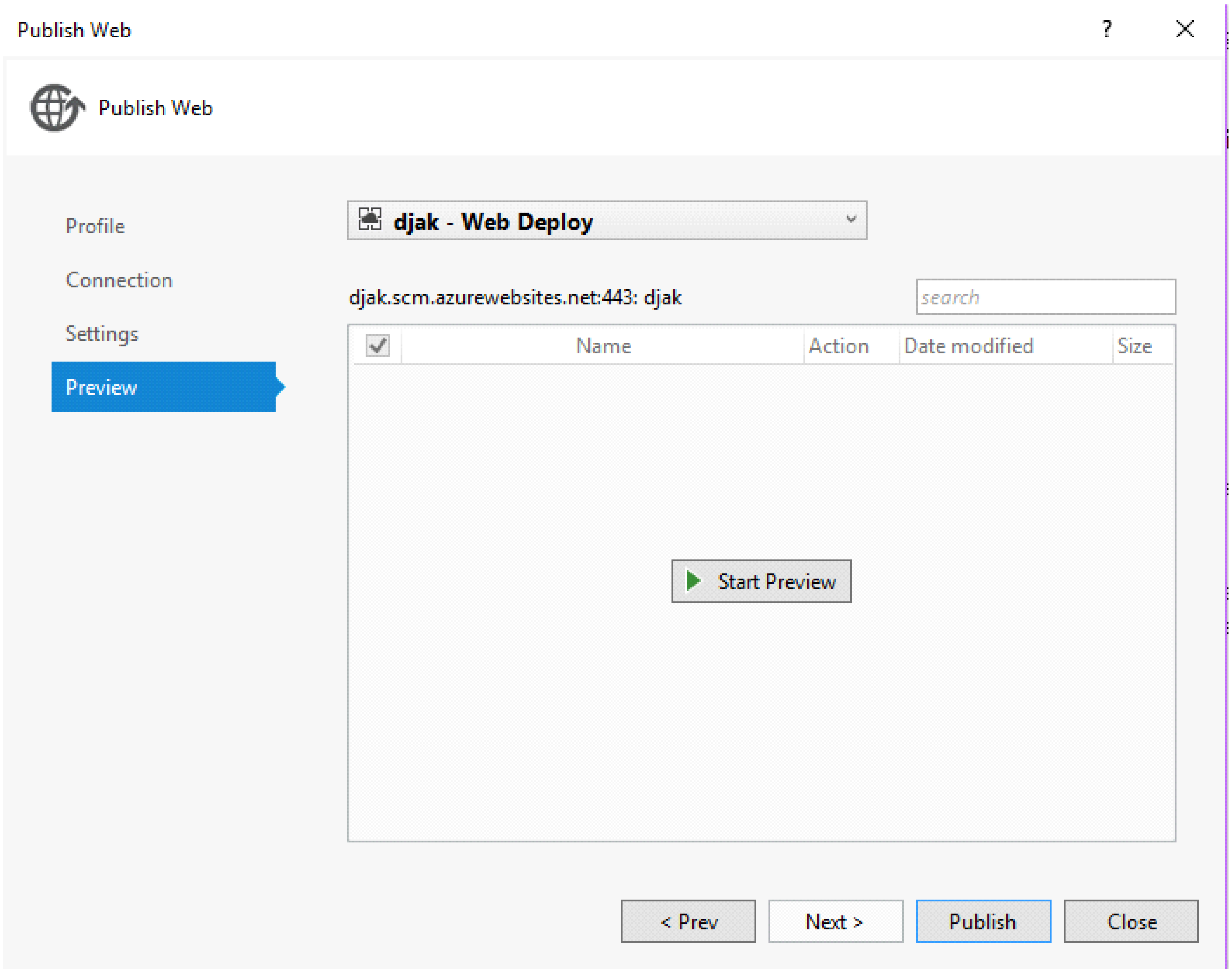
File Publish Options

Databases

No databases found in the project

Incidentally you can expand File Publish Options and click on Remove additional files at destination to clean up the pre-populated html that Azure puts in as a placeholder.

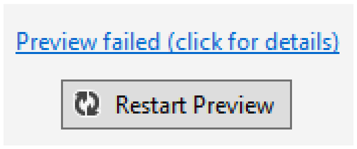
Now here we are on the Preview page:



So let's go slow and Start Preview...

This gives a little 'busy' message:

And I believe this means that it is differencing the existing web site content (what we automatically got from the Azure portal) and what we are trying to push up to that site. Did the Preview work? No:

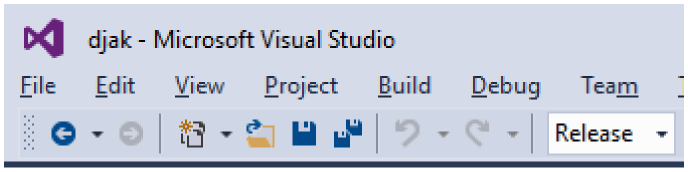


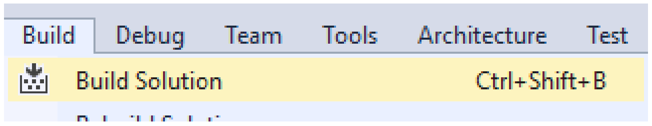
The details are:

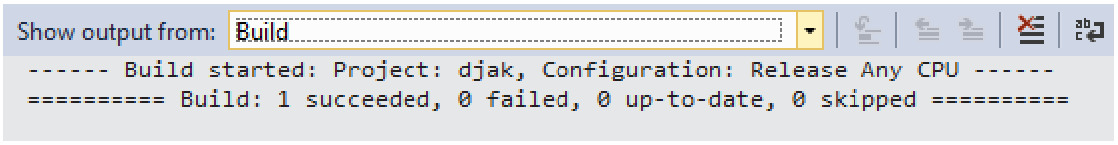
The "exists" function only accepts a scalar value, but its argument "@(_MSDeploySourceManifest->'%(RootDir)%(Directory)')" evaluates to "C:\Users\fatla_000\Documents\Visual Studio

2015\Projects\djak\djak\obj\Release\Package\;C:\Users\fatla_000\Documents\Visual Studio 2015\Projects\djak\djak\obj\Release\Package\" which is not a scalar value.

Notice that above I said 'Release' version but I have clearly only built the Debug version. So let's rebuild as Release in Visual Studio and try this again! (A note from subsequent tests: While this is probably correct it sometimes works to simply click on Restart Preview where the second time it works fine.)

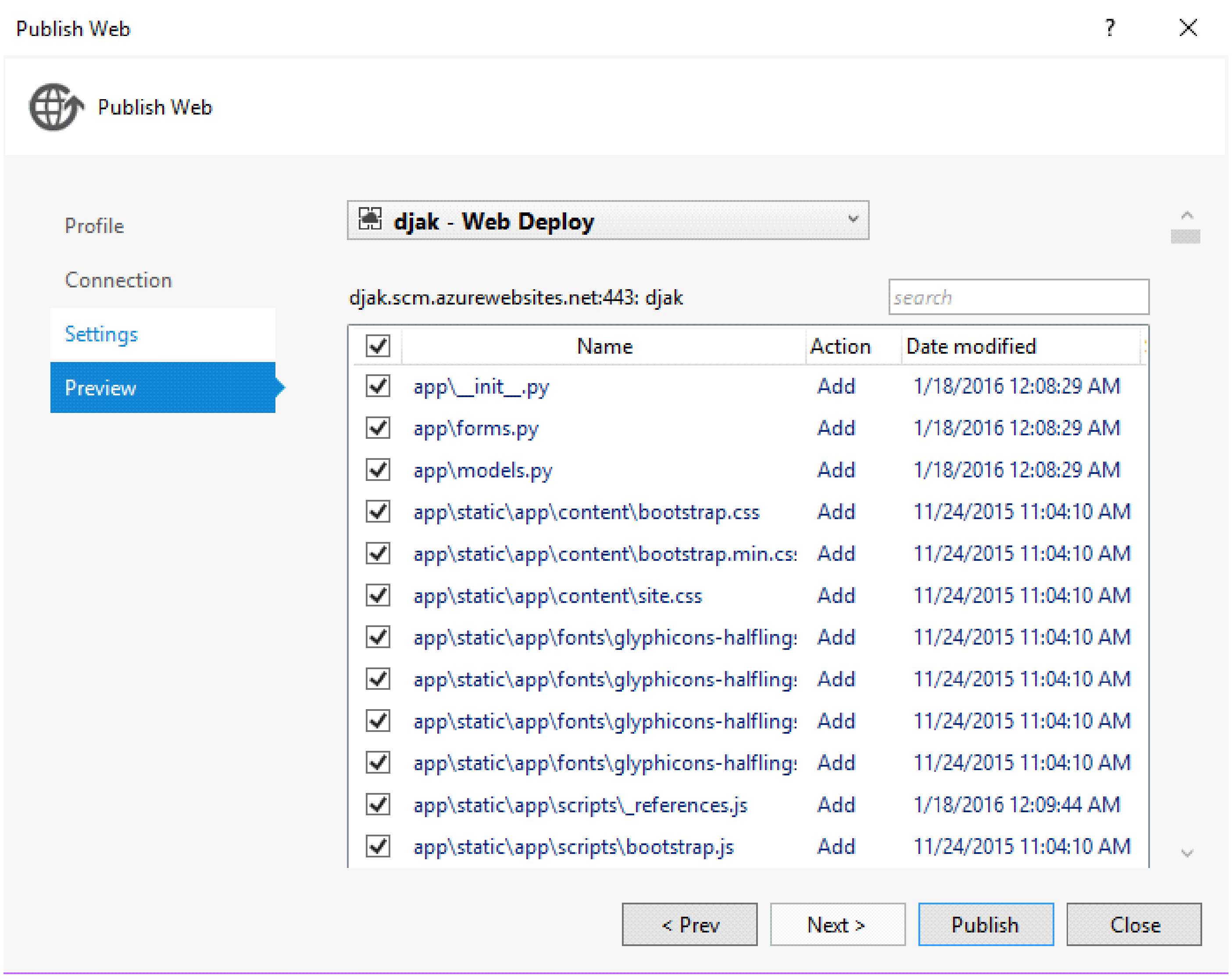





Show output from: Build

```
------ Build started: Project: djak, Configuration: Release Any CPU ------
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

And now try Publish again… and sure enough this time we're ok with a lot of updates to make:



Publish Web

Profile
Connection
Settings
Preview

djak - Web Deploy

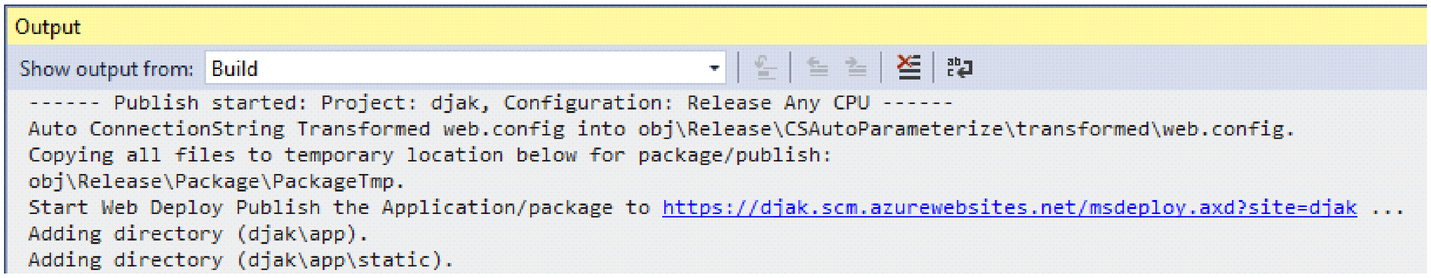
djak.scm.azurewebsites.net:443: djak

✓	Name	Action	Date modified
✓	app__init__.py	Add	1/18/2016 12:08:29 AM
✓	app\forms.py	Add	1/18/2016 12:08:29 AM
✓	app\models.py	Add	1/18/2016 12:08:29 AM
✓	app\static\app\content\bootstrap.css	Add	11/24/2015 11:04:10 AM
✓	app\static\app\content\bootstrap.min.cs	Add	11/24/2015 11:04:10 AM
✓	app\static\app\content\site.css	Add	11/24/2015 11:04:10 AM
✓	app\static\app\fonts\glyphicons-halfling	Add	11/24/2015 11:04:10 AM
✓	app\static\app\fonts\glyphicons-halfling	Add	11/24/2015 11:04:10 AM
✓	app\static\app\fonts\glyphicons-halfling	Add	11/24/2015 11:04:10 AM
✓	app\static\app\fonts\glyphicons-halfling	Add	11/24/2015 11:04:10 AM
✓	app\static\app\scripts_references.js	Add	1/18/2016 12:09:44 AM
✓	app\static\app\scripts\bootstrap.js	Add	11/24/2015 11:04:10 AM

< Prev | Next > | Publish | Close

Hmmm... I wonder if the database not there thing happens again now that I have the correct build (Release, not Debug)

Nope; still no database. But I hit Publish and we are off to the races:



Output

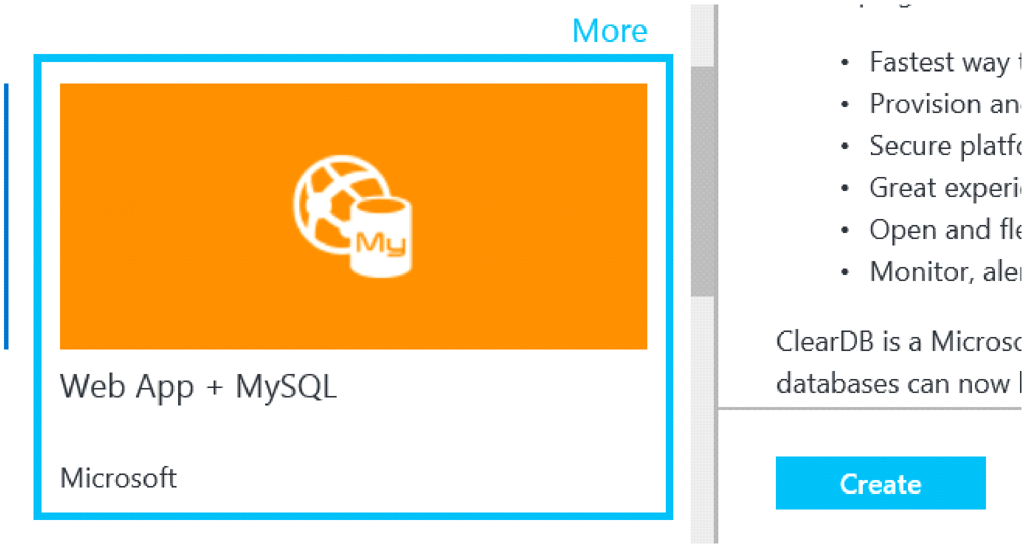
Show output from: Build

```
------ Publish started: Project: djak, Configuration: Release Any CPU ------
Auto ConnectionString Transformed web.config into obj\Release\CSAutoParameterize\transformed\web.config.
Copying all files to temporary location below for package/publish:
obj\Release\Package\PackageTmp.
Start Web Deploy Publish the Application/package to https://djak.scm.azurewebsites.net/msdeploy.axd?site=djak ...
Adding directory (djak\app).
Adding directory (djak\app\static).
```

Ok... great. Go get some coffee. Since this has worked so nicely I decide that I want to go and grab the other three URLs for my other three websites: Multo pronto. Here are there intended purposes:

- Djak is supposed to be a functional recreation of the LiveOcean functionality and will be built out to include a CDN fast-data-delivery component.
- Djunior is also a Django instance that is intended to cooperate with Djak in a web app - to - web app manner.
- Djazz is a third website intended to explore the medium... ok details TBD.
- Djargon is a documentation website... maybe I'll build it with Bottle or Flask though.

Ok let's put in some screencaps from creating Djazz. This time I'm including a MySQL database. This is still in the djak resource group; so from the Azure portal website:

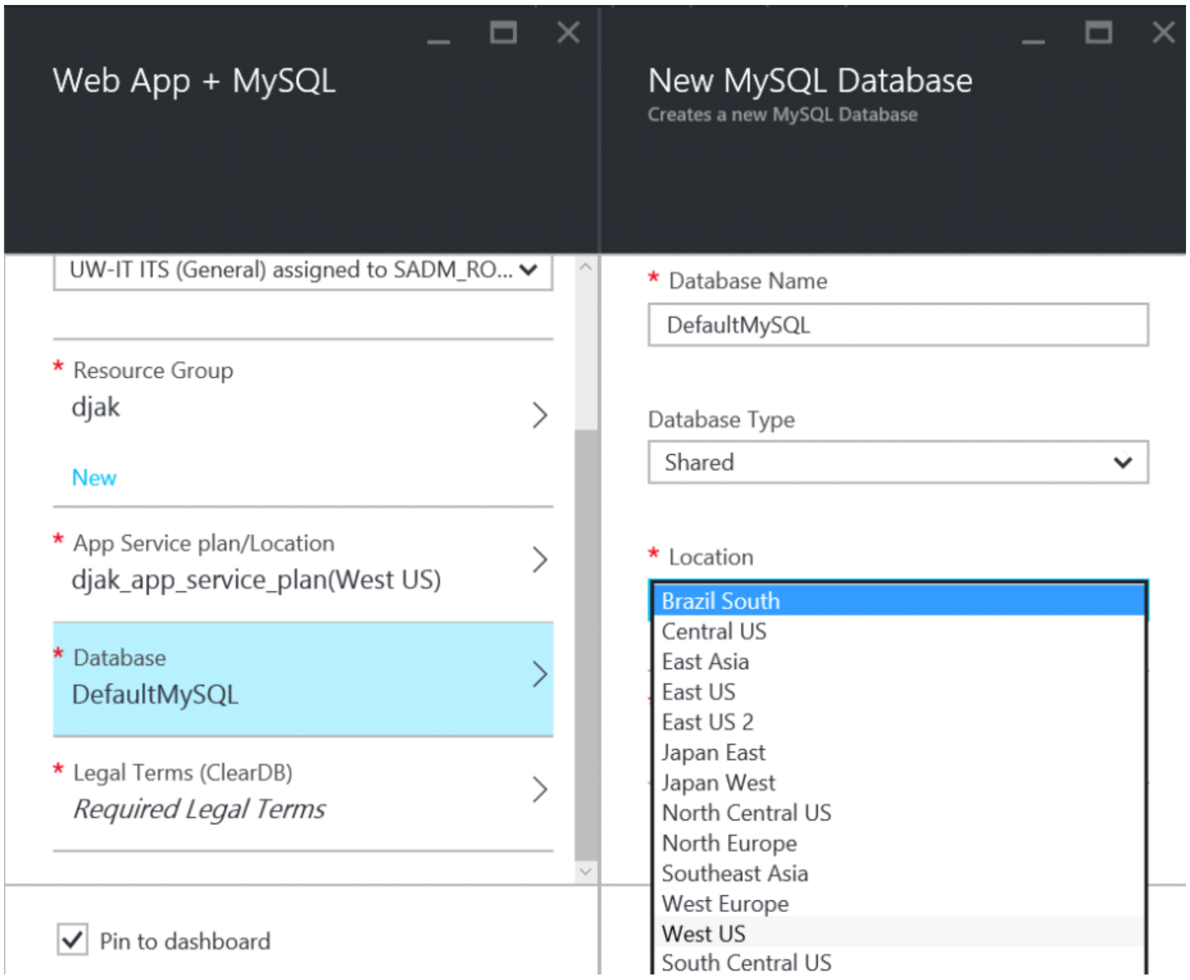

More

Web App + MySQL

Microsoft

- Fastest way t
- Provision an
- Secure platfo
- Great experi
- Open and fle
- Monitor, ale

ClearDB is a Microsc
databases can now l


Web App + MySQL

UW-IT ITS (General) assigned to SADM_RO...

* Resource Group
djak

New

* App Service plan/Location
djak_app_service_plan(West US)

* Database
DefaultMySQL

* Legal Terms (ClearDB)
Required Legal Terms

New MySQL Database
Creates a new MySQL Database

* Database Name
DefaultMySQL

Database Type
Shared

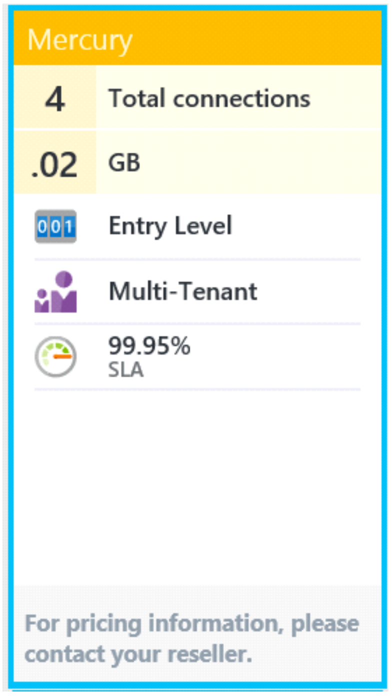
* Location

Brazil South
Central US
East Asia
East US
East US 2
Japan East
Japan West
North Central US
North Europe
Southeast Asia
West Europe
West US
South Central US

Notice that it wants to put my MySQL database in Brazil for some reason; so I choose West US instead.

GOTCHA-2 *When doing provisioning on the Azure portal plan to take some extra time looking into all the defaults. Do they make sense? Are the resources as co-located as possible? It seems like an obvious thing but this might a be one of the rough edges of the cloud: Obvious things don't just always work that way for now.*

Under the philosophy of 'we can always upgrade' I'm going to go for a light DB instance:



Mercury

4 Total connections

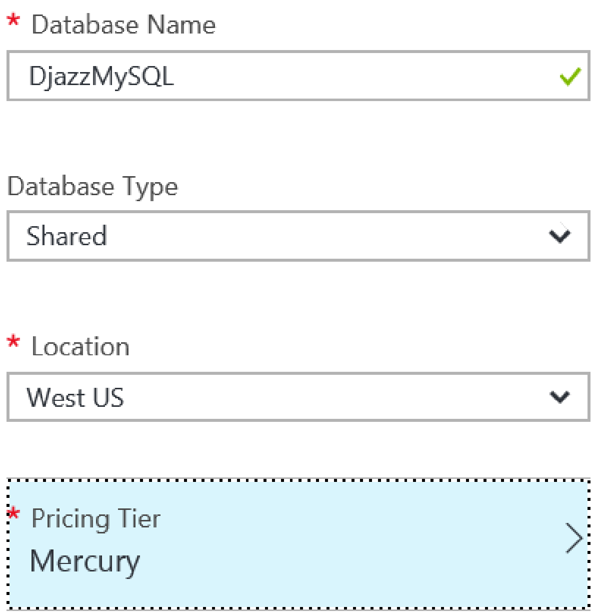
.02 GB

Entry Level

Multi-Tenant

99.95% SLA

For pricing information, please contact your reseller.



* Database Name

DjazzMySQL

Database Type

Shared

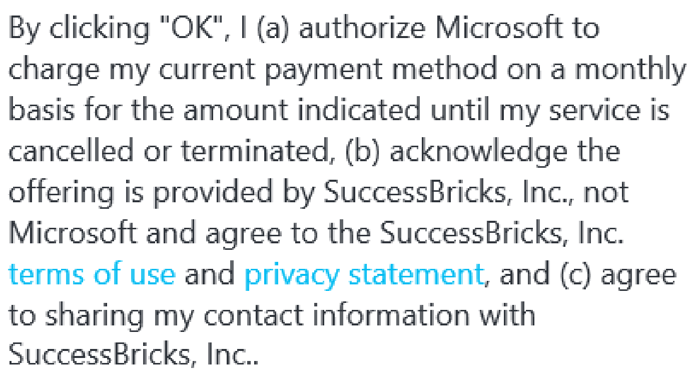
* Location

West US

* Pricing Tier

Mercury

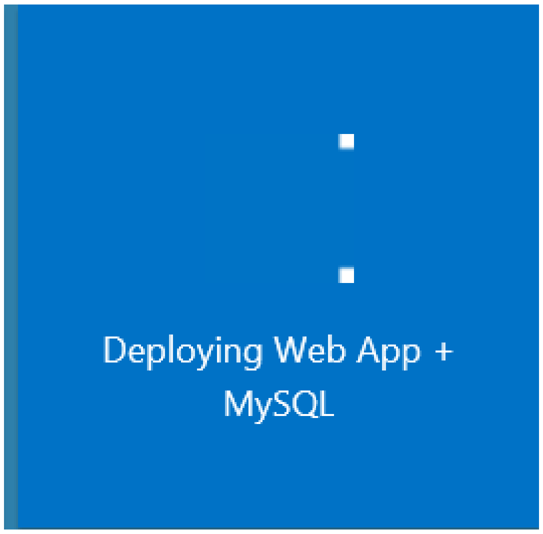
I will put this new web app on the djak_app_service_plan; i.e. I am adding to an existing service plan rather than creating a new one. Furthermore before I can Create the new web app I need to sign off on this bit:



By clicking "OK", I (a) authorize Microsoft to charge my current payment method on a monthly basis for the amount indicated until my service is cancelled or terminated, (b) acknowledge the offering is provided by SuccessBricks, Inc., not Microsoft and agree to the SuccessBricks, Inc. terms of use and privacy statement, and (c) agree to sharing my contact information with SuccessBricks, Inc..

So apparently I should expect some email from SuccessBricks. Such is life.

And off we go…

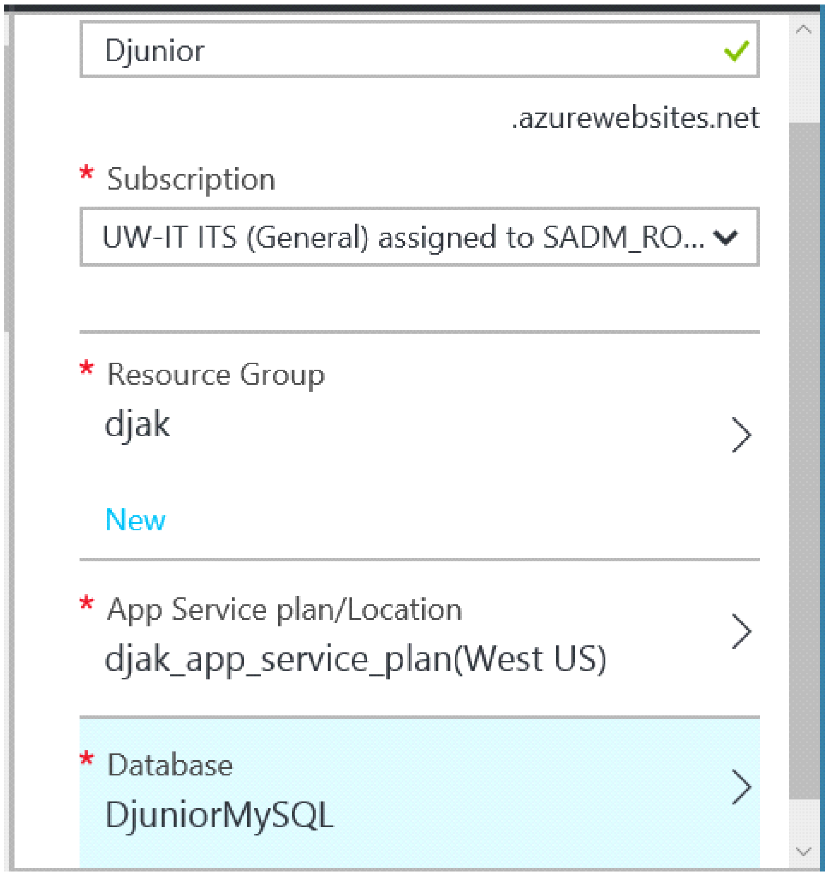


Deploying Web App + MySQL

And now it exists. Hurray!

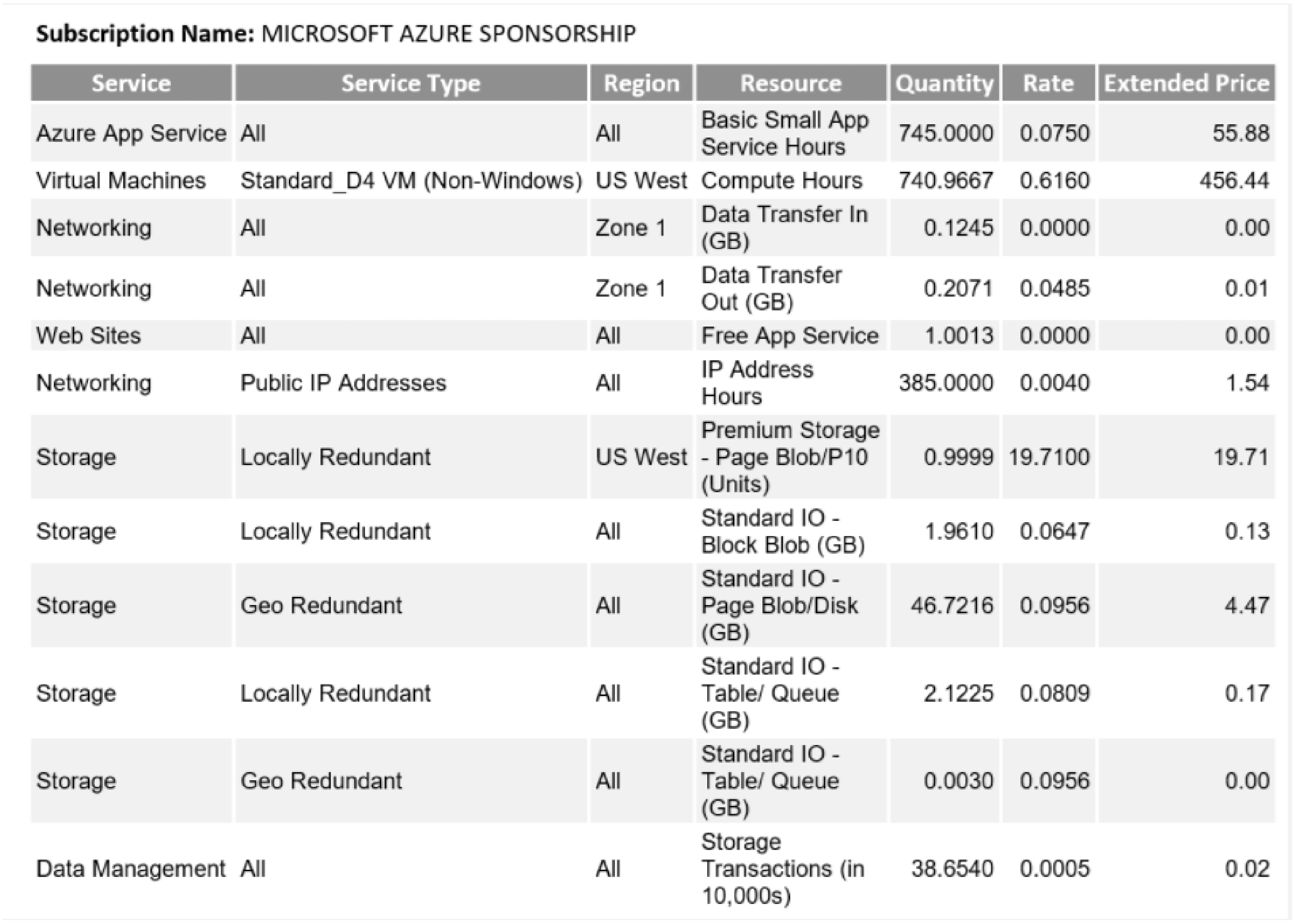
Let's do Djunior and Djargon before we forget. They will also include minimal MySQL databases.

Here is for Djunior:


Djunior
.azurewebsites.net
* Subscription
UW-IT ITS (General) assigned to SADM_RO...
* Resource Group
djak
New
* App Service plan/Location
djak_app_service_plan(West US)
* Database
DjuniorMySQL

And my only problem here is that I forgot to use lower-case d for djunior.

How much does this cost??? I have no idea. But the trick is to get a monthly billing statement like this one:

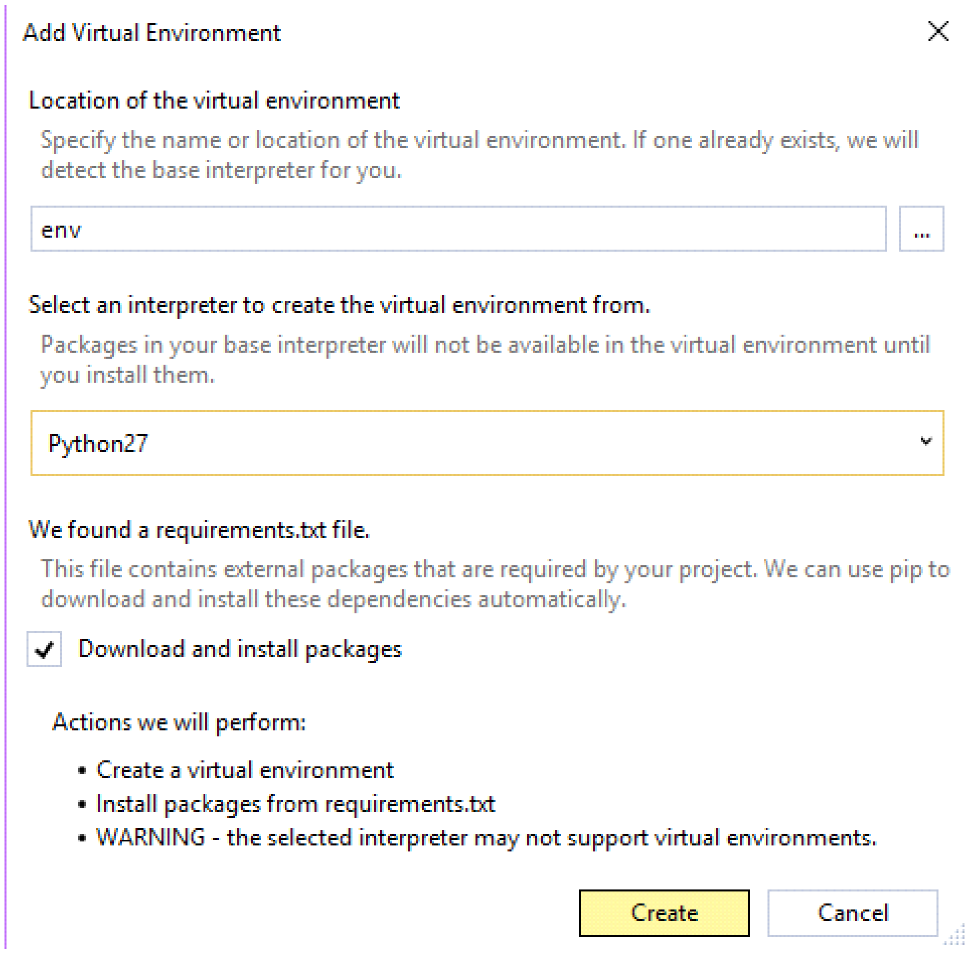

Subscription Name: MICROSOFT AZURE SPONSORSHIP

Service	Service Type	Region	Resource	Quantity	Rate	Extended Price
Azure App Service	All	All	Basic Small App Service Hours	745.0000	0.0750	55.88
Virtual Machines	Standard_D4 VM (Non-Windows)	US West	Compute Hours	740.9667	0.6160	456.44
Networking	All	Zone 1	Data Transfer In (GB)	0.1245	0.0000	0.00
Networking	All	Zone 1	Data Transfer Out (GB)	0.2071	0.0485	0.01
Web Sites	All	All	Free App Service	1.0013	0.0000	0.00
Networking	Public IP Addresses	All	IP Address Hours	385.0000	0.0040	1.54
Storage	Locally Redundant	US West	Premium Storage - Page Blob/P10 (Units)	0.9999	19.7100	19.71
Storage	Locally Redundant	All	Standard IO - Block Blob (GB)	1.9610	0.0647	0.13
Storage	Geo Redundant	All	Standard IO - Page Blob/Disk (GB)	46.7216	0.0956	4.47
Storage	Locally Redundant	All	Standard IO - Table/ Queue (GB)	2.1225	0.0809	0.17
Storage	Geo Redundant	All	Standard IO - Table/ Queue (GB)	0.0030	0.0956	0.00
Data Management	All	All	Storage Transactions (in 10,000s)	38.6540	0.0005	0.02

Notice that one of our team members kept a fairly massive D4 VM running and run up a bill for $456. 62 cents per hour adds up. $18 / day etcetera. So…

GOTCHA-3 *Don't let costs creep up on you. Learn how and when to shut down cloud resources. This is a branch topic (kilroy) that deserves its very own djocumentation.*

Now that I have created spaces for djunior djazz and djargon I have decided to build them into the VS Solution 'djak'; so here is the Create for djunior, another Django web application. Notice I am careful to stipulate Python27, my 32-bit installation of Python.



Add Virtual Environment

Location of the virtual environment

Specify the name or location of the virtual environment. If one already exists, we will detect the base interpreter for you.

env

Select an interpreter to create the virtual environment from.

Packages in your base interpreter will not be available in the virtual environment until you install them.

Python27

We found a requirements.txt file.

This file contains external packages that are required by your project. We can use pip to download and install these dependencies automatically.

[✓] Download and install packages

Actions we will perform:

- Create a virtual environment
- Install packages from requirements.txt
- WARNING - the selected interpreter may not support virtual environments.

I did the same for djazz and for djargon I'm going to use Bottle (which is a bit simpler than Flask and djargon is intended to be a simple website with just some djocumentation like djthis.

Here is the Create page for djargon; notice I still choose the Python27 32-bit installation as my source even though this is a step away from Django. Djargon is not a Django web framework Project; it is a Bottle web framework Project.

Summary of Work So Far

I have done five things inside a Visual Studio Solution and some corresponding things on the Azure portal.

In Visual Studio

1. Created a Django web app called djak. And published it to Azure (see below).
2. Created a Django web app called djunior.
3. Created a Django web app called djazz.
4. Created a Bottle web app called djargon.
5. Added two resource files to the solution: A LiveOceanServer notes file and this documentation as an MHTML document (intended to be installed at djargon).

On Azure

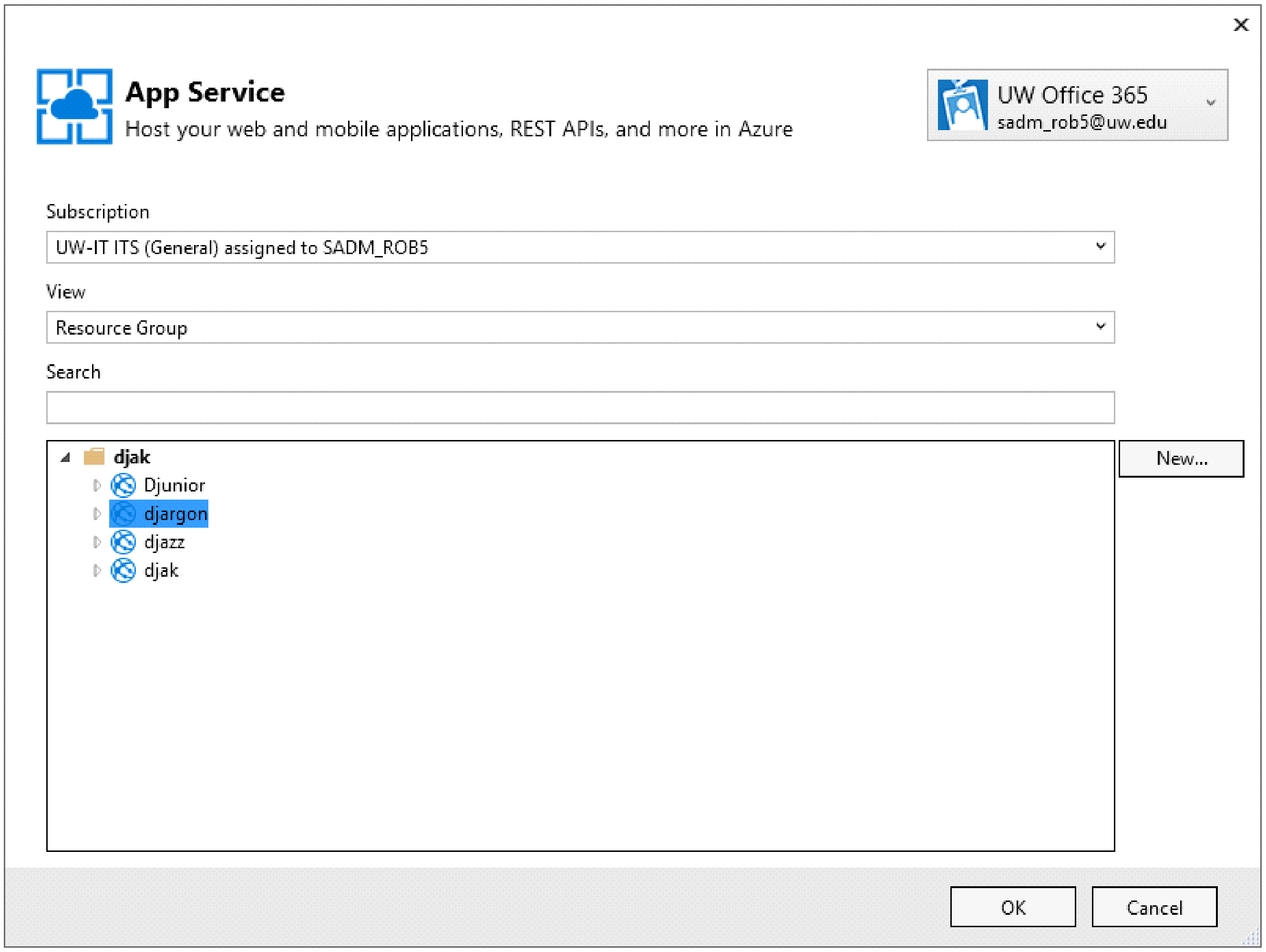
1. Obtained a subscription and logged into the portal using an anonymous browser.
2. Created a resource group called 'djak'
3. Created a web application service plan called djak_app_service_plan that can manage multiple web applications.
4. Created a web app (with no database yet) called djak (http://djak.azurewebsites.net)
5. Created a web app plus MySQL database called djunior. (http://djunior.azurewebsites.net)
6. Created a web app plus MySQL database called djazz (http://djazz.azurewebsites.net)
7. Created a web app (no database) called djargon (http://djargon.azurewebsites.net)
8. Published a rudimentary version of djak to djak from visual studio.

What I have not done -- and this is really important to fix -- is I have not managed to create super user accounts for the Django instances (djak, djunior and djazz) using the 'Synch DB' feature. This continues to fail and it is not clear exactly why. Even upon running 'manage.py migrate' from the command line (env) with success there is still no super user for djak; so this is a major kilroy situation.

Incidentally when I re-export this documentation from OneNote to its location in the Resources folder of the djak Solution I find that Visual Studio notices this and asks if I would like to reload the document into the Solution; so I say yes of course; and I also regularly commit the entire Solution and Sync to GitHub.

To Continue

I now publish djargon.



App Service

Host your web and mobile applications, REST APIs, and more in Azure

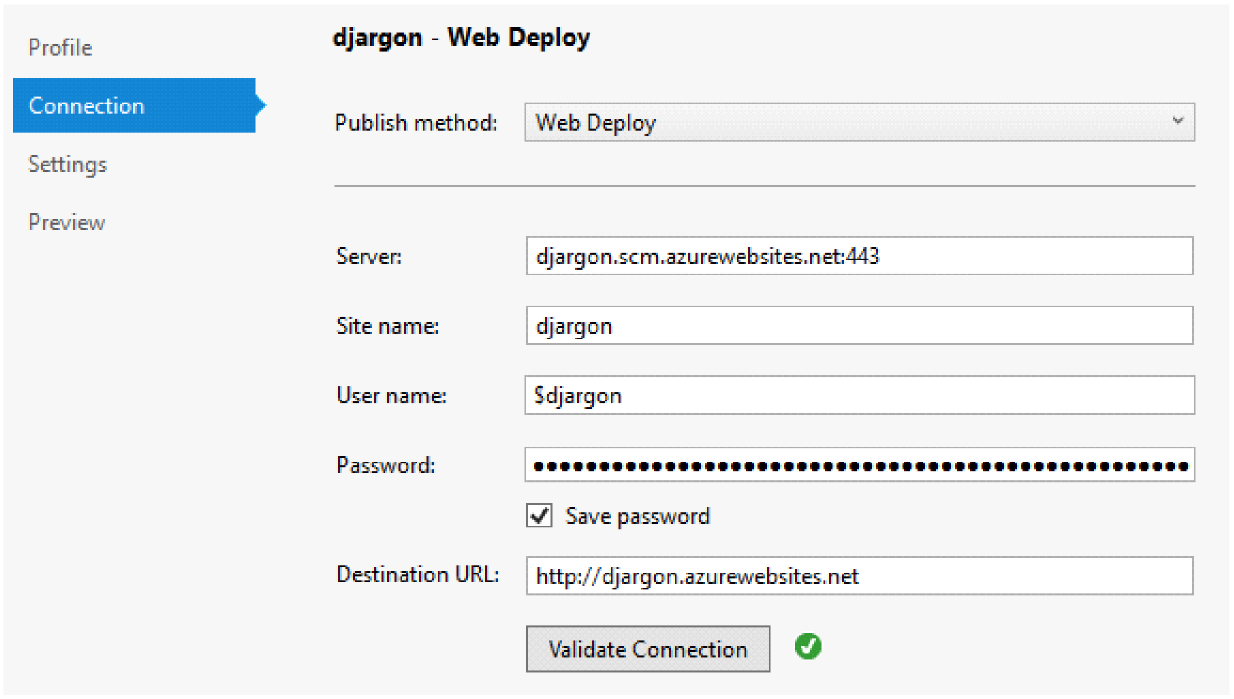
UW Office 365
sadm_rob5@uw.edu

Subscription: UW-IT ITS (General) assigned to SADM_ROB5

View: Resource Group

Search:

djak
- Djunior
- djargon
- djazz
- djak



djargon - Web Deploy

Publish method: Web Deploy

Server: djargon.scm.azurewebsites.net:443

Site name: djargon

User name: $djargon

Password: ●●

✓ Save password

Destination URL: http://djargon.azurewebsites.net

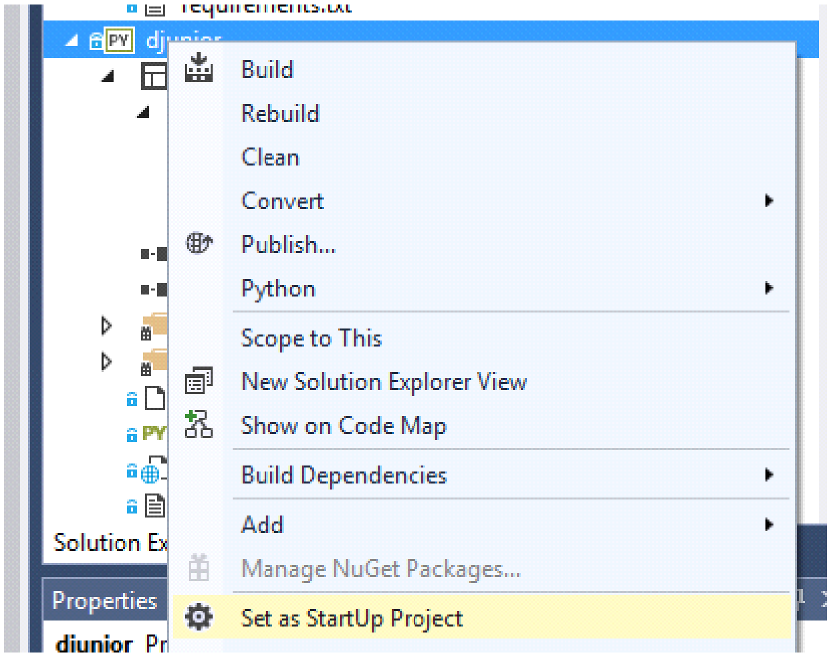
But upon Preview we hit a problem:

Invoke build failed due to exception 'The value "" of the "Project" attribute in element <Import> is invalid. Parameter "path" cannot have zero length. C:\Program Files (x86)\MSBuild\14.0\bin\Microsoft.Common.targets'

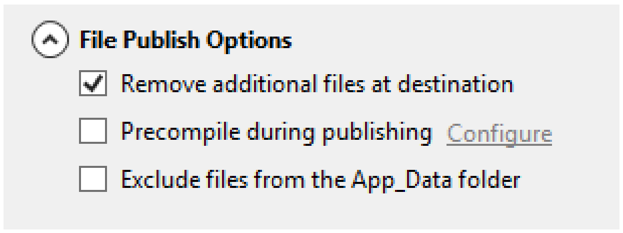
I find (both here and subsequently) that simply clicking 'Try Again' works. In fact that might have worked the first time around for djak when I hit that Preview error.

So now my Bottle code for djargon is connected to the Azure website.

For djunior:



I added one additional detail in the Publish process:


File Publish Options

[✓] Remove additional files at destination

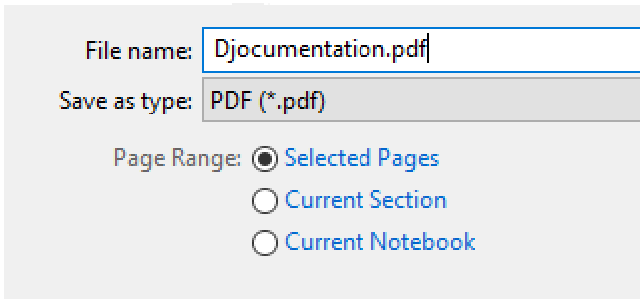
[ ] Precompile during publishing

[ ] Exclude files from the App_Data folder

Expanding the **File Publish Options** I find the 'Remove additional files at destination' and this will clean up the original placeholder html installed when I created the website. So let's do that as a matter of course. Since we develop and then Publish cyclically there are many opportunities to do this. It may not remember how it is set so try and check it in the future.

Ok now how was this web page built into the http://djargon.azurewebsites.net website? To be honest it took some time plunking around with the Bottle documentation. Here is an outline of the steps.

1. Get OneNote running on a Windows PC and enable the WindowsKey + Shift + s screencap utility.
2. Write the stuff above inserting screencaptures liberally. This is incredibly tedious.
3. Export this document to a pdf in a folder inside the djargon Project: Sub-folder of static called 'pdf'


File name: Djocumentation.pdf

Save as type: PDF (*.pdf)

Page Range: [● Selected Pages] ○ Current Section ○ Current Notebook

4. In Visual Studio add the folder and the pdf file that were just installed in the djargon Project sub-folder of the djak Solution.
5. Edit the routes.py file and the layout.tpl in the djargon Project; see below. This will create a new link on the title bar of the djargon website that points to the pdf.
6. Publish djargon and see if the documentation shows up.

Here is the first bit of code one adds to routes.py, a Python file about binding callbacks to sub-pages of the website. This is a hugely important file as it would also be the starting point for a Bottle-based API.

```
# kilroy added this for static content
from bottle import static_file
```

Here is the second bit of code:

```
# kilroy added this @route to bind the method 'kilroy_hardcoded_callback()'
#   to the '/documentation' extension; see layout.tpl
@route('/documentation')
def kilroy_hardcoded_callback():
    return static_file('Djocumentation.pdf', root='./static/pdf/')
```

```
# kilroy: documentation.html at ./static/html is the older version; but this was a pain to build out of
#   OneNote; it took several tedious steps: First it exports as a Microsoft-specific .mht file; then load that
#   into IE and export as HTML. Now you have a huge folder of images associated with the text; but VS
#   allows you to inline those as 64-bit encoded; so you repeat that a couple dozen times and your images
#   appear in your website. Referencing them did not work…
```

And then finally create an entry in layout.tpl corresponding to this URL extension (last line, Djoc):

```
<div class="navbar-collapse collapse">
   <ul class="nav navbar-nav">
      <li><a href="/home">Home</a></li>
      <li><a href="/about">About</a></li>
      <li><a href="/contact">Contact</a></li>
        <li><a href="/documentation">Djoc</a></li>
```

In the larger context of building out web content using Bottle there is room for CSS, for HTML, for pdfs like the one shown here; but more at the soul of the issue there is room for dynamic web pages and for API calls. So that's the next thing to do; and I think it will be more efficient at the moment to stay in the Bottle framework rather than hopping back to Django; at least until a reason for doing so appears.