Deploying and Configuring West Wind Web Connection Applications

So, you've built your shiny new **Web Connection** Application on your local machine for development and you're proud of what you've accomplished. Now it's time to take that local masterpiece and get it online: **Ship it!**

How do you do this?

In this long article I'll take you to the process of creating a small placeholder application and deploy it to a live server on a hosted service called **Vultr** which provides low cost, and high performance virtual machine services that are ideal of hosting a Web Connection site.

I'll use a brand new, virgin Virtual Machine of Windows Server 2016 and configure it from scratch by installing base applications on the server, configuring IIS, and uploading and installing a new Web Connection application.

Here's what we'll cover:

- Creating a new Web Connection project
- Customizing the project slightly for a custom 'application'
- Setting up a Vultr Virtual Machine Windows Server
- Configure the Windows Server

    - Install Server Base Applications (editor, browser, tools)
    - Install IIS
    - Install FoxPro

- Package the Web Connection Project
- Upload the project
- Set up the project on the server
- Test the application
- Install a free TLS Certificate for HTTPS
- Make code changes and update the server

Creating new Applications - Projects Organize your Application

Starting with Web Connection 6.0 the process of creating a new application has been made much more consistent through a way of organizing a project into a well-known folder structure. The idea is to recognize what most projects are made up of:
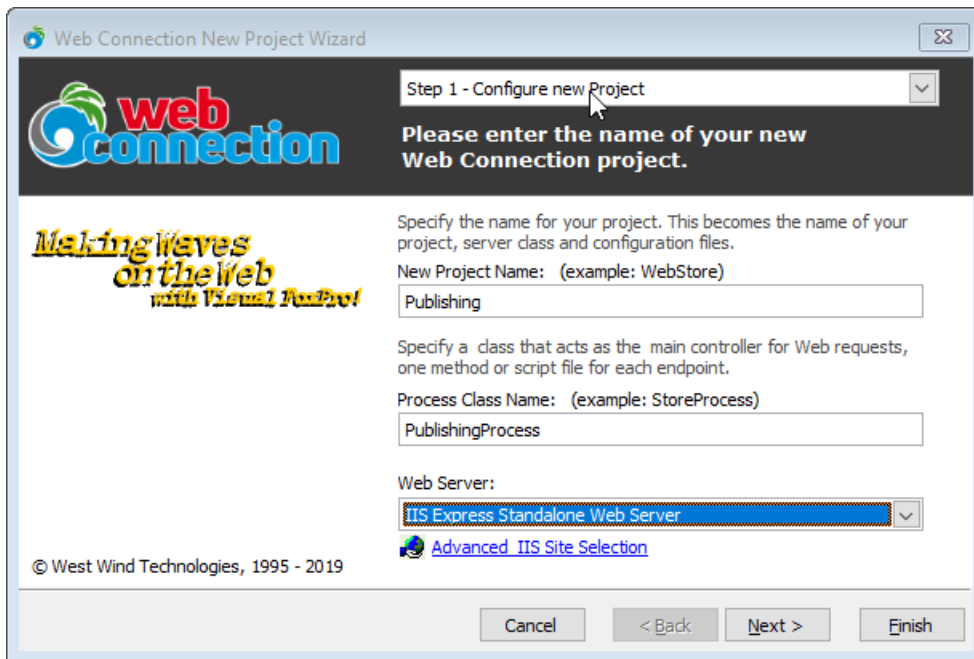
- Code and/or Binary files

- Web content

- Data

The Web Connection new project structure creates a single top level folder, with subfolders for `Deploy` (code and binaries), `Web` and `Data` . The `Deploy` folder then also contains the `Temp` folder where message files and logs are stored. All of this boils down to a known and repeatable set of locations so that a generic installer can properly configure a Web application more easily.

To demonstrate lets create a brand new Web Connection application called 'Publishing'. I won't put any logic into this project other than making a few text and text based code changes - it'll be a stock Web Connection project - that's all we need and it'll make for a small project to distribute to boot.
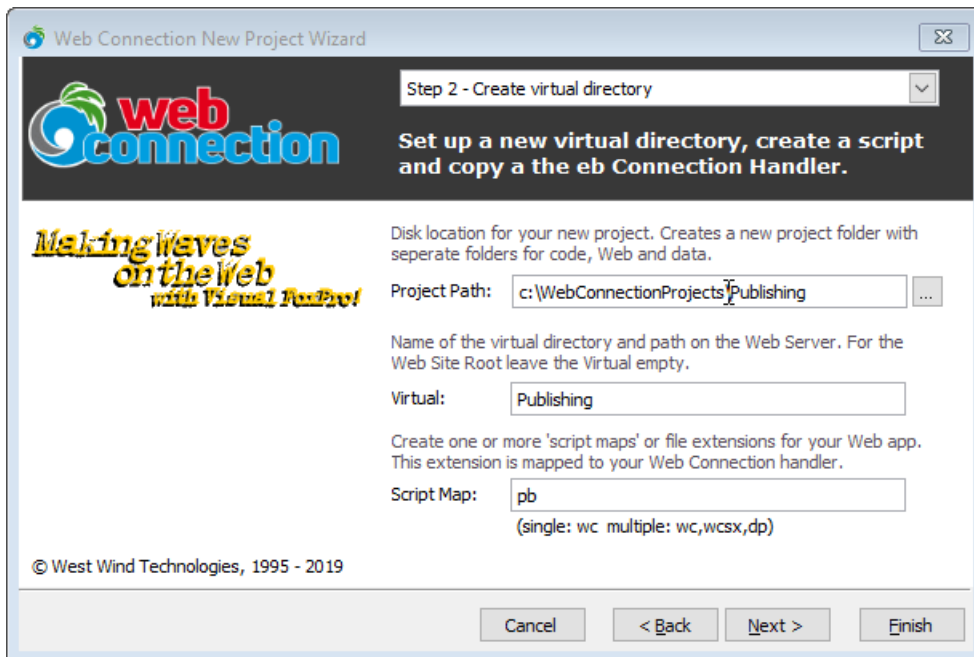
Creating a new Web Connection Project

Create the new project by typing `DO Console` in FoxPro **as an Administrator**.

I chose the name for the project and the Process class as `Publishing` and `PublishingProcess` respectively. Note I use IIS Express here locally so I don't have to install anything.

I use the default to create the project in the `WebConnectionProjects` folder, but you can really put this project anywhere. I also set up a script map for the `.pb` extensions so I can access requests with `.pb` and map them to Web Connection handlers.
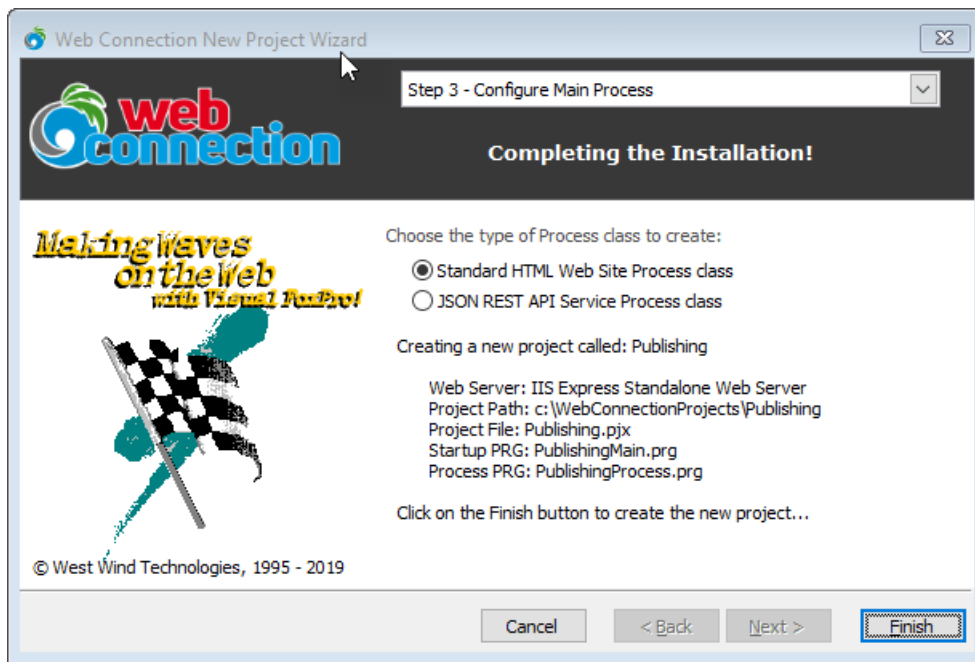
Finally I publish the project as a Web project and let 'er rip. If all is working you should now have a running Web Connection Server and loading the placeholder home page:

```
Running:
DO Launch.prg WITH "IISEXPRESS"


Web Server used:
IIS Express

Launched IISExpress with:
DO console WITH "IISExpress","..\Web",7000

Launching Web Url:
http://localhost:7000

Server executed:
DO PublishingMain.prg
```
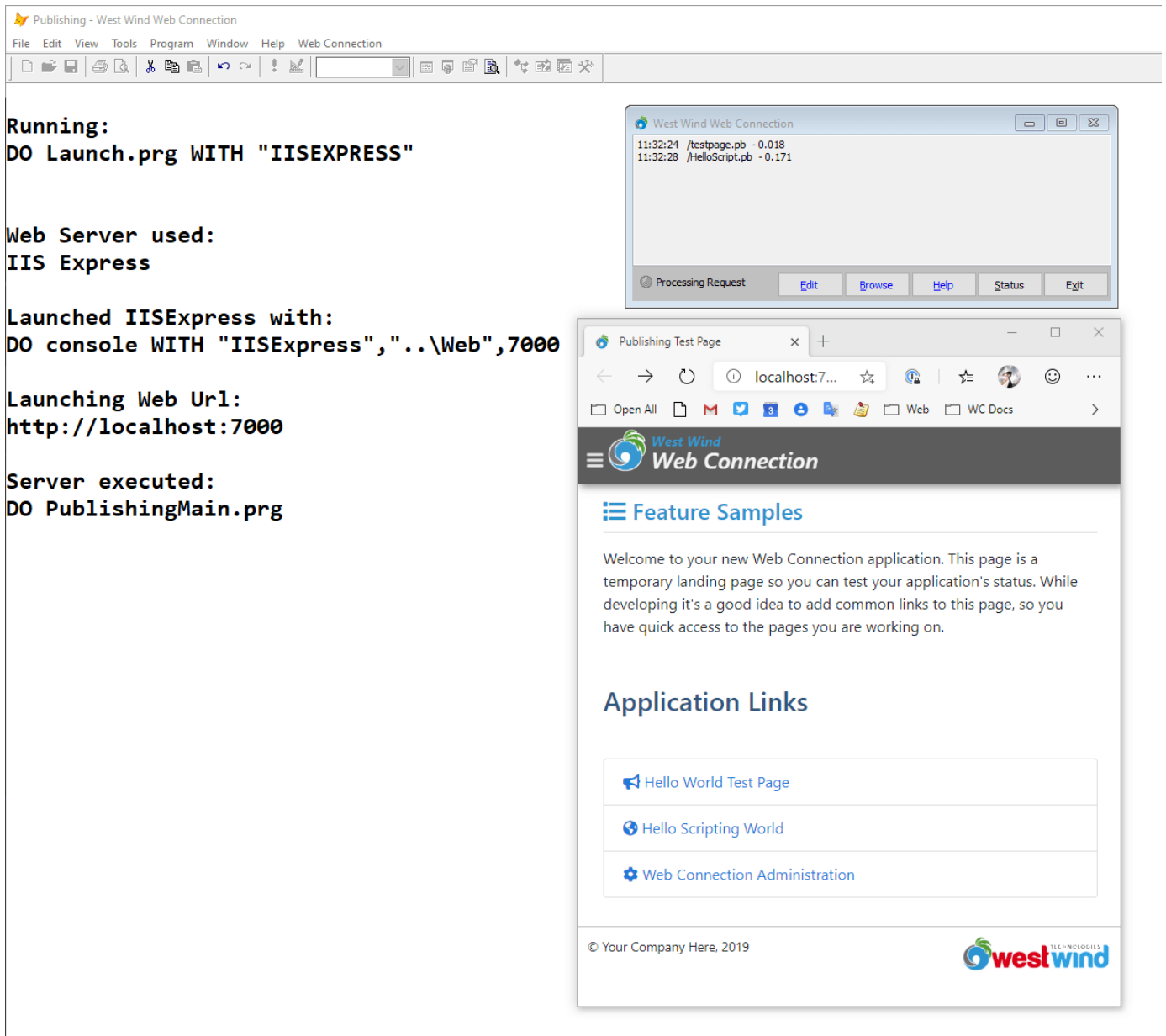
Notice that Web Connection 7.0 and later now has a **simple startup process launcher** that automatically starts the application with `DO Launch` or `Launch()` . The installer runs this automatically, but when you restart the application you'll have to run `Launch()` or `Launch("IISExpress")` from the command window.

This does the following:

- Launches the Web Server if required (IIS Express in this case)
- Launches the Web Connection Server
- Opens a Web Browser on the default URL (http://localhost:7000 for IIS Express)

You should be able to click around the Web page to access the sample links at this point.

We have a running application. Yay!

Making a couple of small Changes

Just so we can see something a little custom, lets change the two sample links. To show off some of the new

features in Web Connection 7.0 lets highlight **site editing and Live Reload functionality**.

If you have **Visual Studio Code** installed you can click on the **Edit button** and open your new site in VS Code. Code is a great light weight cross platform text editor with support for tons of languages **including FoxPro**. **The editor command is configurable in** `Publishing.ini` and it defaults to open VS Code in the project root, which gives access to both the `Deploy` and `Web folders`
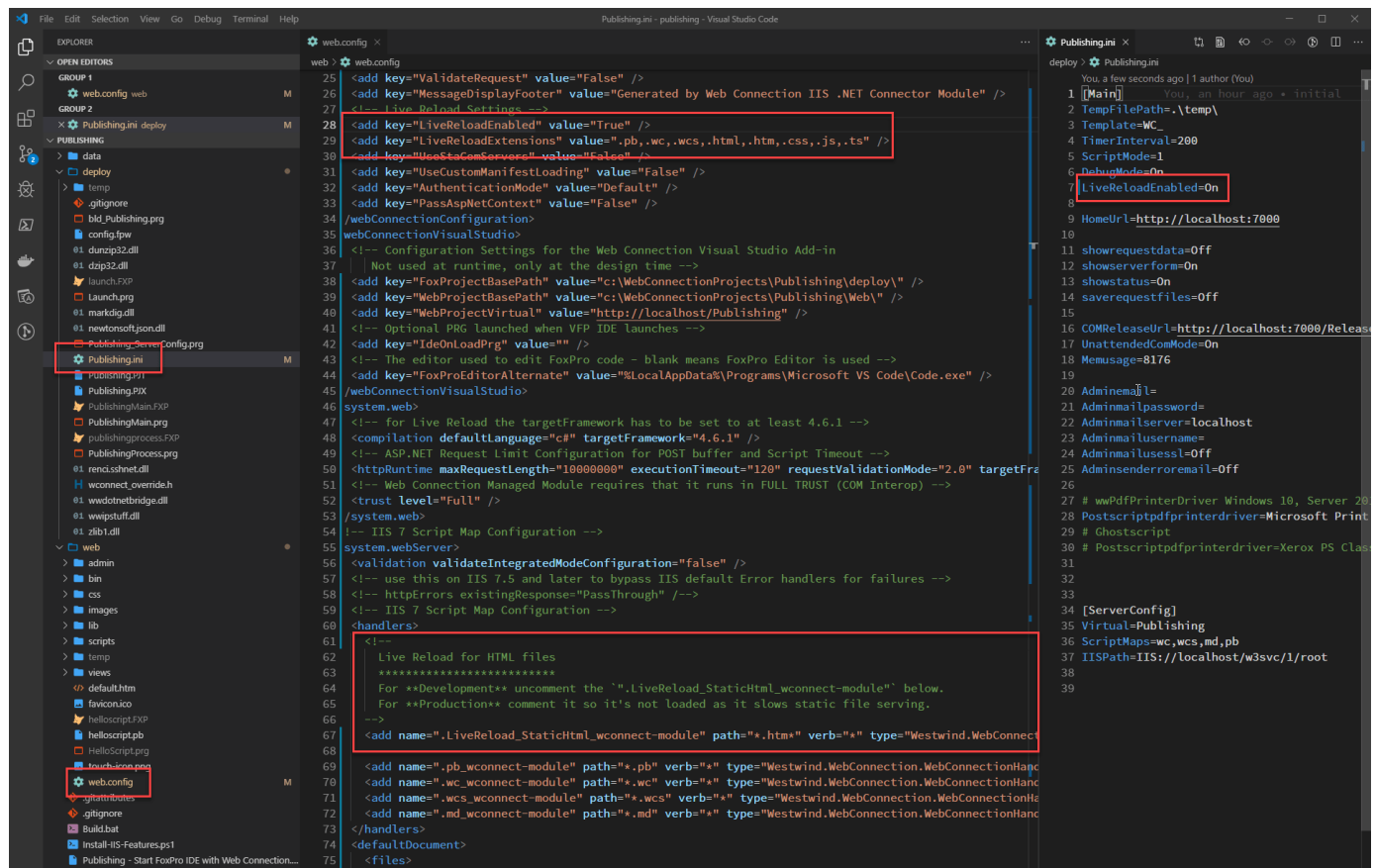
- **Deploy**
  Holds all your FoxPro Source code and 'server' resources and your project's configuration file (ie. `YourProject.ini`).

- **Web**
  Holds all your Web files (templates, html, css, js etc.) as well as `web.config` which configures the Web Connection Web settings for IIS and Web Connection

Enabling Live Reload

A new feature in Web Conection 7.0 is live reload which lets open your Web page, and when you make a change to any Web files, or source code (outside of FoxPro's editor say from VS Code), the code is updated and the browser auto-refreshes. So, when you make a change you see that change immediately reflected in the browser.

Ch..., Ch..., Changes

Let's make some changes. Let's open the `web/default.htm` page and change the headline. Position the Web Browser and Editor so you can see both. Then go in the editor and change the `Feature Samples` header text to `Web Connection Deployment Demo`. If you pay attention you'll see that as soon as you save the change to disk, the Web Page shows the new header! The same works if you make changes to a CSS or JS file.

We can also make changes to Web Connection templates. Open `web/HelloScript.pb` page which is the second test link. You probably want to add the syntax in VS Code, by clicking on the language drop down in the lower right of VS Code's status bar and choose HTML. Again make a change to the header and chage it from the Hello World text to `Ready to Deploy Application`.

Finally we can even make code changes and see those reflected. Open `deploy/publishingProcess.prg` and change the `StandardPage()` header in the `TestPage()` method to read `Hello from FoxPro Publish Project`. Save and notice that the FoxPro server shuts down and restarts itself, and then refreshes the Web page to show the new text.

This is a new feature that is very productive.

> ⚠ **Important: Turn off Live Reload for Production**
>
> Make sure you **turn Live Reload off for production** applications as this feature has some overhead. This feature **requires IIS 10 or later** (Server 2012 R2 /Windows 10 or later or IIS Express 10).

So now we have an updated 'application' that has a few customization. Although this obviously an extremely simple application, it serves nicely to demonstrate the deployment process as it's small and quick to send up to the server as we'll do several times for this demonstration.

Ready to Publish

Ok - so we've made some changes to our project to have a highly, highly customized Web Server we can publish to a brand new Web Server machine. 🗅

Build an EXE

Generally I like to run my application during development using just the PRG Files. I launch with `DO PublishingMain.prg` (or have it launch through the new `launch.prg`). Once I'm ready to deploy however, I **have to compile the project into an EXE** that can be deployed to the server.

> ℹ **Admin Rights Required for Compilation**
>
> Note you need to be an Administrator to compile the project as the project contains a COM component which has to be registered. COM objects require Admin rights for registration.

Test your Server in File and COM Modes

Beyond that you should test your server as it would be run on the server:

- Turn off Debug Mode
- Run the EXE from Explorer
- Or: Invoke the COM Object

If you're running as a COM object, before anything test by instantiating the COM server like this in VFP:

```foxpro
o = CREATEOBJECT("Publishing.PublishingServer")
? o.ProcessHit("query_string=wwMaint~FastHit"))
```

This fires a test request against the server. You can also access a specific page as a GET request with:

```foxpro
? o.ProcessHit("physical_path=Testpage.pb")
```

You can use `ShowText()` or `ShowHtml()` in `wwUtils.prg` to display the content if it's long.

Run the Application as an EXE or COM Object Locally

Once that works next run your application using the EXE or the COM object. In theory this should just work because you're running the same application in exactly the same environment so other than the packaging an EXE is no different than running inside of the FoxPro IDE.

Switching to COM can be done on the Admin Page.

- Admin Page at **http://localhost:7000/admin/admin.aspx**
- Then click on the **Web Connection Module Administration**

On this page you can toggle Web Connection between File and COM Mode. Here I've switched into COM mode:

When Web Connection compiles it locally registers the server for the `INTERACTIVE` user so servers can show on the desktop. Note that standard COM registration will not do this and simply inherit the IIS account that is running the Application Pool.

You'll want to test your application now running in COM mode as well as file mode as an EXE and ensure the app runs as you'd expect locally.

> If the app doesn't run locally, it sure won't run on the Web Server either, so make sure it all works **before you send it up to the server**.
>
> Debugging is a lot easier locally than on a remote Web server!

Understanding the Project Layout

So at this point you should have a project that works and runs. The next step is to package up everything into something you can install on the new server.

Let's review a new project layout.
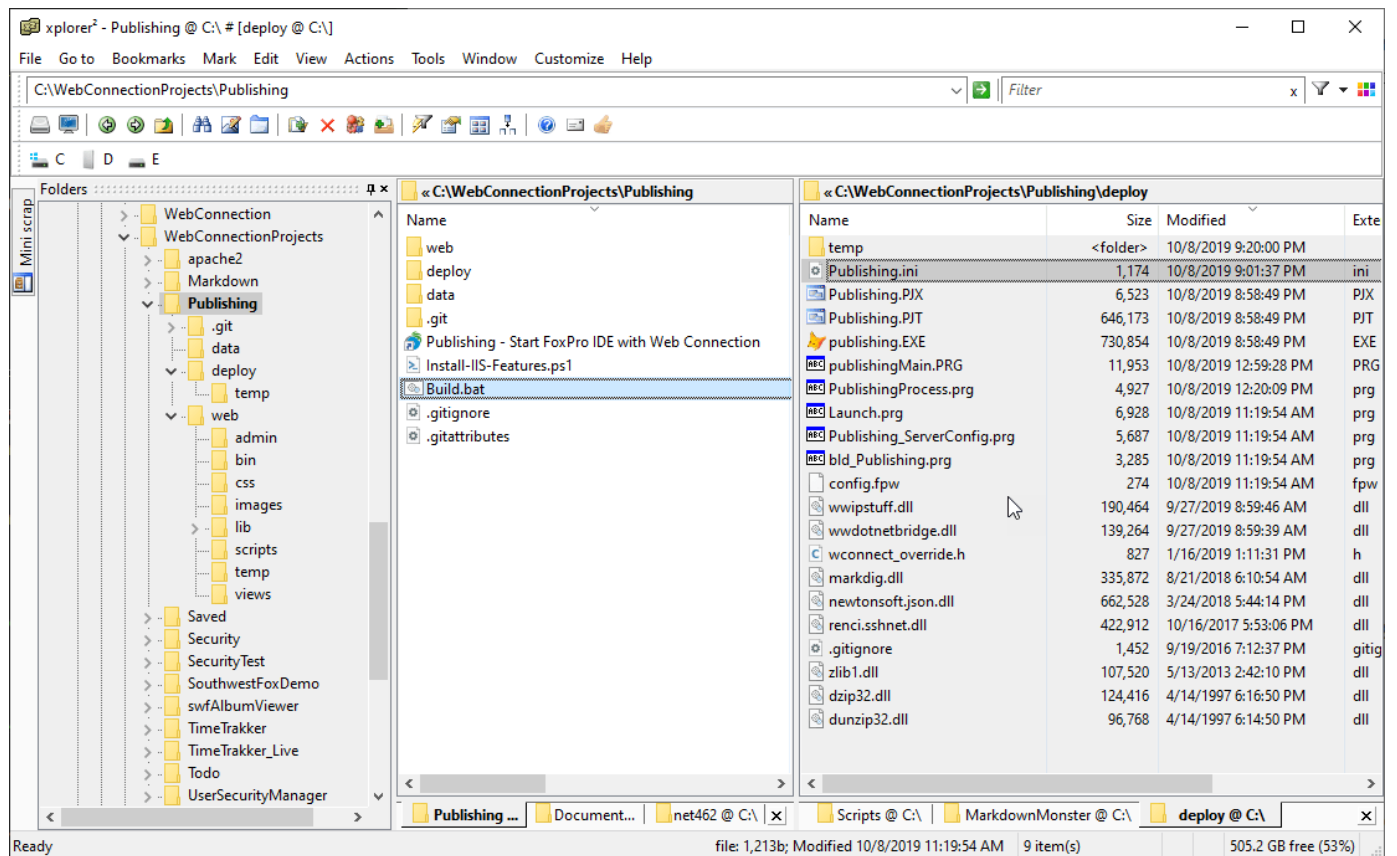
Root Folder

The root folder of the project contains administration files. IIS Installation, a `build.bat` you can use to package files and a link to locally start FoxPro in the `deploy` folder.

Deploy Folder: Source Files and Binaries

The `deploy` folder is your FoxPro folder - this is where you code goes as well as the compiled binary of your application (the EXE). The folder also holds Web Connection support DLLs that need to be deployed to the Web server.

When you deploy this folder only the binary files are picked up - source code files are ignored.

Web Folder: Web Resources

The Web Folder holds all your Web resources which are Web Connection Scripts and Templates, HTML, CSS and JavaScript files, images and anything else that your Web application needs to run.

Packaging up Everything: Build.bat

To help with getting everything ready for a first time deploy when you basically need to move everything to the

Web server, Web Connection 6.5 and later provides a `build.bat` in the root folder which creates a ZIP file of all the files required to run your application.

You can run it by double clicking on the folder which produces a new `Build` directory which contains all the copied files and a zip file of everything.
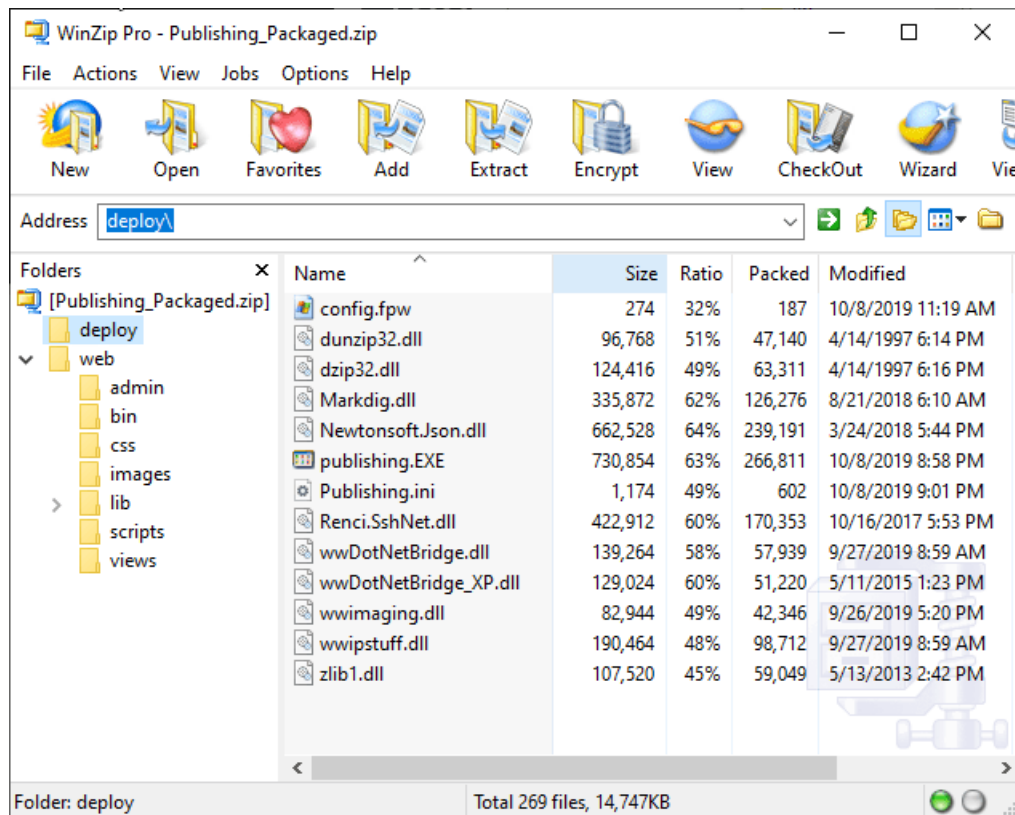
You're now ready to take that publish file to your server.

But before we can do that we need to configure the server and get it ready for running as a Web Server.

Setting up a New Virtual Machine Windows Server in the Cloud

I'm going to use a brand new Virtual Server on **Vultr** which is the hosting company I personally use to host my own Web sites. Vultr is very reasonably priced (especially compared to Azure and AWS), and provides high performance hardware for the prices they charge. Vultr is a plain VPS provider, meaning they provide virtual and physical servers, storage space but little else in the way of additional application services. If you need support services, like extra storage, hosted SQL or NoSQL solutions, then you need to look into more complete services like Azure or AWS. But if all you want is to create one or mor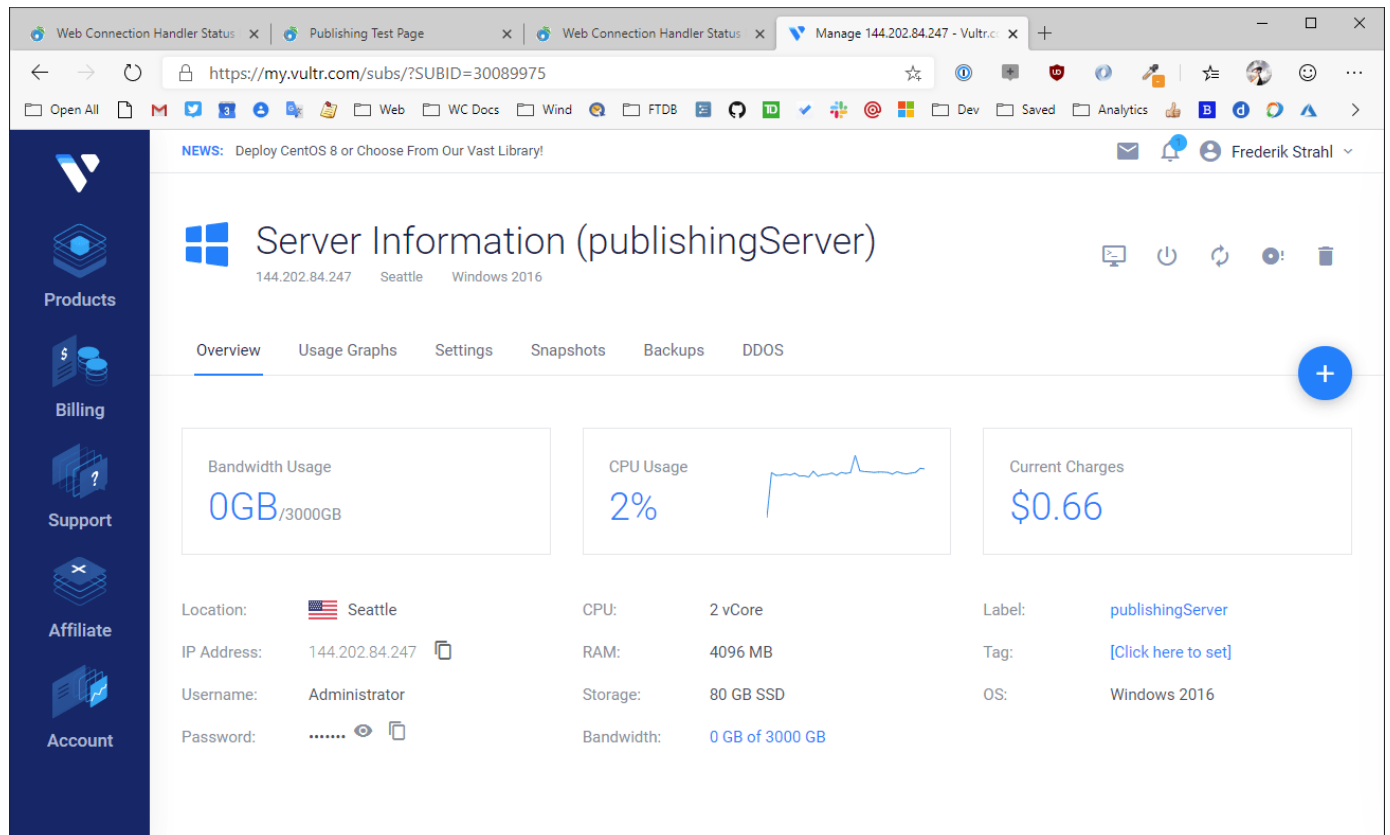e Virtual Servers to host in the cloud, then you'll be hard pressed to beat the value that Vultr provides. I spent a lot of time looking around for a good Windows hosting service, and Vultr is what I ended up with.

I can spin up a new Vultr VPS server in about 10 minutes and I've done just that for this demonstration.

This is what I think is the minimum hosting set up you should use for a Web Connection application which is:

- 2 cores (never use a single core setup!)
- 4gb of RAM
- 80gb of Disk Space
- Windows Server 2016

This setup costs $40/month and includes the Windows Server license. I use an older version of this package for hosting my Web Server and that site runs 20+ sites, SQL Server, MongoDb. This hardware goes a long way and it's very fast for all of my Web sites. The biggest limitation on this package is the disk space. 80gb is not a lot when you figture the Windows footprint (my old package is more expensive but includes more disk space). The next step up is $70 for 4 cores, 16gb RAM and 160gb of storage which is totally worth it if you need it.

Remember these specs are for VPS servers which doesn't reliably compare to 'real' processors, but I found that

Vultr is much closer than Azure or Amazon in performance to what I would expect in a physical setup of these specs. And on either of those platforms you'd pay **at least** twice as much for lesser VPS hardware.

Remote Desktop for Server Setup

Vultr sets up a new virtual Server which is basically base Windows Server 2016 installation. The first thing we need to do is use Remote Desktop into the new server and start configuring it.

Create a new User

The first rule for a new server is: Don't use the Administrator account. Instead the first thing you should do is create a new user and add it to the **Administrators** group then log on and use only that account.

You can then disable the Administrator account. This reduces your machine attack surface as most server password attacks start with the Administrator account.

Install Required Software

Next up there are a number of bits of software that are needed on any new machine. At the very least you should always install a Web Browser other than IE (Chrome or Firefox), a good editor (VS Code) and an FTP Client (Filezilla).

I highly recommend using **Chocolatey** for this. Chocolatey is a package manager for Windows that allows you quickly install common tools and applications for Windows silently from the commandline. You install Chocolatey which is an application that sits in your global path and lets you execute chocolately commands.

To install Chocolatey you can run a single Powershell command from an Administrator prompt:

```powershell
Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

Once it's installed you can install software from the (Admin) Command Line - silently. For example:

```powershell
choco install vscode
choco install GoogleChrome
```

So here is my typical server installation process:

- Install Chocolatey

- Run a Chocolatey script to install tools

    - VS Code Editor (or another editor of choice)

    - A decent Browser (Chrome or Firefox or Edgium)

    - 7zip

    - FileZilla FTP Client

    - FileZilla FTP Server (if you need to upload files)

- Install IIS using script provided with Web Connection

- Install **Visual FoxPro Runtimes** (or full IDE)
- Install FoxPro SP2 (if installing IDE)

I've provided the scripts for all these tasks in the **Github Repository** for this session with the exception of the FoxPro installation. I tend to store scripts like these in an `\admin` folder off the root of the server.

The IIS install Script ships with Web Connection. The others are custom scripts I use and am sharing here - make sure you check them before running to add or remove those that don't fit your environment. You can find more things to install on the **Chocolatey site search**.

Install Chocolatey

Once installed you should shut down Powershell and restart it. Once installed you can now easily install any of Chocolatey's packages. On the server there are a few things that I consider ABSOLUTELY necessary:

```powershell
# Don't prompt for installations
choco feature enable -n allowGlobalConfirmation

# Remove the annoying UAC prompts on Server
choco install disableuac

# Install essential Apps
choco install GoogleChrome
choco install vscode
choco install 7zip.install
choco install curl
choco install filezilla
choco install filezilla.server
choco install git
choco install tortoisegit
choco install xplorer2
choco install procexp
```

You can check Chocolatey for additional things you might need on the server - there are hundreds of tools and applications and even full software to install from there.

Install IIS

Next you need to install IIS on the server. Windows Server makes this a royal pain in the butt with it's user-hostile Roles and Features interface.

Luckily you can sidestep that mess, and use a Powershell script instead.

Web Connection ships with an `Install-IIS-Features.ps1` script that installs all the required components needed to run a Web Connection application. You can go to your local machine and copy that script to the clipboard, then create a new Powershell file called `Install-Iis-Features.ps1` and run it **as an Administrator**.

Here's what that script looks like:

```powershell
# This script installs IIS and the features required to
```

```powershell
# run web Connection.
#
# * Make sure you run this script from a Powershel Admin Prompt!
# * Make sure Powershell Execution Policy is bypassed to run these scripts:
# * YOU MAY HAVE TO RUN THIS COMMAND PRIOR TO RUNNING THIS SCRIPT!
Set-ExecutionPolicy Bypass -Scope Process

# To list all Windows Features: dism /online /Get-Features
# Get-WindowsOptionalFeature -Online
# LIST All IIS FEATURES:
# Get-WindowsOptionalFeature -Online | where FeatureName -like 'IIS-*'

Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebServerRole
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebServer
Enable-WindowsOptionalFeature -Online -FeatureName IIS-CommonHttpFeatures
Enable-WindowsOptionalFeature -Online -FeatureName IIS-HttpErrors
Enable-WindowsOptionalFeature -Online -FeatureName IIS-HttpRedirect
Enable-WindowsOptionalFeature -Online -FeatureName IIS-ApplicationDevelopment

Enable-WindowsOptionalFeature -online -FeatureName NetFx4Extended-ASPNET45
Enable-WindowsOptionalFeature -Online -FeatureName IIS-NetFxExtensibility45

Enable-WindowsOptionalFeature -Online -FeatureName IIS-HealthAndDiagnostics
Enable-WindowsOptionalFeature -Online -FeatureName IIS-HttpLogging
Enable-WindowsOptionalFeature -Online -FeatureName IIS-LoggingLibraries
Enable-WindowsOptionalFeature -Online -FeatureName IIS-RequestMonitor
Enable-WindowsOptionalFeature -Online -FeatureName IIS-HttpTracing
Enable-WindowsOptionalFeature -Online -FeatureName IIS-Security
Enable-WindowsOptionalFeature -Online -FeatureName IIS-RequestFiltering
Enable-WindowsOptionalFeature -Online -FeatureName IIS-Performance
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebServerManagementTools
Enable-WindowsOptionalFeature -Online -FeatureName IIS-IIS6ManagementCompatibility
Enable-WindowsOptionalFeature -Online -FeatureName IIS-Metabase
Enable-WindowsOptionalFeature -Online -FeatureName IIS-ManagementConsole
Enable-WindowsOptionalFeature -Online -FeatureName IIS-BasicAuthentication
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WindowsAuthentication
Enable-WindowsOptionalFeature -Online -FeatureName IIS-StaticContent
Enable-WindowsOptionalFeature -Online -FeatureName IIS-DefaultDocument
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebSockets
Enable-WindowsOptionalFeature -Online -FeatureName IIS-ApplicationInit
Enable-WindowsOptionalFeature -Online -FeatureName IIS-ISAPIExtensions
Enable-WindowsOptionalFeature -Online -FeatureName IIS-ISAPIFilter
Enable-WindowsOptionalFeature -Online -FeatureName IIS-HttpCompressionStatic
Enable-WindowsOptionalFeature -Online -FeatureName IIS-ASPNET45
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebSockets

# The following optional components require Chocolatey

# Install UrlRewrite Module for Extensionless Urls (optional)
#choco install urlrewrite -y

# Install WebDeploy for Deploying to IIS (optional)
```

```
# Install WebDeploy for Deploying to IIS (optional)
# choco install webdeploy -y

# Disable Loopback Check on a Server - to get around no local Logins on Windows Server
New-ItemProperty HKLM:\System\CurrentControlSet\Control\Lsa -Name "DisableLoopbackCheck" -
Value "1" -PropertyType dword
```

Keep an eye on the script execution and look for errors. This script takes a while to run (about 5 minutes or so on my new Virtual Machine server). Once it's done you should be able to access IIS Management Console (type `IIS` into the Windows search box).

Install Web Deploy and UrlRewrite

IIS installation provides core features, but there are usually at least a couple of addins I always install on IIS. These addins come from Microsoft's Web Platform installer where they can be manually installed. Or you can use Chocolatey as I'll do here.

Web Deploy is an IIS plug-in that allows you to deploy IIS from other machines using the MsDeploy tool. MSDeploy is integrated into Visual Studio and I'll use that to publish the site to IIS.

UrlRewrite lets you rewrite URLs so that you can access one URL and automatically send the request to a different URL in your site. This is useful for many things such as automatically forcing all requests to use HTTPS for example. For Web Connection, if you want to use ExtensionLess URLs you can use UrlRewrite to rewrite extensionless URLs to your Web Connection application.

The IIS Script includes the Chocolatey code to also install `WebDeploy` as well as `UrlRewrite` which is another add-on module for IIS. The default script has these commented out because they use Chocolatey and so may fail if it's not installed. Un-comment them before running the script or if you forgot execute them one at a time after the IIS install is complete:

```powershell
choco install WebDeploy
choso install UrlRewrite
```

Alternately install an FTP Server to push Files

If you don't want to use Visual Studio and Web Deploy, then you should probably install an FTP server. FileZilla Server works great and is easy to use. Whatever you do, **don't use the IIS FTP server** - it's terrible.

To install filezilla:

```powershell
choco install filezilla.server
```

Once installed you can then use use an FTP client to copy files from your local machine to the Web server, but the process is more manual than with WebDeploy which does incremental updates.

Setting up your Web Connection Application

The server is now ready. Now we need two more steps before we can get the site to run:

- Copy the packaged Build files to the server
- Configure the Web Connection Web Site

Copying Files

Now that the server is configured and ready to go - all we need to do now is get our Web site over to it.

There are a few ways to do this:

- Remote Desktop Drive Mapping to Local Drive
- Visual Studio Publish
- Using an FTP Server

The easiest here is the Remote Desktop File sharing and while that works it's infuriatingly slow, but it works and that's what I'll use here. I'll also show using Web Deploy and Visual Studio publishing later to update Web resources.
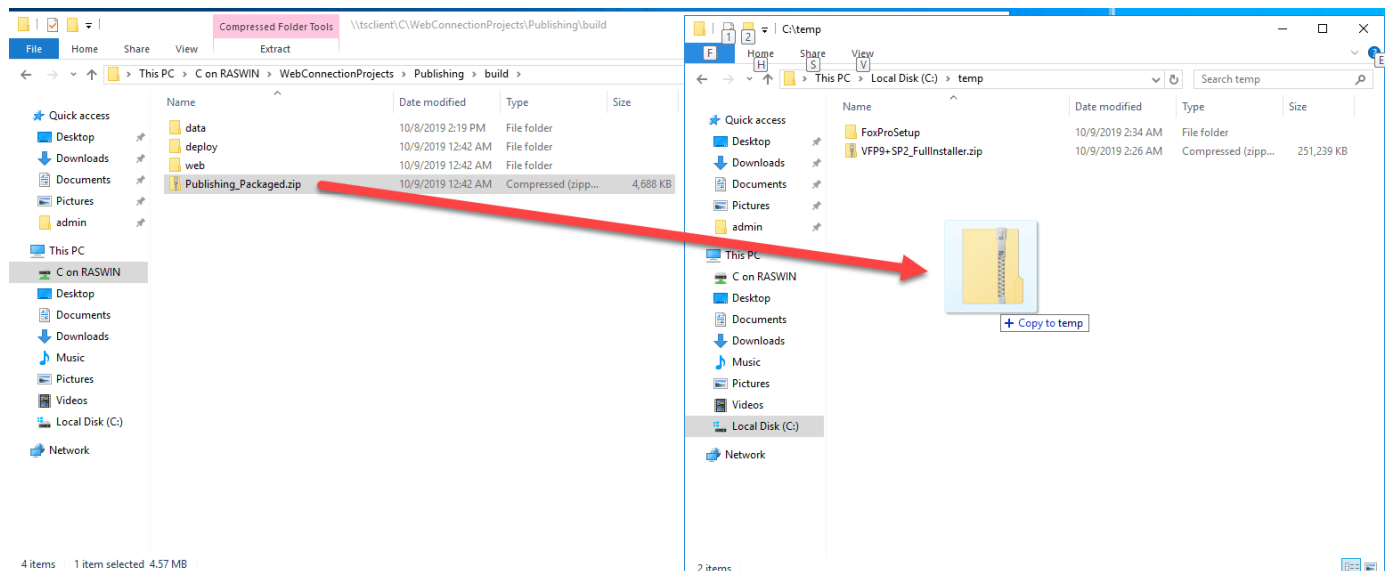
Now create a new folder structure into which to unpack the files. It can be anywhere - personally I like to use the same structure as I had on my client install so I'll put it in:

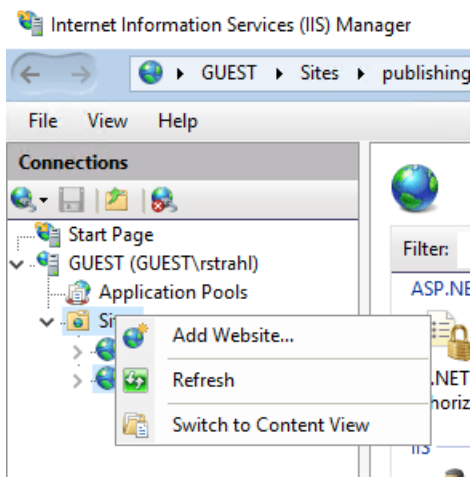```powershell
c:\webconnectionprojects\publishing
```
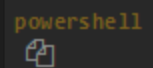
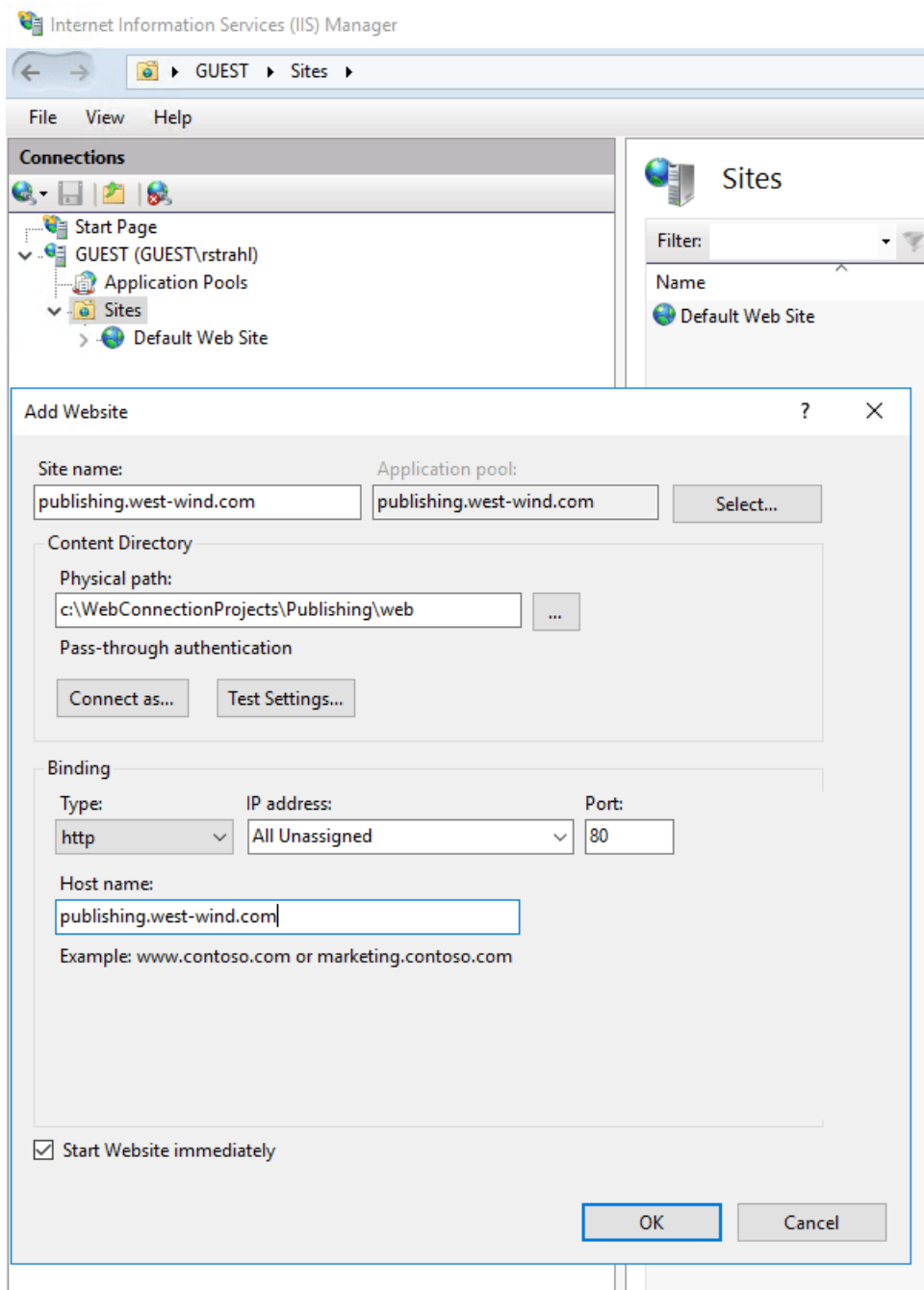This should match the structure of your local files.

## Create the IIS Web Site

This is the only manual configuration step - we need a Web site which can be configured for Web Connection.

So create a new Web Site. Open the IIS Manager:

Then create the Site by pointing it at the `web` folder of the Publishing project:

Note that I have to set a hostname for host header binding so that multiple sites can share port 80 on the server. I'm going to set up `publishing.west-wind.com` with my DNS provider at **DnSimple**, mapping the Vultr IP address to an A DNS record. This is necessary so the site can be accessed remotely on a shared port 80 with a custom domain name.

If you do this after initial creation you'll need to jump into the IIS Site's Bindings.

## Application Pool Configuration

IIS Web sites run inside of an Application Pool and that application Pool needs to be configured. While Web Connection can create a new application pool for a new virtual, for root Web sites, **the site has to be created first** and an Application Pool has to be associated with it.

This means for a new Web site, we have to manually configure the Application Pool.The Application Pool is the IIS host process for the application and it determines the environment in which the Web Connection server runs.

The only setting that really needs to be set is the **Identity** - or the user account - that the Application Pool runs under. By default this will be set to *ApplicationPool Identity* and you definitely **do not want to run with this account** as it has no rights to access the file system or anything else on the machine. It's also difficult to set permissions on resources for this account because it doesn't show up on the permissions UI.

So to fix this, go to Application Pools and open the `publishing.west-wind.com` Pool:

Find **Identity** - by default this is set **ApplicationPool Identity** and change it to another account. LocalSystem, NetworkService, or a specific user account work here, but make sure that that account has sufficient rights in the folders.

I recommend starting with LocalSystem to start, as it has full permissions on the local machine. Make sure you get your app running first, and once it's up and running and working, you can dial back the security with a specific account that you give the exact rights required to run the application.

In addition, I also recommend setting the **Enable 32-bit Applications**, which runs your Application Pool in 32 bit mode. Although 64 bit will work running Web Connection in 64 bit mode has no benefits at all and requires extra overhead in the COM calls made when running in COM mode. Additionally, 32 bit generally has lower memory requirements.

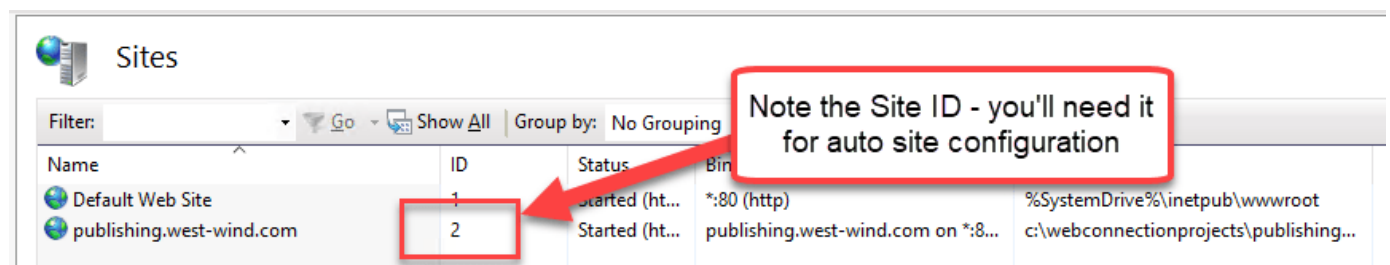## Configuring the Web Connection Web Site

Now that the site is up and configured, we still need to configure Web Connection so it's connected to this new Web Site.

To do this we can use the Configuration Feature in Web Connection. When you create a Web Connection server it has a built in Configuration script that can self configure itself for IIS by running the application with a `CONFIG` parameter from the Command Prompt.

But before you do that we need to apply the Site ID from above to the configuration settings which are stored in the app ini file - in `publishing.ini`:

```ini
[ServerConfig]
Virtual=
ScriptMaps=wc,wcs,md,pb
IISPath=IIS://localhost/w3svc/2/root
```

Note that I applied the SiteId of `2` in the `IISPath` before the `/root`. This ID is important and you can get it from the IIS Site list:

The ID ensures that our configuration run configures the correct Web Site.

Note I'm going to create a new Web site with the app running **at the root of the site**, so the `Virtual` is empty, meaning the site root is configured. The `ScriptMaps` let you specify each of the script map extensions to create in IIS - each of those extensions are routed to your Web Connection server. This should have been set up in the project originally and you likely don't have to change it, but if you need to you can add additional extensions here.

With the configuration set we can now run the `CONFIG` command and hook up our Web Connection server settings to IIS.

```powershell
.\Publishing.exe CONFIG
```

This should take 10-20 seconds or so to run as the configuration creates the virtual, configures the Application Pool, creates the scriptmaps and sets file permissions.

Once that's done you should now have a functioning Web Connection server and Web site.

> ℹ️ **You can Re-Run this Script**
>
> This configuration script can be run multiple times on a server - it won't hurt anything and will simply rewrite the settings each time it runs. It's great if you need to move an application to a new location. Simply move and re-run the CONFIG script and you're ready to go again.

Testing the Site

At this point your Application should be ready to rock n' roll!

I recommend you start in file mode, so perhaps double check your `web\web.config` file and make sure that:

```xml
<add key="MessagingMechanism" value="File" />
```

Then go to the `deploy` folder and launch your main EXE - `Publishing.exe` in this case. This starts the file based Web Server.

Now navigate to your DNS location on the local machine - or any browser:

http://publishing.west-wind.com

Assuming DNS has resolved you should be able to get to the Default page now. If this is setup for the Default Web Site and there's only one you can also use `localhost` or the machine's IP Address to navigate instead of the host name.

When the default page comes up, click on the two sample links and you should see our custom headers in the application.

Yay! Success.

Now, go to the Administration page at:

http://publishing.west-wind.com/admin/admin.aspx
* no you won't have access to mine at this address

Then to **Web Connection Module Administration** and **File → Switch** to toggle into COM mode. Go back to the home page and hit those two links again and if all goes well the application should still work the same as in file mode.

Hoorah again!

Create a Free TLS Certificate with LetsEncrypt

You may notice in the picture above that Chrome is complaining that the site is *Not secure*. This is because the site is running without HTTPS - there's no Server Certificate installed. In order to make the link display less scary, you need to install a TLS certificate so the site can run over HTTPS.

A few years ago LetsEncrypt - an consortium of various Internet Service Providers - created a certificate authority not beholden to a commercial company with the goal to provide free SSL/TLS certificates. It was a huge success, and LetEncrypt is now serving billions of certificates. This organization which is supported solely through donations and sponsorships not only has made certificates free, but also provided the tools that make it possible to completely automate the process of installing and renewing certificates.

Using these tools it literally takes two minutes or less to create a certificate and install it in IIS including a setup that auto-renews the certificate when it expires in 90 days or so.

On Windows there's an easy to use tool called **Win-Acme** that makes this process trivially simple via a command line tool.

Download the tool and copy the files into a location of your choice. I use `\utl\LetsEncrypt`. Open a command window in that location and run `.\wacs.exe` to bring up the command line based interface.

```
 Select Administrator: Windows PowerShell                                    —   □   ✕

PS C:\utl\LetsEncrypt> .\wacs

[INFO] A simple Windows ACMEv2 client (WACS)
[INFO] Software version 2.0.10.444 (RELEASE)
[INFO] IIS version 10.0
[WARN] Scheduled task not configured yet
[INFO] Please report issues at https://github.com/PKISharp/win-acme

N: Create new certificate (simple for IIS)
M: Create new certificate (full options)
L: List scheduled renewals
R: Renew scheduled
S: Renew specific
A: Renew *all*
O: More options...
Q: Quit

Please choose from the menu: n

[INFO] Running in mode: Interactive, Simple

 Please specify how the list of domain names that will be included in the
 certificate should be determined. If you choose for one of the "all bindings"
 options, the list will automatically be updated for future renewals to reflect
 the bindings at that time.

1: Single binding of an IIS website
2: All bindings of an IIS website
3: All bindings of multiple IIS websites
4: Manual input
5: Read a CSR created by another program
<Enter>: Abort

How shall we determine the domain(s) to include in the certificate?: 1

1: publishing.west-wind.com (SiteId 2)
<Enter>: Abort

Choose binding: 1

[INFO] Target generated using plugin IISBinding: publishing.west-wind.com

Enter email(s) for notifications about problems and abuse (comma seperated): rstrahl@west-wind.com

Terms of service:   C:\ProgramData\win-acme\acme-v02.api.letsencrypt.org\LE-SA-v1.2-November-15-2017.pdf

Open in default application? (y/n*)  - yes

Do you agree with the terms? (y*/n)  - yes

[INFO] Authorize identifier: publishing.west-wind.com
[INFO] Authorizing publishing.west-wind.com using http-01 validation (SelfHosting)
[INFO] Authorization result: valid
[INFO] Requesting certificate [IISBinding] publishing.west-wind.com
[INFO] Store with CertificateStore...
[INFO] Installing certificate in the certificate store
[INFO] Adding certificate [IISBinding] publishing.west-wind.com 2019/10/9 4:26:30 to store WebHosting
[INFO] Installing with IIS...
[INFO] Adding new https binding *:443:publishing.west-wind.com
[INFO] Committing 1 https binding changes to IIS
[INFO] Adding Task Scheduler entry with the following settings
[INFO] - Name win-acme renew (acme-v02.api.letsencrypt.org)
[INFO] - Path C:\utl\LetsEncrypt
[INFO] - Command wacs.exe --renew --baseuri "https://acme-v02.api.letsencrypt.org/"
[INFO] - Start at 09:00:00
[INFO] - Time limit 02:00:00
[INFO] Adding renewal for [IISBinding] publishing.west-wind.com
[INFO] Next renewal scheduled at 2019/12/3 3:26:30

N: Create new certificate (simple for IIS)
M: Create new certificate (full options)
L: List scheduled renewals
R: Renew scheduled
S: Renew specific
A: Renew *all*
```

It literally takes only a few prompts and you're ready to go with a new certificate.

Select:

- New Certificate

- Single Binding of an IIS Web Site

- Pick your new Web Site

- Agree to the terms

Let it 'er rip - and you're done! Yup that's it. Then navigate to your site via SSL.

https://publishing.west-wind.com

After you've installed a certificate we can now navigate to the site over `https://` and get a less scary browser bar:
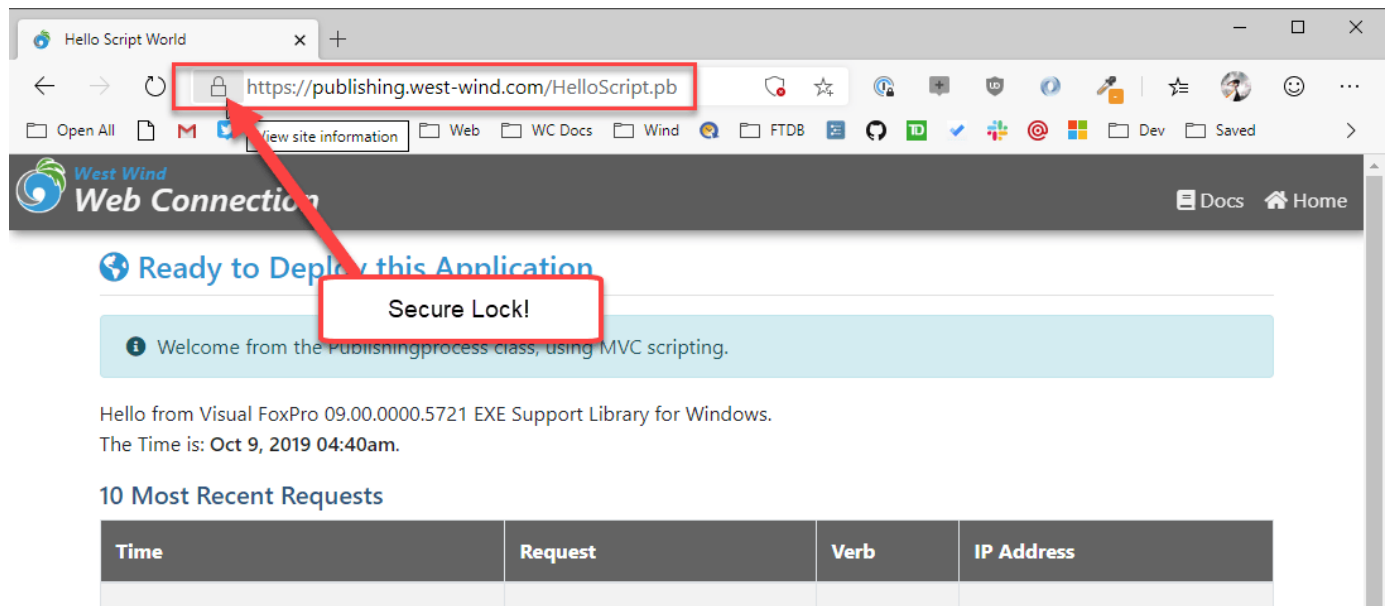
Certificates installed this way by default are automatically renewed after 90 days. Note that certificates are shorter lived, but with the auto-renewal is relatively painless to renew more often because it happens automatically.

Updating your Server and Maintenance

Once you've uploaded your site and got it all running, you're invariably going to want to change something about the site whether it's HTML content, or the actual compiled code.

Updating Web Content

There are a number of ways you can update content obviously.

- Visual Studio Web Deploy for Site, or individual Files

- Manual FTP of Files

- RDP file copy (too slow to be used on a regular basis)

Visual Studio and Web Deploy

I like to use Visual Studio proper along with its Web Deploy features, because it's directly integrated into Visual Studio and super easy to push up either individual files or the entire site.

You can view Web Connection Web projects quite nicely in Visual Studio using the `Web Site Projects` . This is an old project type so it's a bit out of the way now.

Go to File → Open → Web Site which opens the site in Visual Studio. You can now edit files and make changes using Visual Studio.

When you're ready to publish or update files right click on the project node:

- Select Publish Web Application

- Publish to IIS

- Fill in the info for the Dialog
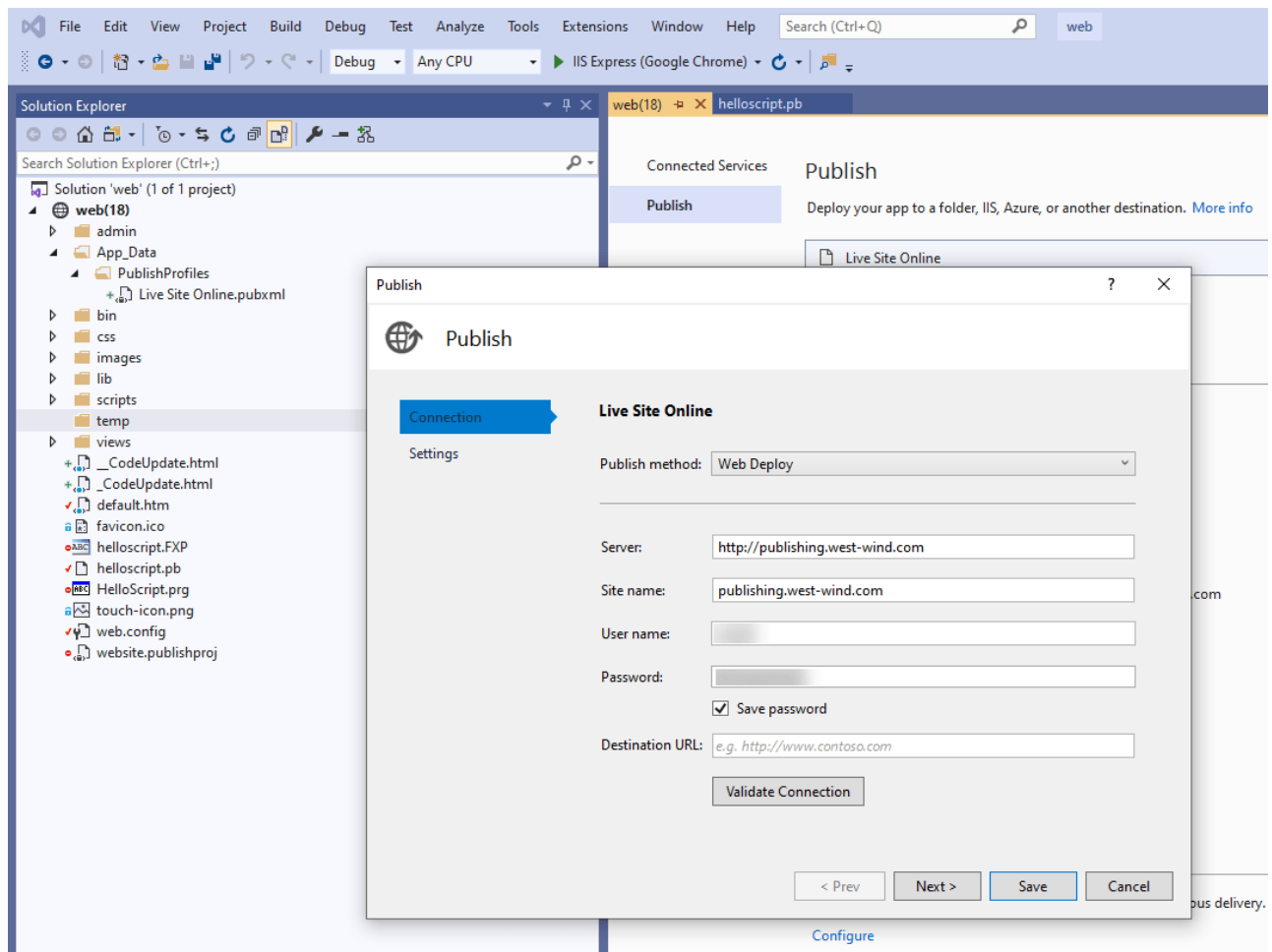
Note that this **requires WebDeploy** which I installed as part of the IIS installation earlier. If WebDeploy is not installed on the server you're likely going to see long hangs and timeouts here.

Once set up you can now publish either the entire site, or individual files. Right click on the project to publish the full site, or right click on a file to publish just that file.

One really nice thing about this tool is that it is smart and compares what's on the server to what's on the client and only updates what's changed. Even if you publish the full site but only changed 1 or 2 files, only those files plus some meta data is sent to the server. This makes Web Publish very efficient and fast. I often publish individual files using the default hot key `Alt-;-P` which isn't very intuitive but because I use it so much I have muscle memory from it 🙂.

> ### ℹ Using Web Deploy for Generic Files
>
> You can also use WebDeploy to send up other files. For example, if it turns out you need new versions of the Web Connection support DLLs you can zip them up and upload them into the Web site temporarily. You can then RDP into the server pick up the zip file and swap ou the DLLs. The same works for any other resources files.

You can do pretty much the same thing if you have an FTP server installed and if you transfer lots of files to the server all the time a dedicated FTP server is more flexible than WebDeploy and it's close ties to Microsoft tools. FTP works with anything, but it's beyond the scope of this session to talk about setting up an FTP server.

Update your Server

Web Connection includes some tools that can let you automatically publish your updated EXE FoxPro Server by uploading it and efficiently hot-swapping it. You can do this without shutting down the Web Server.

The process is:

- Update your EXE

- Navigate to the Server Module Administration Page

- Use Upload Server Exe to upload the EXE

- Use Update Server Exe to hotswap the uploaded EXE

Here are the links on the Module Administration page for uploading and updating:

The process works by uploading a new server executable which is named on the server as `YourExe_Update.exe` extension and once uploaded then hotswapping

Click on **Upload Server Exe** to upload your new compiled EXE server to the server.

Once uploaded, click the **Update Server Exe** button to then hot-swap the server. This link will shut down all running server instances, put the server on hold so new requests start reloading the Web Connection server instances, and then copies the `_Update.exe` to actual server EXE. This routine also re-registers the COM object so if there are changes in the COM interface they are reflected in this update. All the servers are then restarted.

This process typically takes a couple of seconds, depending on how many server instances you have running and how fast they are to start up.

Automatic Updating via `bld_yourProject`

This manual process can also be automated directly from your FoxPro Web Connection project. When a new project is created Web Connection creates a custom PRG file that builds the Web Connection application into an EXE. Optionally you can pass a parameter of `.t.` to this function which causes it to build and then publish to the server.

Before you can do this you need to edit the generated `bld_publishing.prg` file and change the URLs for the online project. By default the URL points to `localhost` and you need to change this to point at the actual live,

deployed site instead:

```foxpro
*** Server Update Urls - fix these to point at your production Server/Virtual
HTTP_UPLOADURL      =             "http://publishing.west-wind.com/UploadExe.wc"
HTTP_UPDATEURL      =             "http://publishing.west-wind.com/UpdateExe.wc"
```

Then you can now simple run:

```foxpro
DO bld_Publishing with .T.
```

You'll get prompted for username and password, and if a valid pair is entered your EXE file is uploaded and hotswapped on the server.

Summary

Alright - there you have it. We've gone from creating a new application, creating a brand new **Vultr Virutal Machine**, configuring it, setting up IIS and a new Web Site, doing a first time publish, configuration of the Web server, then installing the application and running the application. Finally we updated the application with a new version.

You've seen:

- Setting up a Server 2012
- Installing System Applications
- Setting up IIS
- Setting up a Web Site and Application Pool
- Packaging your Application
- Publishing your Application
- Configuring your Application on the Server
- Testing your Application
- Running the application
- Installing an SSL Certificate
- Updating your Application Web files
- Updating your Application Executable

Full circle. □

You now have all you need to know to publish your Web Connection applications successfully.

Resources

- **Session Materials**
- **Vultr deal: $50 credit**
- **Win-Acme LetsEncrypt SSL Installation**