

XPlatformCloudKit

RemoteAppSettings Documentation



The RemoteAppSettings service allows for any settings, changes, or additions to be retrieved from a remote location. Effectively, maintenance can be performed by the developer, on-the-fly, with little to no intervention by the end user.

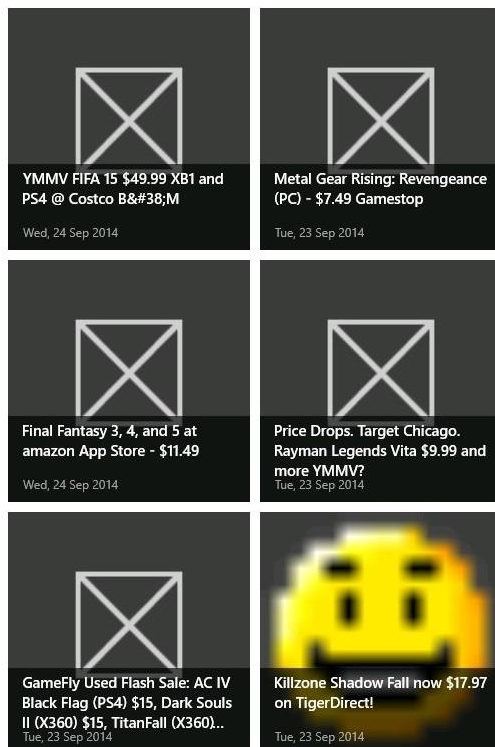
The Good

At first glance, the idea of RemoteAppSettings sounds complex, but it's simple and straight-forward to implement. It only needs but a few steps in order to get it up and running. Open up the **XPlatformCloudKit** solution in **Visual Studio**. In the **Solution Explorer**:

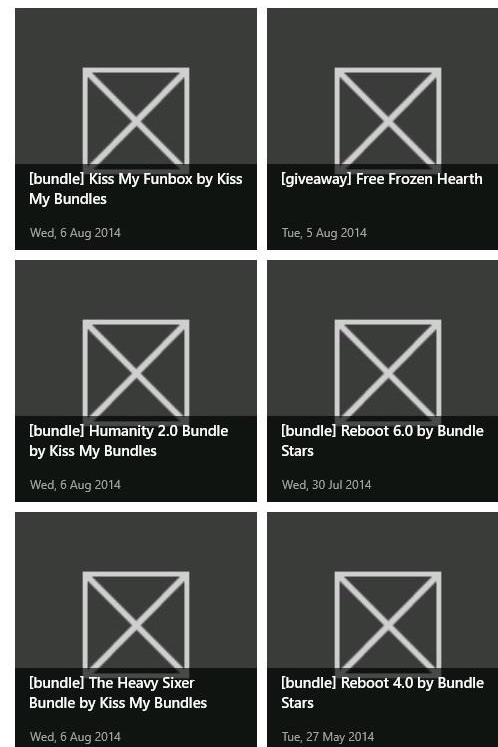
1. Expand the **XPlatformCloudKit.PCL** project and customize your app in **AppSettings.cs**. I'm a cheap gamer, so here's my example that retrieves RSS feeds from CheapAssGamer.com and IsThereAnyDeal.com.

Frugal Gamer

From Cheap Ass Gamer



From IsThereAnyDeal?



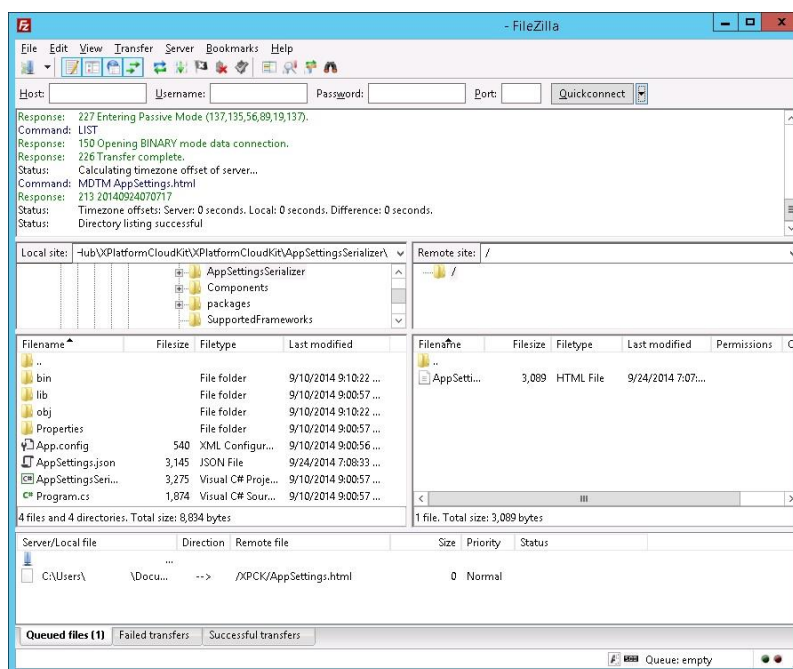
- We need to convert our **AppSettings.cs** data into a JSON file. In the **Solution Explorer**, right click the **AppSettingsSerializer** project, select **Set as Startup Project** from the context menu and then **Run**. A console window will appear similar to the one below. The JSON file is located in the project's root directory. You can get there by right clicking the **AppSettingsSerializer** project in the **Solution Explorer** and selecting **Open in File Explorer** from the context menu.

```

"ForceYoutubeVideosToLoadFullScreen": true,
"MaximizeYoutubeVideos": true,
"EnableTwitterService": false,
"TwitterConsumerKey": "",
"TwitterConsumerSecret": "",
"TwitterAccessToken": "",
"TwitterAccessSecret": "",
"TwitterAddressCollection": [
  {
    "Url": "https://api.twitter.com/1.1/statuses/user_timeline.json?user
    id=pjdecarlo",
    "Group": "PJDeCarlo",
    "Type": null
  }
],
"EnableRemoteUrlSourceService": false,
"RemoteUrlSourceUrl": "",
"EnableRemoteAppSettings": false,
"RemoteAppSettingsService": ""
}

*Completed*
See: ..\..\AppSettings.json for formatted output - Press a key to continue
  
```

- Upload the resulting JSON file to your remote source. I'm using FileZilla to upload the JSON file to my web server. **Be sure to rename the JSON file on the remote source as a HTML file.**



4. Back to **AppSettings.cs**, set **EnableRemoteAppSettings** to **true**.

```
#region RemoteAppSettings
//Allows for using a remote AppSettings file - meaning any setting, changes, or additions can be retrieved remotely
//To use: First make your baked in modifications to AppSettings.cs as normal
//Next, run the included AppSettingsSerializer project and host the result of AppSettings.json on a remote server
//Set EnableRemoteAppSettings to true
//Point to the remote Url which contains your AppSetting.json output in RemoteAppSettingsService

public static bool EnableRemoteAppSettings = true;

//public static string RemoteAppSettingsService = "http://pjdecarlo.com/playground/XPCKSampleRemoteAppSettings/AppSettings.html";
public static string RemoteAppSettingsService = "";
#endregion
```

5. Set **RemoteAppSettingsService** to point to the remote location that contains the JSON file.

```
#region RemoteAppSettings
//Allows for using a remote AppSettings file - meaning any setting, changes, or additions can be retrieved remotely
//To use: First make your baked in modifications to AppSettings.cs as normal
//Next, run the included AppSettingsSerializer project and host the result of AppSettings.json on a remote server
//Set EnableRemoteAppSettings to true
//Point to the remote Url which contains your AppSetting.json output in RemoteAppSettingsService

public static bool EnableRemoteAppSettings = true;

public static string RemoteAppSettingsService = "http://pjdecarlo.com/playground/XPCKSampleRemoteAppSettings/AppSettings.html";
//public static string RemoteAppSettingsService = "";
#endregion
```

6. Deploy your app!

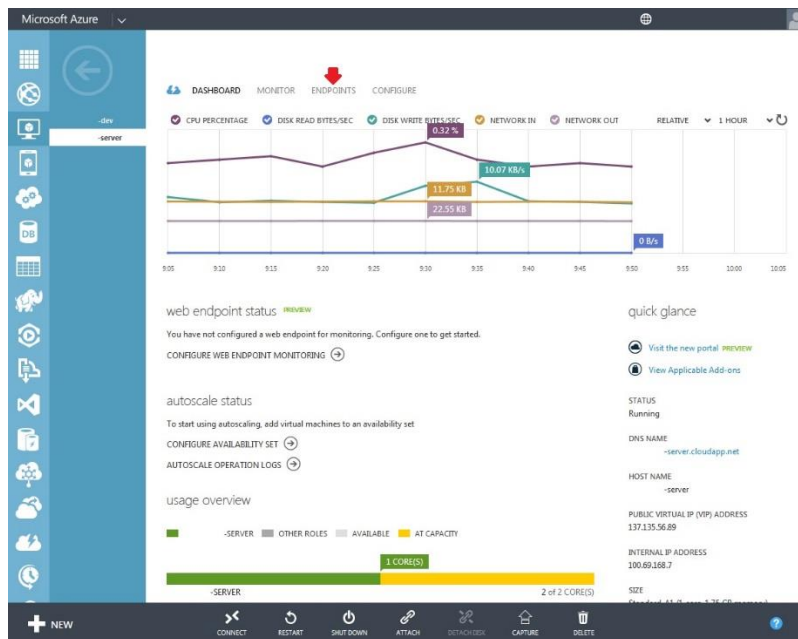
Sounds simple, right? Unfortunately, in order to test this, we need an available web server to host the JSON file. You could use one of the many free or paid web hosting services available. However, to be as meticulous as possible, we will be setting up a web server with passive FTP in a Microsoft Azure VM.

The Bad

Before we get started, we need to create an Azure VM. Use the **Windows Server 2012 R2 Datacenter** image from the gallery. Take note of the VM's **DNS Name** (*.cloudapp.net) you gave it as you will be using it later.

More on the next page.

First, we need to add some endpoints to the VM. In the **Azure Management Portal**, navigate to **Virtual Machines** and select your newly created VM. This should bring up its **Dashboard**. Click on **Endpoints** in the row near the top.



Click on **Add** in the bottom row and a popup window will appear. Click on **Next** (the arrow) in the bottom right corner leaving the default selection for the first page. In the **Name** dropdown box, select **FTP** and then click on **Complete** (the checkmark). It will take a few moments to add the endpoint to the VM. Do the same for **HTTP** and **HTTPS**.

For passive FTP, we also need to add some custom endpoints for the FTP data channel. Alongside the ones we configured earlier, I added 6 additional endpoints, ports 5000-5005. Repeat the process earlier but instead give your custom endpoints a **Name**, e.g. FTP 5000. Configure the **Public Port** and the **Private Port** with the port number you've chosen. Do the same for your other custom endpoints.

ADD ENDPOINT

Specify the details of the endpoint

NAME
FTP 5000

PROTOCOL
TCP

PUBLIC PORT
5000

PRIVATE PORT
5000

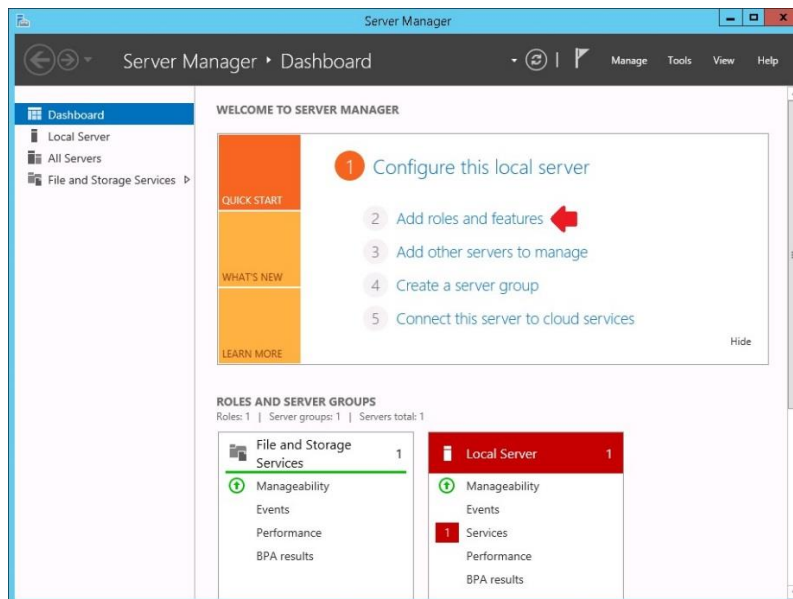
☐ CREATE A LOAD-BALANCED SET ?

☐ ENABLE DIRECT SERVER RETURN ?

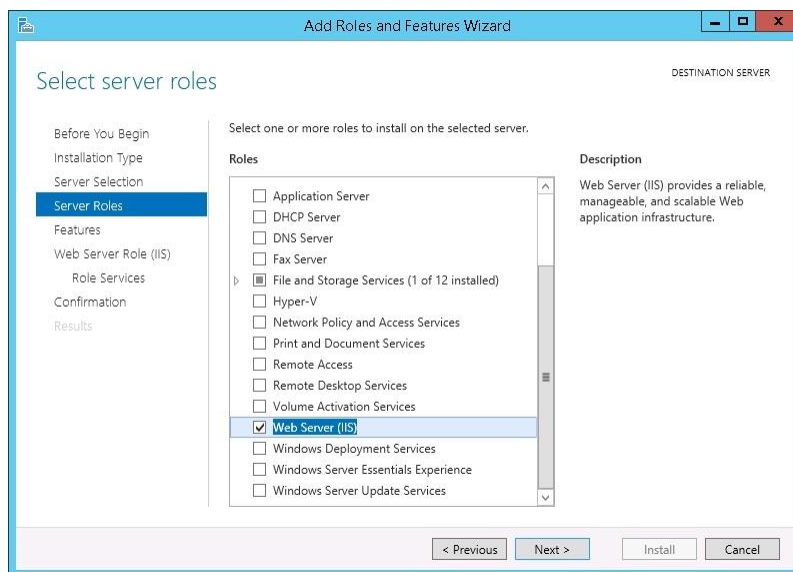
Once you're done you should see something similar to this on your **Endpoints** page. So far we've configured an **FTP** endpoint, **HTTP** and **HTTPS** endpoints, and the FTP data channel endpoints.

NAME	PROTOCOL	PUBLIC PORT	PRIVATE PORT	LOAD-BALANCED SET NAME
FTP	TCP	21	21	-
FTP 5000	TCP	5000	5000	-
FTP 5001	TCP	5001	5001	-
FTP 5002	TCP	5002	5002	-
FTP 5003	TCP	5003	5003	-
FTP 5004	TCP	5004	5004	-
FTP 5005	TCP	5005	5005	-
HTTP	TCP	80	80	-
HTTPS	TCP	443	443	-
Powershell	TCP	5986	5986	-
Remote Desktop	TCP	62560	3389	-

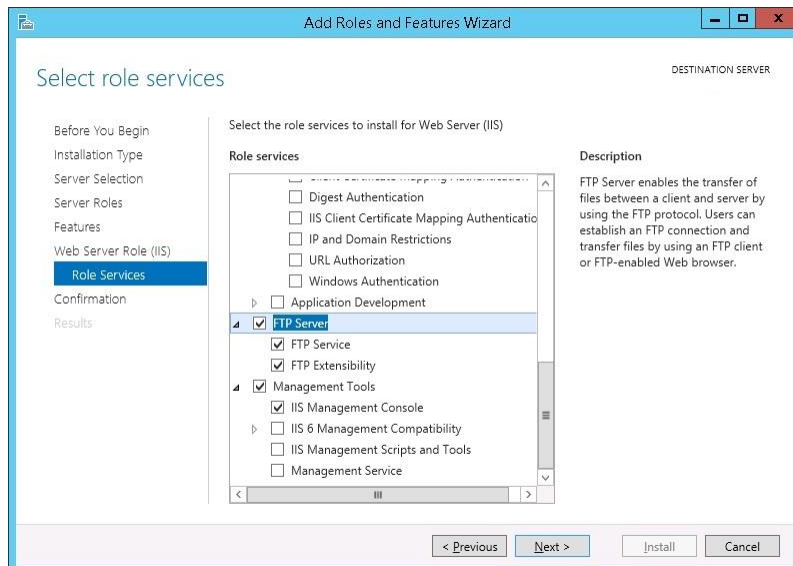
Next, we need to add web server functionality, specifically IIS (Internet Information Services). **Connect** to the VM and run or wait for the **Server Manager** to appear. Click on **Add roles and features**.



The **Add Roles and Features Wizard** will appear. Click **Next** leaving the default selection until you reach **Server Roles**. Scroll down and check **Web Server (IIS)**. A **popup window** will show asking you confirm the addition of the role plus the IIS Management Console. You will be using it later. Click **Add Features**.

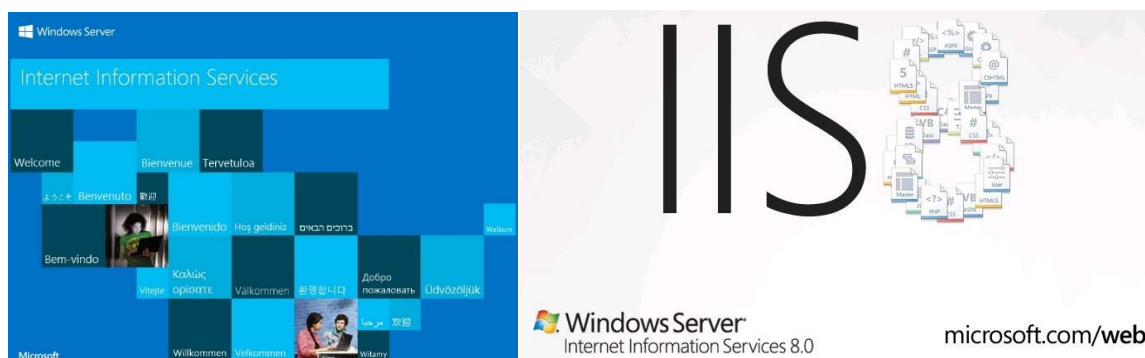


Click **Next** leaving the default selection until you reach **Role Services**. We will need to add the FTP server role. With FTP, you can upload the JSON file from your development machine to the remote source. On the flip side, HTTP will provide a way for the XPCCK app to retrieve its settings. Make sure both **FTP Service** and **FTP Extensibility** are checked.



Click **Next** which should bring you to the Confirmation screen. I assume everything is okay, so click **Install**. Wait for it to finish installing. Restart the VM, if required.

At this point, you should check to see if the web server is functioning, at least the HTTP part of it. On your local machine, open a web browser and navigate to the VM's **DNS Name** (*.cloudapp.net). You should be greeted with a page like either one below.

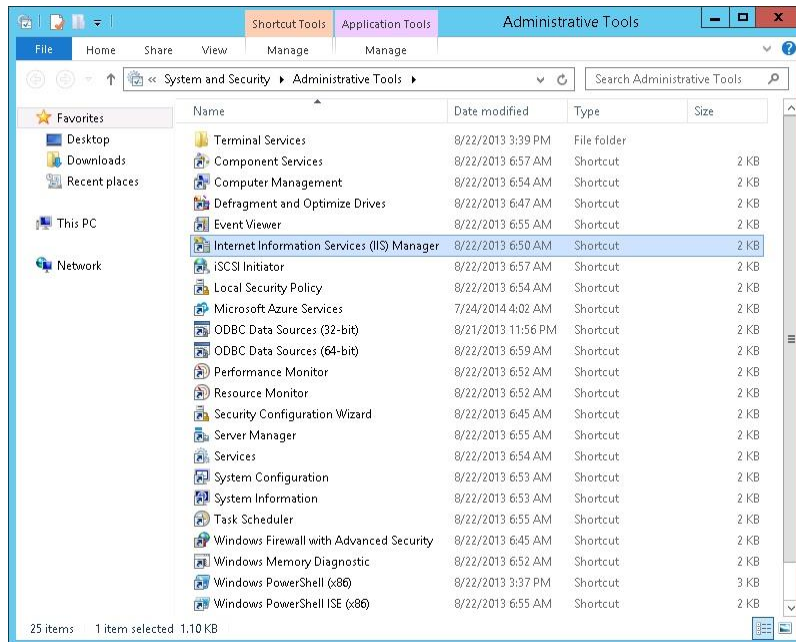


If you can see either one of these pages, you're halfway there! If not, make sure you've followed each step up until this point. You may have missed something.

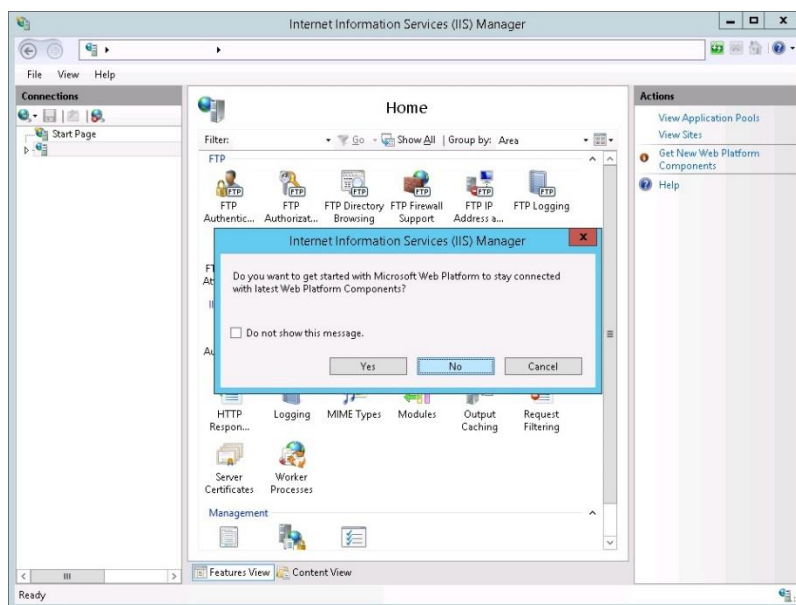
The Ugly

Even though your web server is up and running, we still need to configure the FTP server. This will require quite a bit of work since we're using passive FTP.

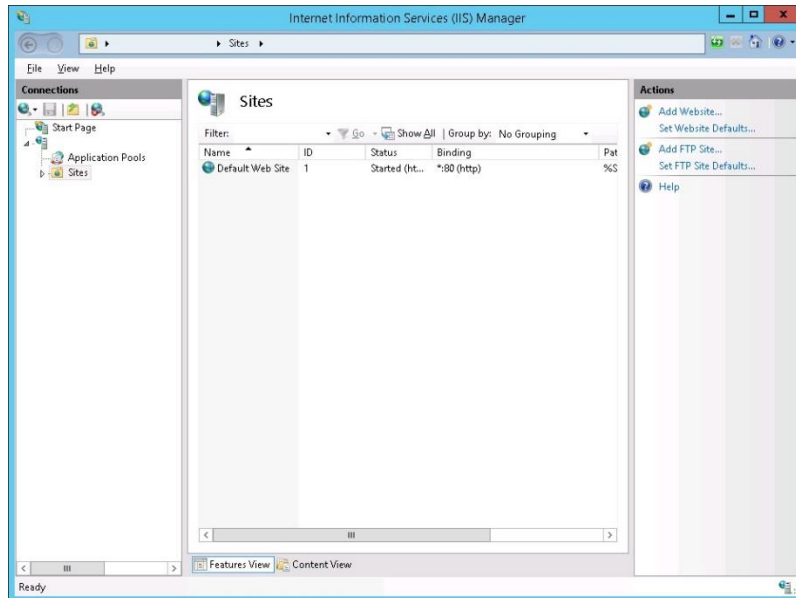
On your VM's desktop, right click the **Windows button** (Start button). A context menu will appear. Click **Control Panel**. Navigate to **System and Security > Administrative Tools**. Click on **Internet Information Services (IIS) Manager**.



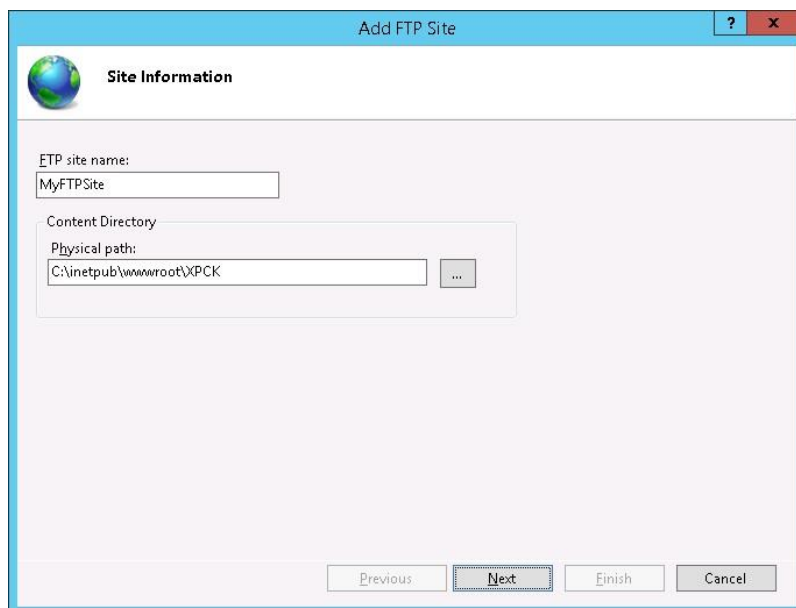
On the left hand side under **Connections**, click the server's *NAME*. It should be the only listing besides the *Start Page*. You may be greeted with a popup window asking if you want to stay connected with the latest Web Platform Components. Click **No**.



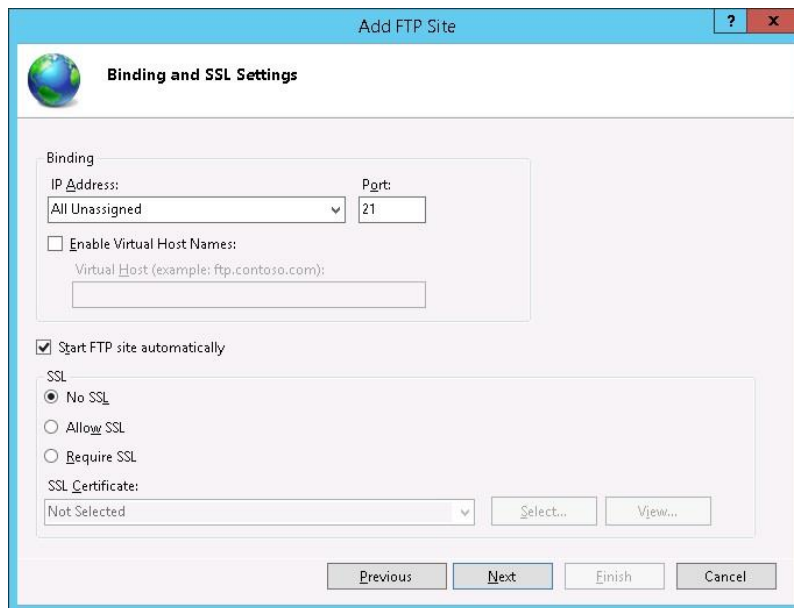
On the right hand side under **Actions**, click **View Sites**. You should now see a listing of the sites the server currently hosts. Right now there is only the default HTTP site that we saw earlier. Scroll to the right and take note of the **Path**. You'll find the page's HTML file and its accompanying image asset there. We want the FTP site to point to the same directory.



On the right hand side under **Actions**, click **Add FTP Site...** Give your FTP site a name and have it point to `%SystemDrive%\inetpub\wwwroot`. This is the same directory as the one *Default Web Site* is pointing to. Remember, any files or folders in this directory can be viewed over the web. To keep things tidy, I had my FTP site point to a subdirectory called *XPCK*. Click **Next**.

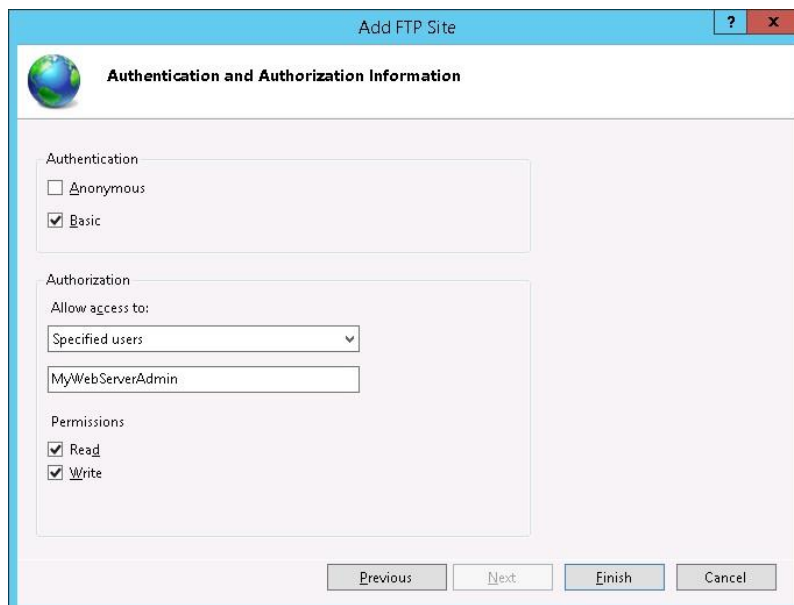


Make sure **SSL** is not set to **Require SSL**. Set it to **No SSL** if you don't want SSL or else set it to **Allow SSL**. Click **Next**. It may prompt you to add a SSL certificate. In the **SSL Certificate** list box, select the server's certificate and then try again.



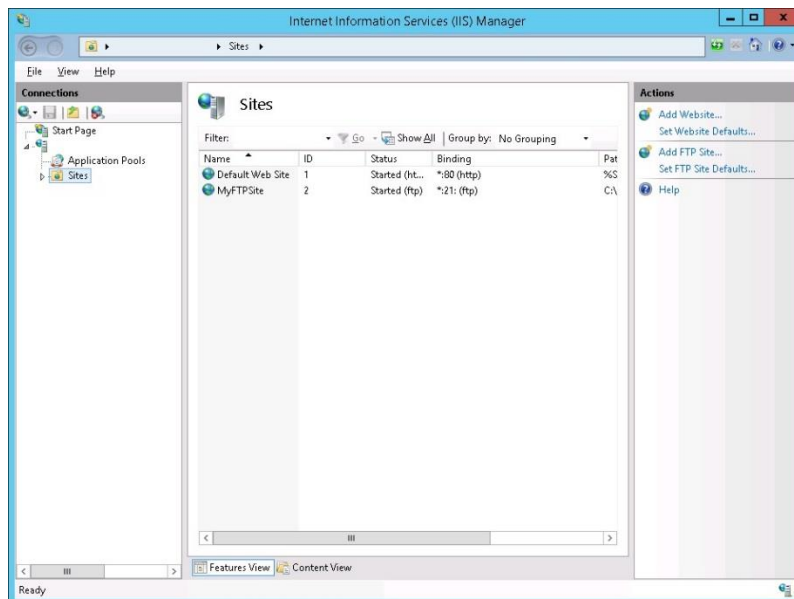
The screenshot shows the 'Add FTP Site' wizard at the 'Binding and SSL Settings' step. The 'Binding' section has 'IP Address' set to 'All Unassigned' and 'Port' set to '21'. The 'Enable Virtual Host Names' checkbox is unchecked. The 'Start FTP site automatically' checkbox is checked. In the 'SSL' section, 'No SSL' is selected. The 'SSL Certificate' dropdown is set to 'Not Selected'. Navigation buttons at the bottom include 'Previous', 'Next', 'Finish', and 'Cancel'.

Now we will configure who has access to the FTP site. In the **Authentication** group box, check **Basic**. In the **Authorization** group box, set **Allow Access to:** list box to **Specified Users**. Fill out your VMs built-in administrator account name (not recommended), a local administrator or user account name in the text box below. Under **Permissions**, check both **Read** and **Write**. Click **Finish**.

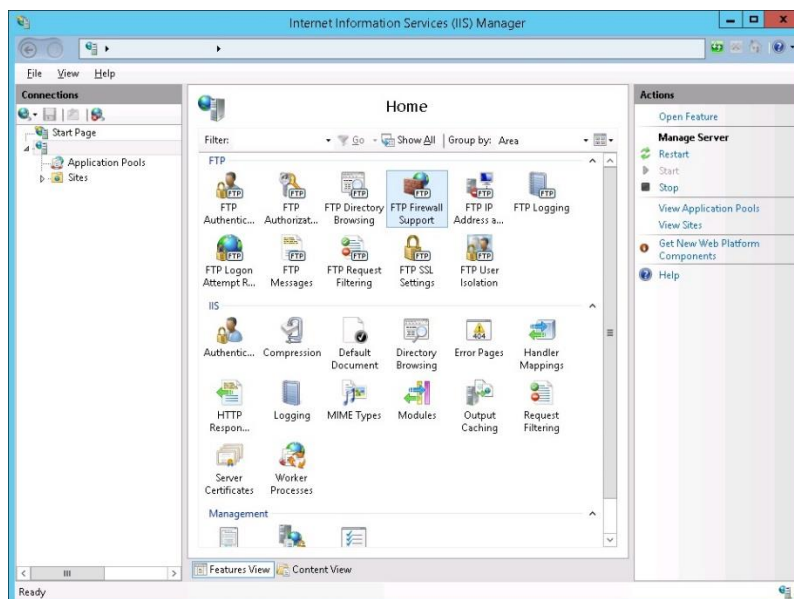


The screenshot shows the 'Add FTP Site' wizard at the 'Authentication and Authorization Information' step. In the 'Authentication' section, 'Basic' is checked. In the 'Authorization' section, 'Allow access to:' is set to 'Specified users' and the text box contains 'MyWebServerAdmin'. In the 'Permissions' section, both 'Read' and 'Write' are checked. Navigation buttons at the bottom include 'Previous', 'Next', 'Finish', and 'Cancel'.

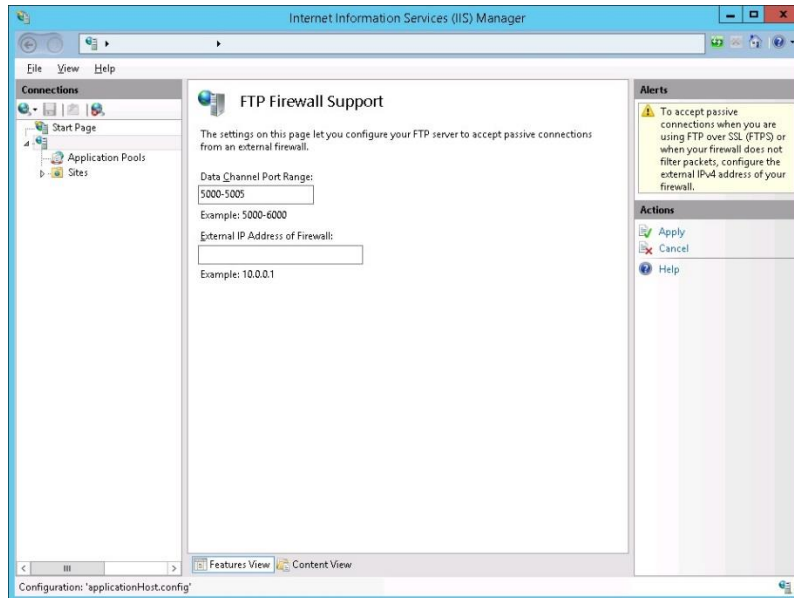
Your new FTP site should appear on the **Sites** listing. You should now have local connectivity through the loopback interface. However, if you were to browse the site using FileZilla or the ftp command on your local machine or another VM, the connection will timeout. You'd be right to think it's the firewall.



Navigate back to the server's home page under **Connections**. In the **FTP** group, Click on **FTP Firewall Support**.



We will need to specify the data channel ports that will be used for passive FTP. In the **Data Channel Port Range** textbox specify a Port Range. In this case, I have used 5000-5005. **The External IP Address of Firewall** is the virtual IP (VIP) of the VM. You should be able to find it in the **Azure Management Portal** in your VM's **Dashboard**. Under **Actions**, click **Apply**.



We're almost there. Although the firewall seems to allow all traffic that's required, you also need to enable stateful FTP filtering on the firewall.

```
netsh advfirewall set global StatefulFtp enable
```

Finally, restart the FTP service and we should be up and running.

```
net stop ftpsvc  
net start ftpsvc
```

As shown in **Command Prompt (Administrative)**.



```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

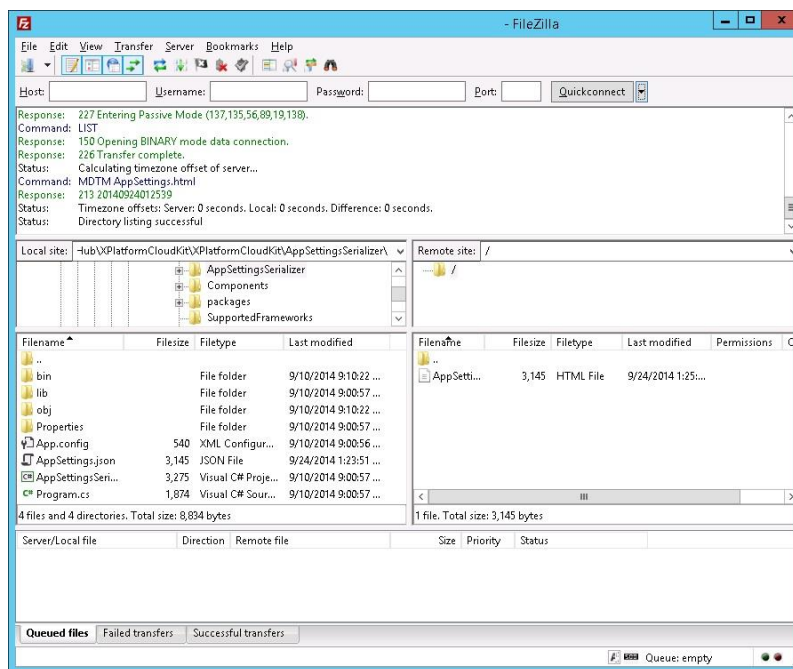
C:\Windows\system32>netsh advfirewall set global StatefulFtp enable
Ok.

C:\Windows\system32>net stop ftpsvc
The Microsoft FTP Service service is stopping.
The Microsoft FTP Service service was stopped successfully.

C:\Windows\system32>net start ftpsvc
The Microsoft FTP Service service is starting.
The Microsoft FTP Service service was started successfully.

C:\Windows\system32>_
```

Testing with FileZilla confirms that we can now successfully connect to our new FTP site, hosted on a Microsoft Azure VM.



You can try this out by downloading and running FileZilla. Near the top of the window, enter the server's **DNS Name** (*.cloudapp.net) as the **Host**, the user credentials you allowed for the FTP site under **Username** and **Password**, and the port number of the **FTP** endpoint we configured earlier (in this case, port 21). Click on **Quickconnect**. In the status window, you should see "*Status: Directory listing successful.*" In the left explorer window, you'll see your local or development machine (depending on which one you used to connect to the server) and, in the right explorer window, the FTP site.

If you want to be sure you're connecting to your remote source via passive FTP, in the status window, look for the line "*Response: 227 Entering Passive Mode (...).*" Enclosed in brackets are a bunch of comma-separated numbers. Take note of the last two numbers.

Perform the following operation. In my case, those two numbers are 19 and 138.

$$256 * 19 + 138 = X$$

$$X = 5002$$

This value should correspond with one of the custom endpoints you specified earlier. If it matches one of them, you're using passive FTP. If not, you're on your own!

We're done! Give yourself a pat on the back! This will give you a line of communication from your development machine to your remote source. With this, you can easily push updates to your XPCK app and thus to your end users.