

Create Your First Application: Node.js and Windows Azure

Windows Azure Developer Center

Quick Step-by-Step



Microsoft®

Create Your First Application: Node.js and Windows Azure

Windows Azure Developer Center

Summary: Create your first application using Node.js and Windows Azure.

- Implement a simple Hello World application in Node.js and deploy the application to a Windows Azure Web Site.
- Learn how to use the Windows Azure PowerShell cmdlets to create a Node.js application, test it in the Windows Azure Emulator, and then deploy it as a Windows Azure Cloud Service.
- Implement a Node.js application using WebMatrix, and then deploy it to a Windows Azure web site.
- Implement a task list application using Node.js and MongoDB.

Category: Quick Step-by-Step

Applies to: Windows Azure, Node.js

Source: Windows Azure Developer Center ([link to source content](#))

E-book publication date: June 2012

Copyright © 2012 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Microsoft and the trademarks listed at

<http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx> are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Contents

Create and deploy a Node.js application to a Windows Azure Web Site.....	5
Set up the Windows Azure environment	5
Create a Windows Azure account	5
Enable Windows Azure Web Sites	6
Create a Windows Azure Web Site and enable Git publishing.....	7
Install developer tools.....	11
Build and test your application locally.....	12
Publish your application	13
Publish changes to your application.....	15
Revert to a previous deployment.....	16
Next steps	17
Additional Resources	17
Node.js Cloud Service.....	18
Setting Up the Development Environment.....	18
Creating a New Node Application.....	19
Running Your Application Locally in the Emulator.....	23
Deploying the Application to Windows Azure.....	24
Creating a Windows Azure Account	24
Downloading the Windows Azure Publishing Settings	25
Publishing the Application.....	26
Stopping and Deleting Your Application	28
Create and deploy a Node.js application to a Windows Azure Web Site using WebMatrix.....	30
Set up the Windows Azure environment	31
Create a Windows Azure account	31
Enable Windows Azure Web Sites	31
Create a Windows Azure Web Site.....	33
Import the web site into WebMatrix and apply the Express template	34
Publish your application to Windows Azure.....	37
Modify and republish your application.....	40
Next Steps.....	42
Node.js Web Application with Storage on MongoDB	43
Prerequisites.....	45
Preparation	45
Create a virtual machine and install MongoDB	45
Sign up for the Windows Azure Web Sites preview feature	45
Enable Windows Azure Web Sites	46
Install modules and generate scaffolding.....	47
Install express and generate scaffolding	48
Install additional modules.....	49

Using MongoDB in a node application.....	50
Create the model.....	50
Create the controller.....	50
Modify app.js	52
Modify the index view	52
Run your application locally.....	53
Deploy your application to Windows Azure	56
Install the Windows Azure command-line tool for Mac and Linux.....	57
Import publishing settings.....	57
Create a Windows Azure Web Site.....	58
Publish the application.....	59
Next steps	60
Additional resources	60
Node.js Web Application using the Windows Azure SQL Database	61
Prerequisites.....	62
Enable the Windows Azure Web Site feature.....	62
Create a web site with database.....	63
Get SQL Database connection information.....	71
Design the task table	72
Install modules and generate scaffolding.....	75
Install express and generate scaffolding	75
Install additional modules.....	77
Use SQL Database in a node application	78
Modify the controller.....	78
Modify app.js	79
Modify the index view	80
Modify the global layout.....	81
Create configuration file.....	82
Run your application locally.....	82
Deploy your application to Windows Azure	84
Publish the application.....	84
Switch to an environment variable.....	84
Next steps	86
Additional resources	86

Create and deploy a Node.js application to a Windows Azure Web Site

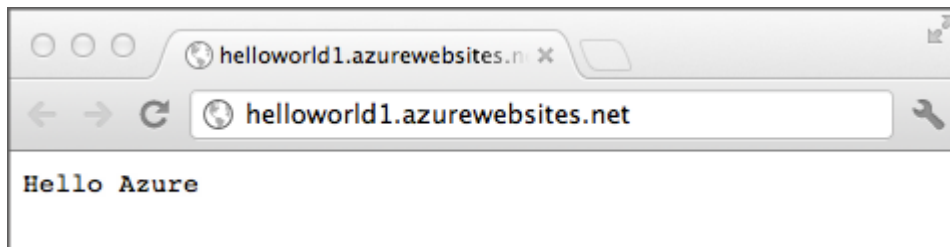
This tutorial shows you how to create a [node](#) application and deploy it to a Windows Azure Web Site using [Git](#). The instructions in this tutorial can be followed on any operating system that is capable of running node.

You will learn:

- How to create a Windows Azure Web Site using the Windows Azure Developer Portal
- How to publish and re-publish your application to Windows Azure using Git

By following this tutorial, you will build a simple Hello World web application in Node.js. The application will be hosted in a Windows Azure Web Site when deployed.

A screenshot of the completed application is below:

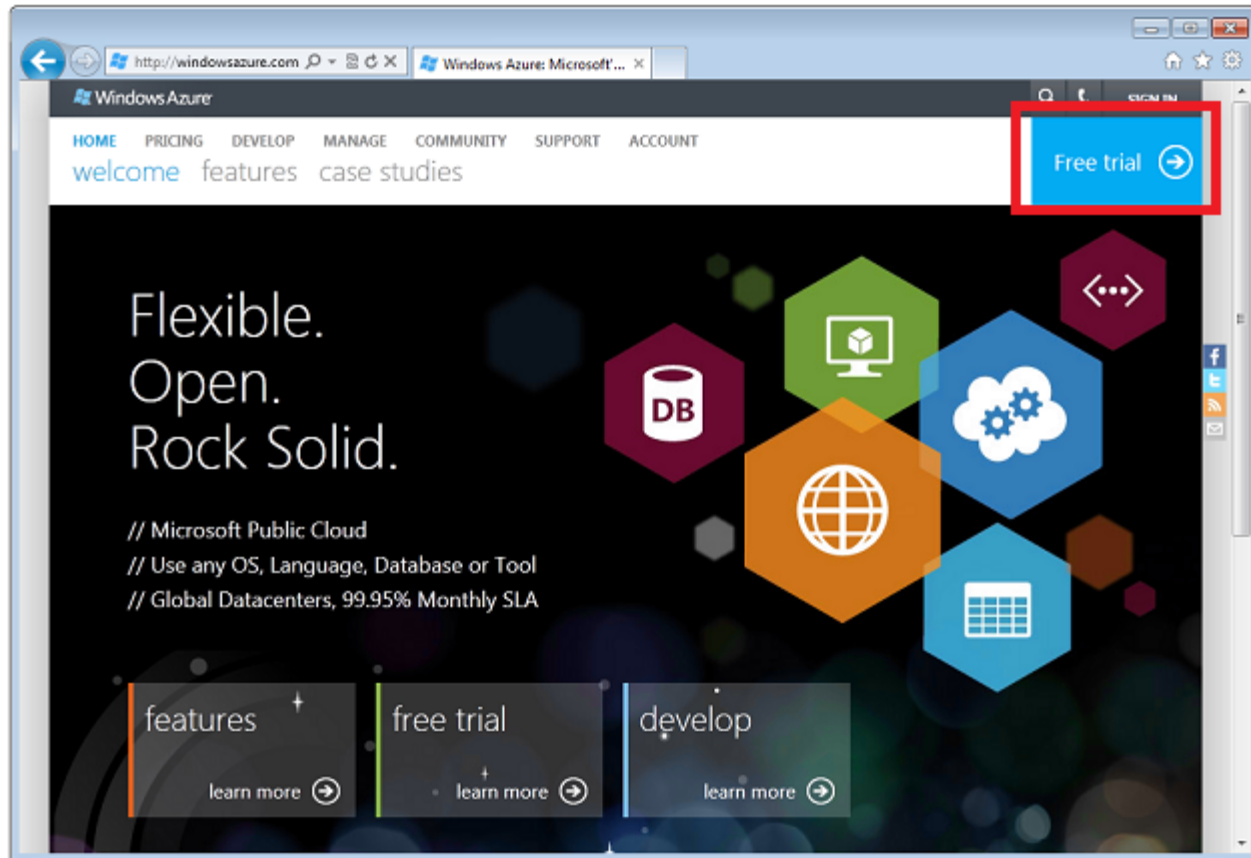


Set up the Windows Azure environment

First, set up the Windows Azure environment. You'll create a Windows Azure account and enable this account to use the Windows Azure Web Sites preview feature.

Create a Windows Azure account

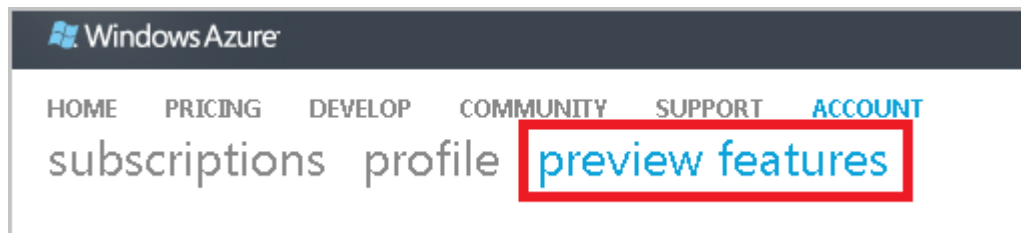
1. Open a web browser and browse to <http://www.windowsazure.com>.
2. To get started with a free account, click **Free Trial** in the upper-right corner and follow the steps. You'll need a credit card number and a mobile phone number for proof of identity, but you will not be billed.



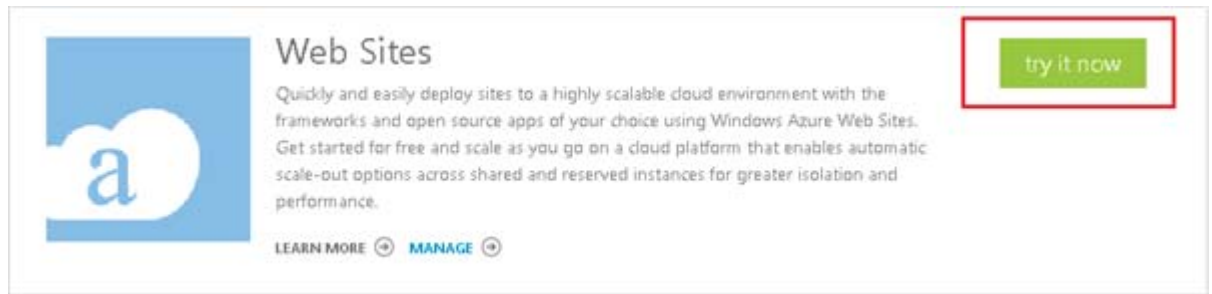
Enable Windows Azure Web Sites

After signing up, follow these steps to enable the Windows Azure Web Site feature.

1. Navigate to <https://account.windowsazure.com/> and sign in with your Windows Azure account.
2. Click **preview features** to view the available previews.



3. Scroll down to **Web Sites** and click **try it now**.



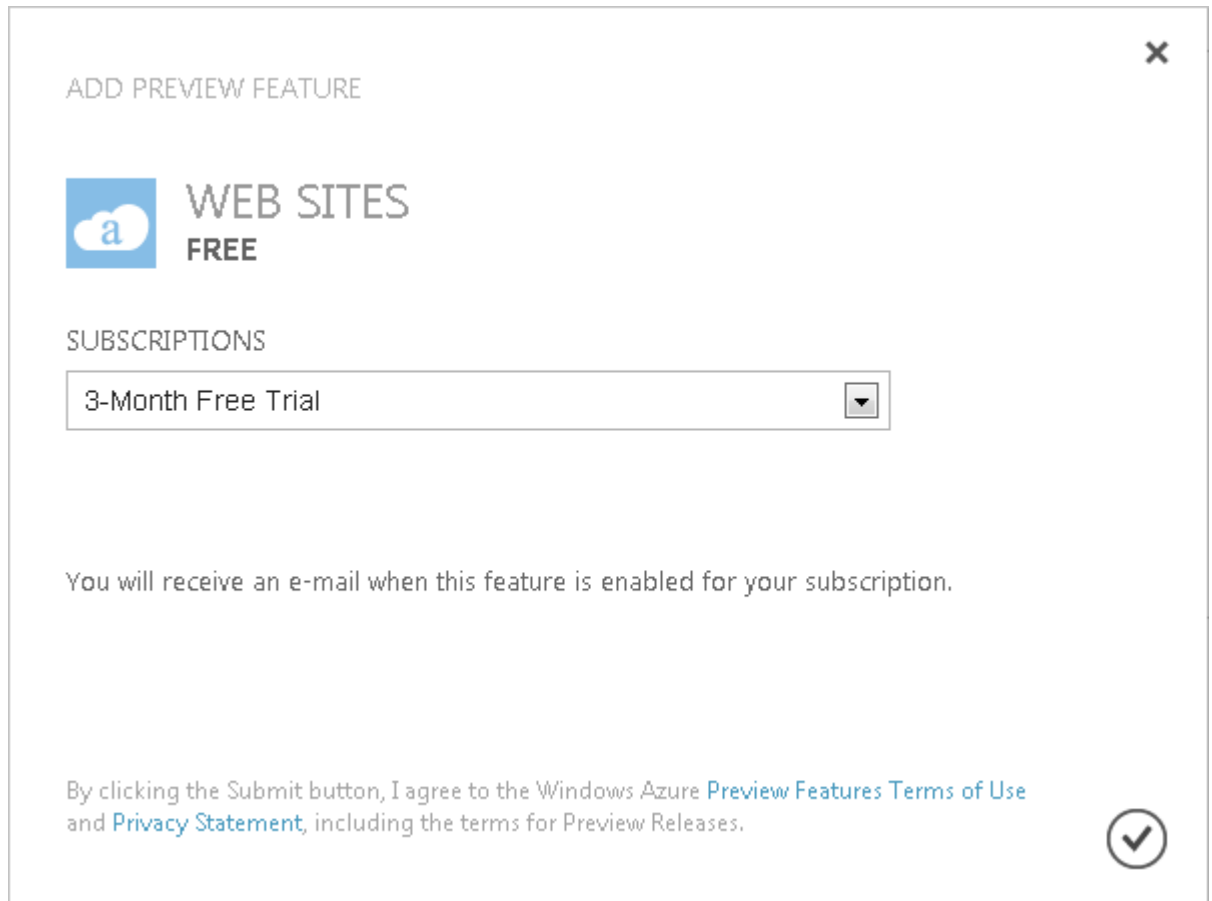
Web Sites

Quickly and easily deploy sites to a highly scalable cloud environment with the frameworks and open source apps of your choice using Windows Azure Web Sites. Get started for free and scale as you go on a cloud platform that enables automatic scale-out options across shared and reserved instances for greater isolation and performance.


[LEARN MORE](#) [MANAGE](#)

[try it now](#)

4. Select your subscription and click the check.



ADD PREVIEW FEATURE

 **WEB SITES**
FREE

SUBSCRIPTIONS

3-Month Free Trial

You will receive an e-mail when this feature is enabled for your subscription.

By clicking the Submit button, I agree to the Windows Azure [Preview Features Terms of Use](#) and [Privacy Statement](#), including the terms for Preview Releases.

☒

Create a Windows Azure Web Site and enable Git publishing

Follow these steps to create a Windows Azure Web Site, and then enable Git publishing for the web site.

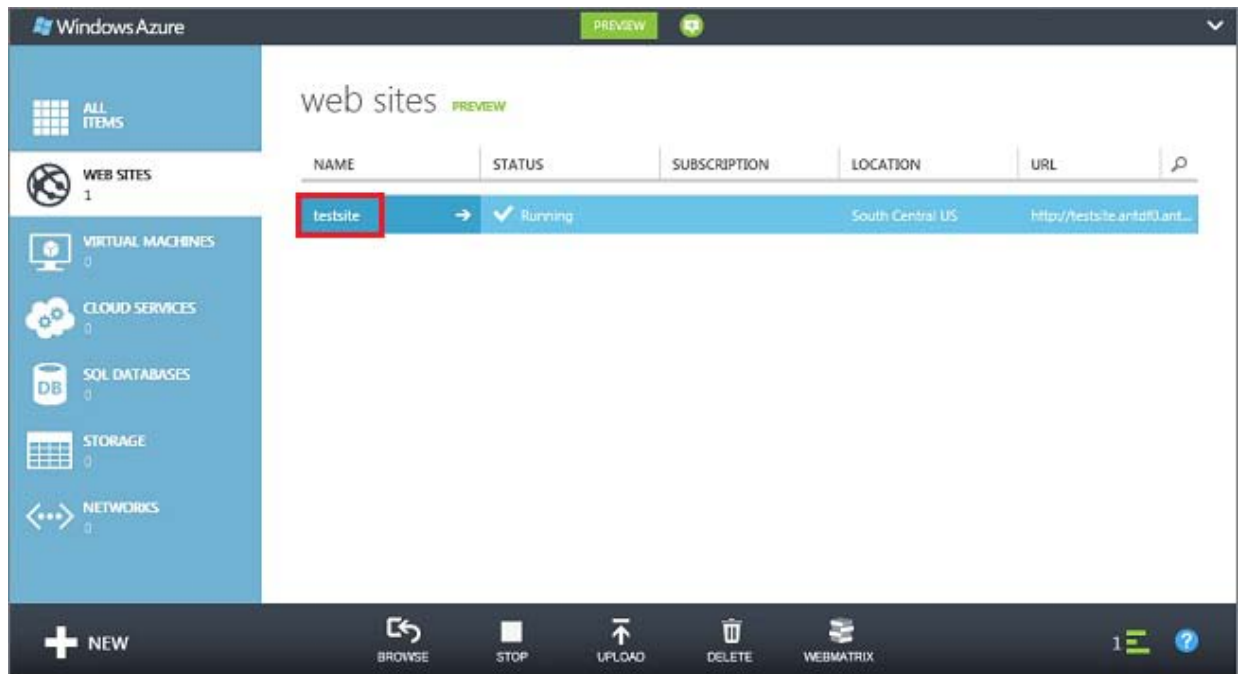
1. Login to the [Windows Azure Portal](#).
2. Click the + **NEW** icon on the bottom left of the portal



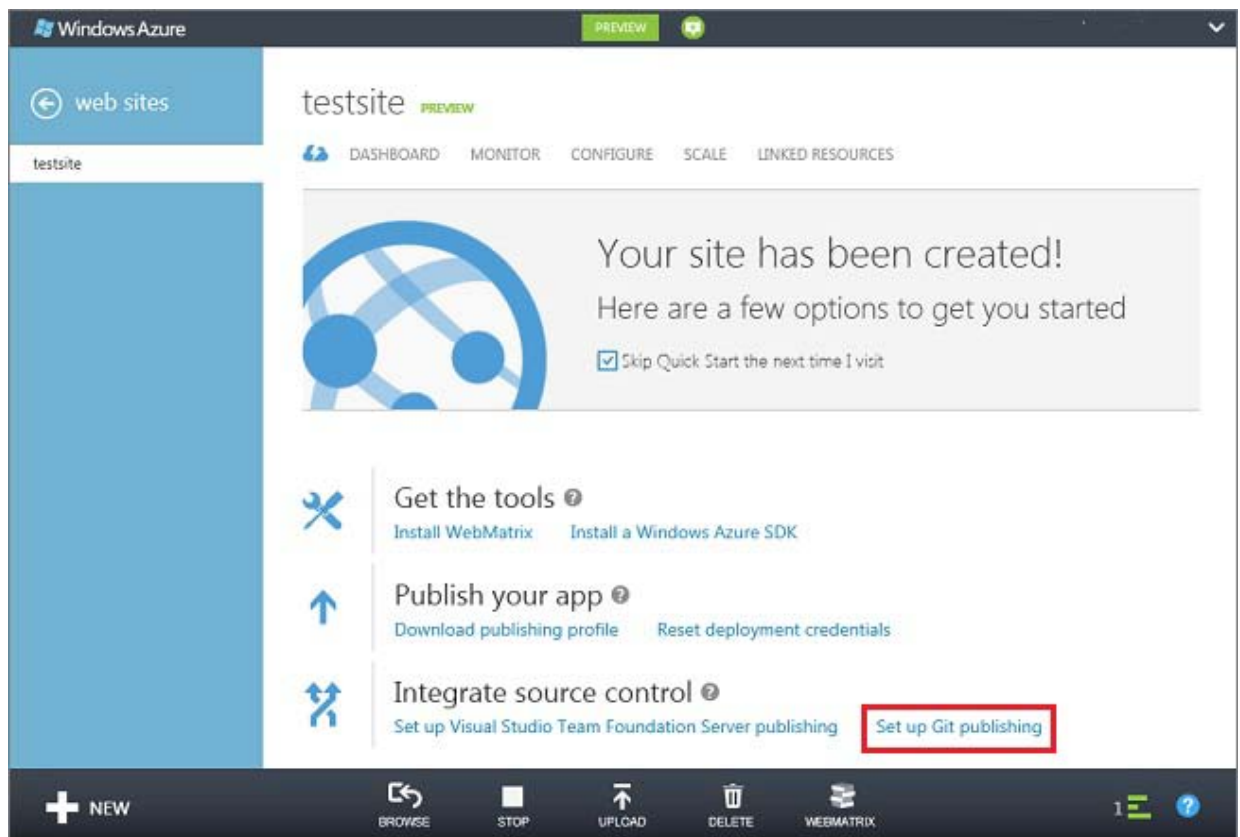
3. Click **WEB SITE**, then **QUICK CREATE**. Enter a value for **URL** and select the datacenter for your web site in the **REGION** dropdown. Click the checkmark at the bottom of the dialog.

The screenshot shows the 'NEW' dialog in the Azure portal. On the left, there is a list of service categories: WEB SITE, VIRTUAL MACHINE, CLOUD SERVICE, SQL DATABASE, STORAGE, and NETWORK. The 'WEB SITE' category is selected and highlighted. To the right of this list, under the 'QUICK CREATE' section, there are three options: 'CREATE WITH DATABASE', 'FROM GALLERY', and 'FROM GALLERY'. The 'QUICK CREATE' section is also highlighted. On the right side of the dialog, there is a form with two fields: 'URL' and 'REGION'. The 'URL' field contains the text '.antdf0.antares-test.windows-int.net'. The 'REGION' dropdown menu is set to 'South Central US'. At the bottom right of the dialog, there is a button labeled 'CREATE WEB SITE' with a checkmark icon.

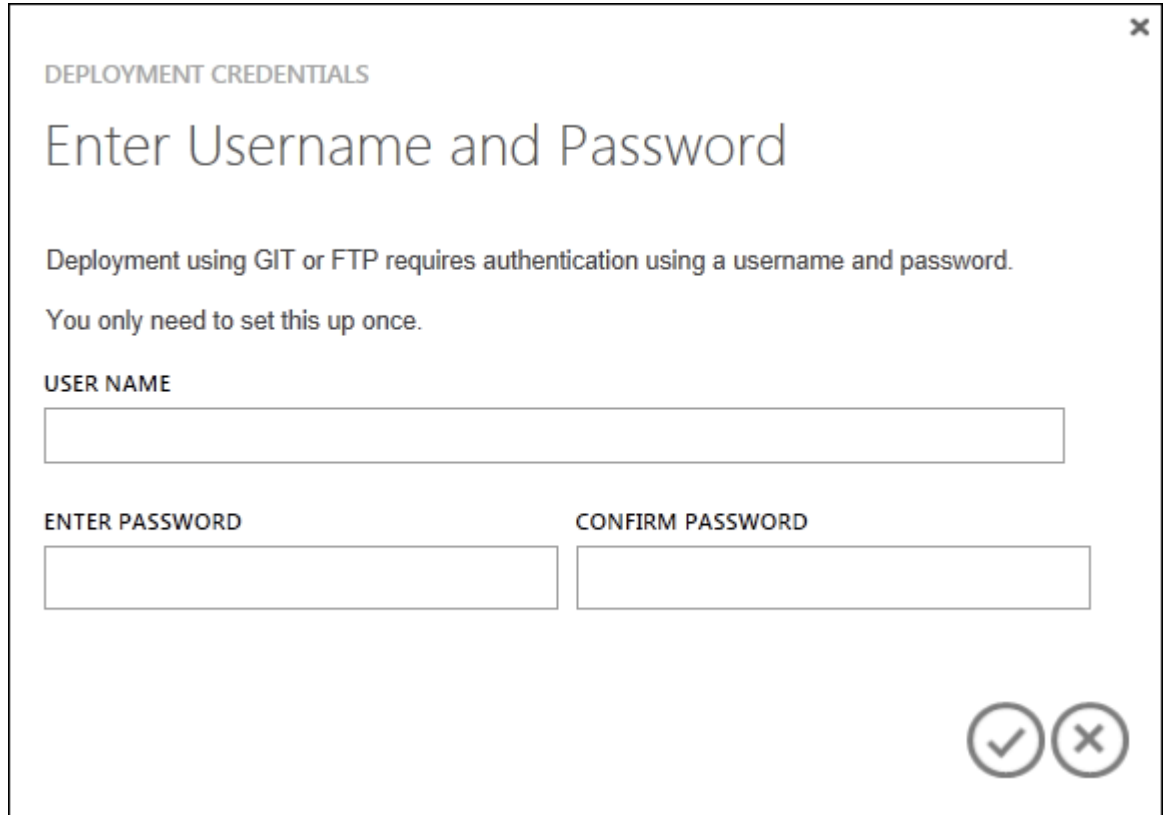
4. Once the web site status changes to **Running**, click on the name of the web site to access the **Dashboard**



- At the bottom right of the Dashboard, select **Set up Git Publishing**.



6. To enable Git publishing, you must provide a user name and password. If you have previously enabled publishing for a Windows Azure Web Site, you will not be prompted for the user name or password. Instead, a Git repository will be created using the user name and password you previously specified. Make a note of the user name and password, as they will be used for Git publishing to all Windows Azure Web Sites you create.



A screenshot of a 'DEPLOYMENT CREDENTIALS' dialog box. The title bar is light gray with a close button (X) in the top right corner. The main content area is white. At the top, the title 'DEPLOYMENT CREDENTIALS' is in a small, dark gray font. Below it, the heading 'Enter Username and Password' is in a large, bold, dark gray font. A message in a smaller font states: 'Deployment using GIT or FTP requires authentication using a username and password. You only need to set this up once.' Below this message are three input fields. The first is a single-line text box labeled 'USER NAME'. The second and third are single-line text boxes labeled 'ENTER PASSWORD' and 'CONFIRM PASSWORD' respectively, positioned side-by-side. At the bottom right of the dialog, there are two circular buttons: one with a checkmark and one with an 'X'.

DEPLOYMENT CREDENTIALS

Enter Username and Password

Deployment using GIT or FTP requires authentication using a username and password.
You only need to set this up once.

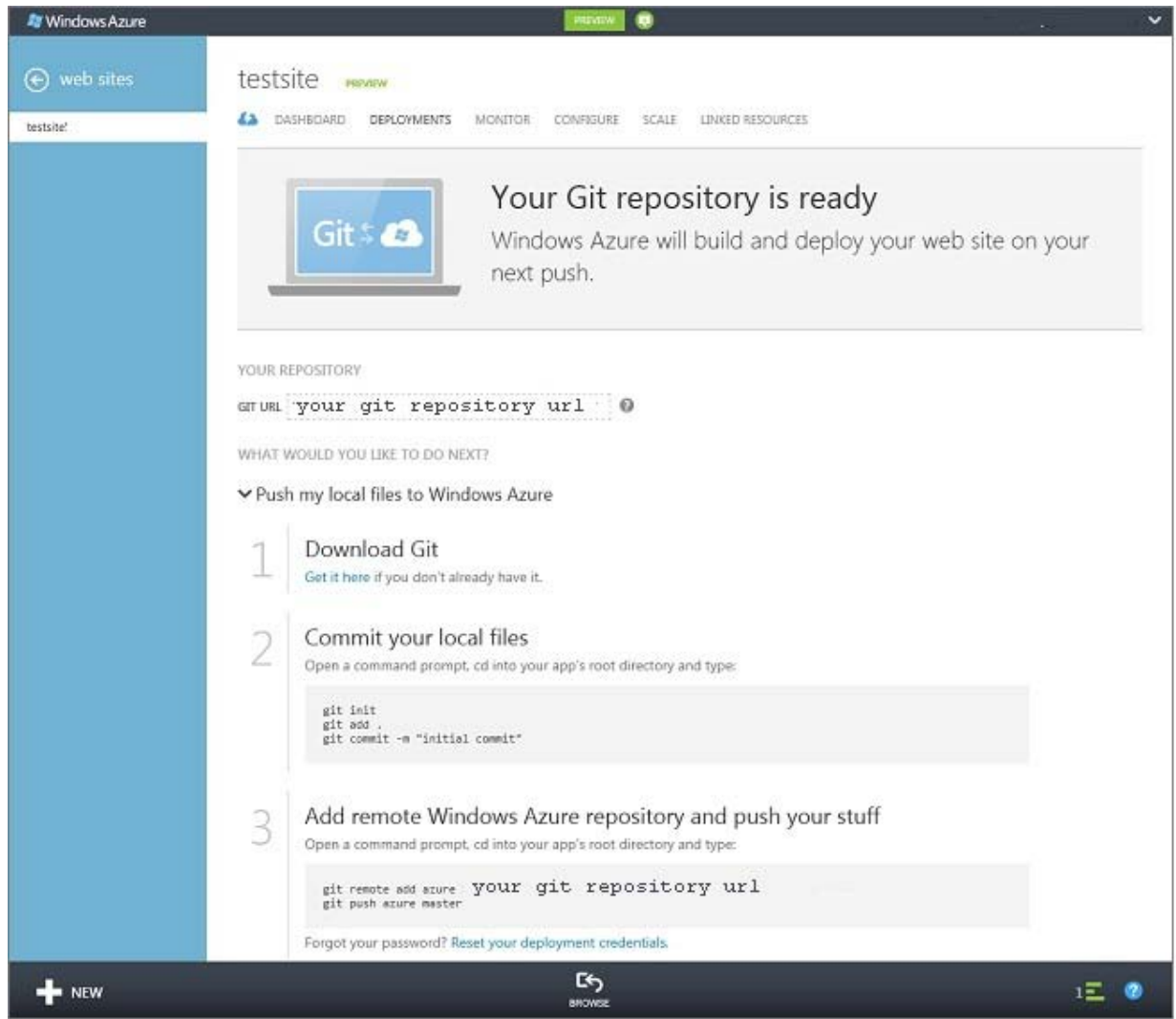
USER NAME

ENTER PASSWORD

CONFIRM PASSWORD

✓ ✕

7. Once the Git repository is ready, you will be presented with instructions on the Git commands to use in order to setup a local repository and then push the files to Windows Azure.



Note Save the instructions returned by the **Push my local files to Windows Azure** link, as they will be used in the following sections.

Install developer tools

To successfully complete the steps in this tutorial, you must have a working installation of Node.js and Git. Installation package for Node.js is available from the [nodejs.org download page](https://nodejs.org/download) while installation package for Git is available from the [git-scm.com download page](https://git-scm.com/download).

Note If you are performing this tutorial on Windows, you can set up your machine with [Windows Azure SDK for Node.js](#) that includes Node.js.

Build and test your application locally

In this section, you will create a **server.js** file containing the 'hello world' example from [nodejs.org](#). This example has been modified from the original example by adding `process.env.port` as the port to listen on when running in a Windows Azure Web Site.

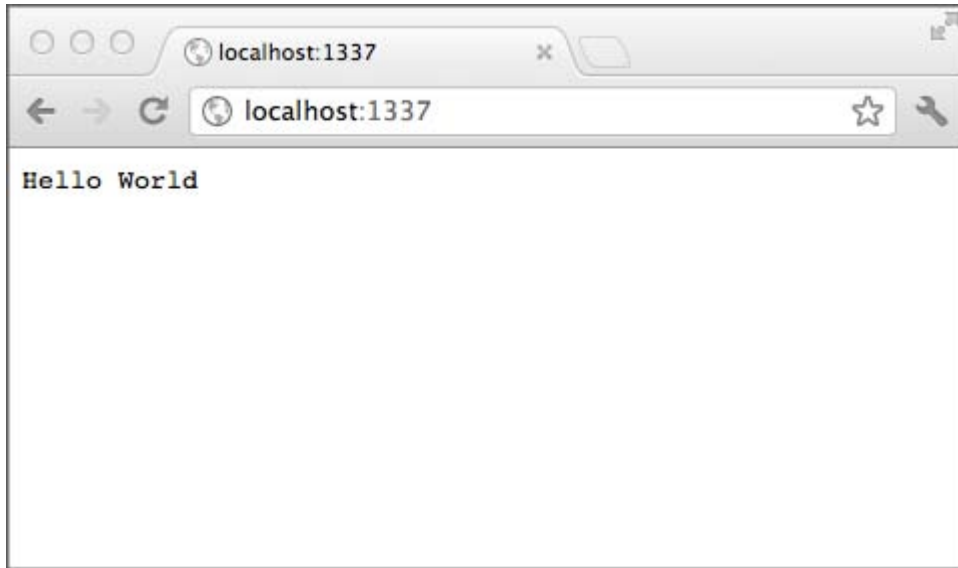
Note This tutorial makes reference to the **helloworld** folder. The full path to this folder is omitted, as path semantics differ between operating systems. You should create this folder in a location that is easy for you to access on your local file system, such as **~/node/helloworld** or **c:\node\helloworld**

Note Many of the steps below mention using the command-line. For these steps, use the command-line for your operating system, such as **Windows PowerShell**, **cmd.exe**, **GitBash** (Windows,) or **Bash** (Unix Shell). On OS X systems you can access the command-line through the Terminal application.

1. Using a text editor, create a new file named **server.js** in the **helloworld** directory. If the **helloworld** directory does not exist, create it.
2. Add the following as the contents of the **server.js** file, and then save it:

```
var http = require('http')
var port = process.env.port || 1337;
http.createServer(function(req, res) {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World\n');
}).listen(port);
```
3. Open the command-line, and use the following command to start the web page locally:

```
node server.js
```
4. Open your web browser and navigate to `http://localhost:1337`. A web page displaying "Hello World" will appear as shown in the screenshot below:



5. To stop the application, switch to the Terminal window and hold down the **CTRL** and **C** keys on your keyboard.

Publish your application

1. From the command-line, change directories to the **helloworld** directory and enter the following commands to initialize a local Git repository.

```
git init
```

2. Use the following commands to add files to the repository:

```
git add .  
git commit -m "initial commit"
```

3. Add a Git remote for pushing updates to the Windows Azure Web Site you created previously, using the following command:

```
git remote add azure [URL for remote repository]
```

Note the URL used should be the one returned at the end of the **Create a Windows Azure Web Site and Set up Git Publishing** section. If you forgot to save the URL earlier you can retrieve it now by clicking the "Deployment" tab of your Windows Azure Web Site within the management portal



4. Push your changes to Windows Azure using the following command:

```
git push azure master
```

You will be prompted for the password you created earlier and will see the following output:

```

hello123 — bash — 80x24
$ git remote add azure https://kudu1@hello123.scm.azurewebsites.net/hello123.git
$ git push azure master
Password:
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 374 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: New deployment received.
remote: Updating branch 'master'.
remote: Preparing deployment for commit id '5ebbe250c9'.
remote: Preparing files for deployment.
remote: Deploying Web.config to enable Node.js activation.
remote: Deployment successful.
To https://kudu1@hello123.scm.azurewebsites.net/hello123.git
 * [new branch]      master -> master
$

```

If you navigate to the deployments tab of your Windows Azure Web Site within the management portal, you will see your first deployment in the deployment history:



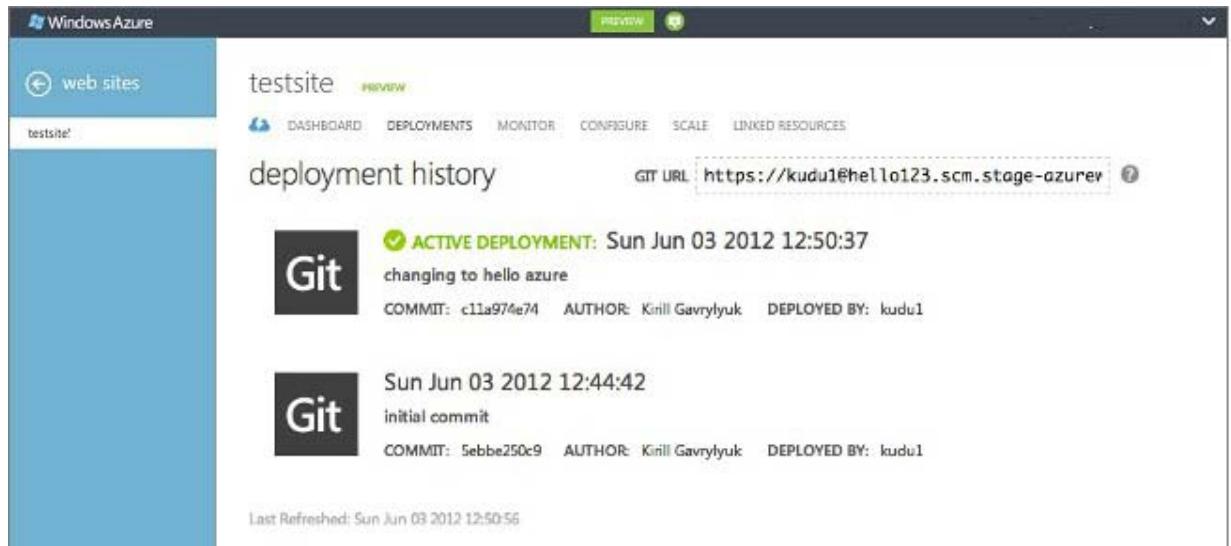
5. Browse to **http://[your web site url]/** to begin using the application. You can find your web site url on the "Dashboard" tab of your Windows Azure Web Site within the management portal.

Publish changes to your application

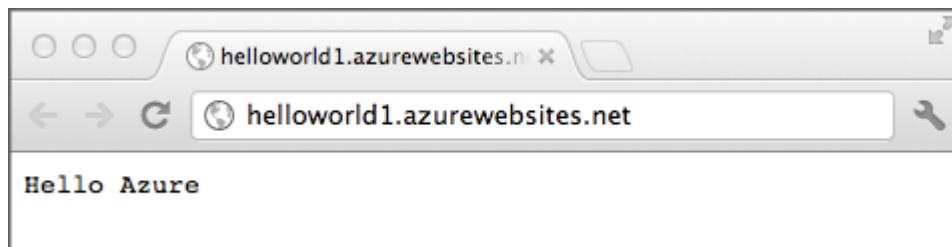
1. Open the **server.js** file in a text editor, and change 'Hello World\n' to 'Hello Azure\n'. Save the file.
2. From the command-line, change directories to the **helloworld** directory and run the following commands:

```
git add .  
git commit -m "changing to hello azure"  
git push azure master
```

You will be prompted for the password you created earlier. If you navigate to the deployments tab of your Windows Azure Web Site within the management portal, you will see your updated deployment history:



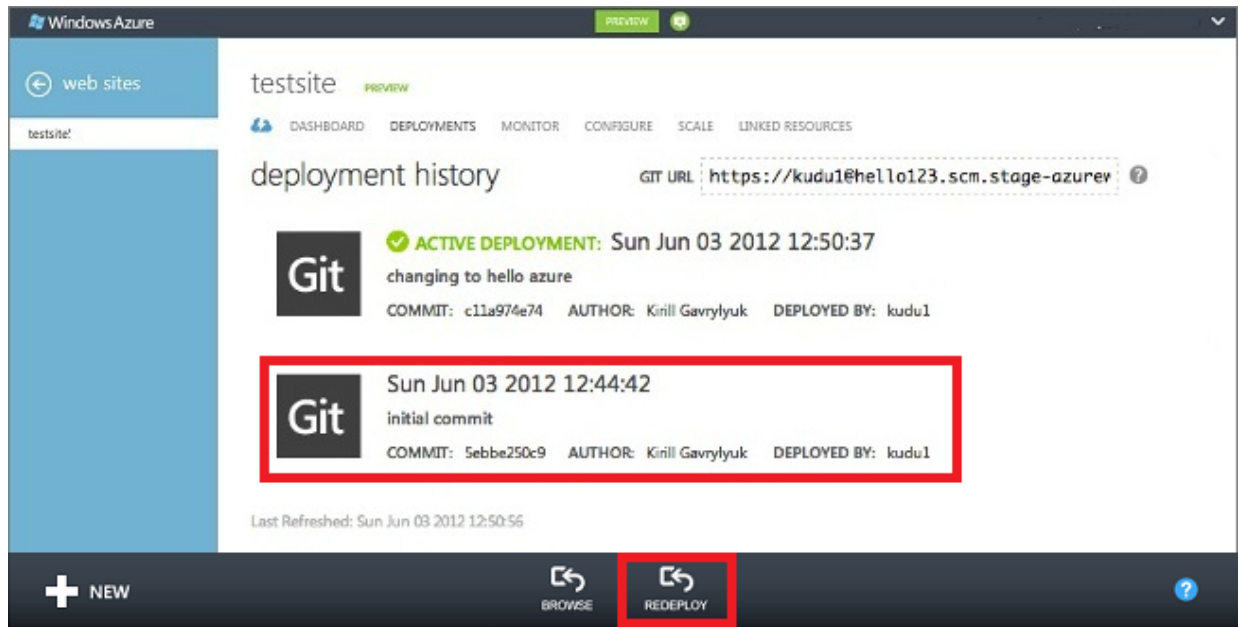
3. Browse to **[http://\[your web site url\]/](http://[your web site url]/)** and note that the updates have been applied.



Revert to a previous deployment

Since Windows Azure maintains a git repository for your web site, you can use the **Deployments** page to revert to a previous deployment.

1. In the [Windows Azure Portal](#), select your web site and then select **Deployments**.
2. Select a previous deployment, and then click **Redeploy** at the bottom of the page. When prompted, select **Yes**.



3. Once the deployment status changes to **Active Deployment**, view the web site in your browser and note that it has reverted to the selected deployment.

Next steps

While the steps in this article use the Windows Azure Portal to create a web site, you can also use the [Windows Azure Command-Line Tools for Mac and Linux](#) to perform the same operations.

Additional Resources

[Windows Azure PowerShell](#)

[Windows Azure Command-Line Tools for Mac and Linux](#)

Node.js Cloud Service

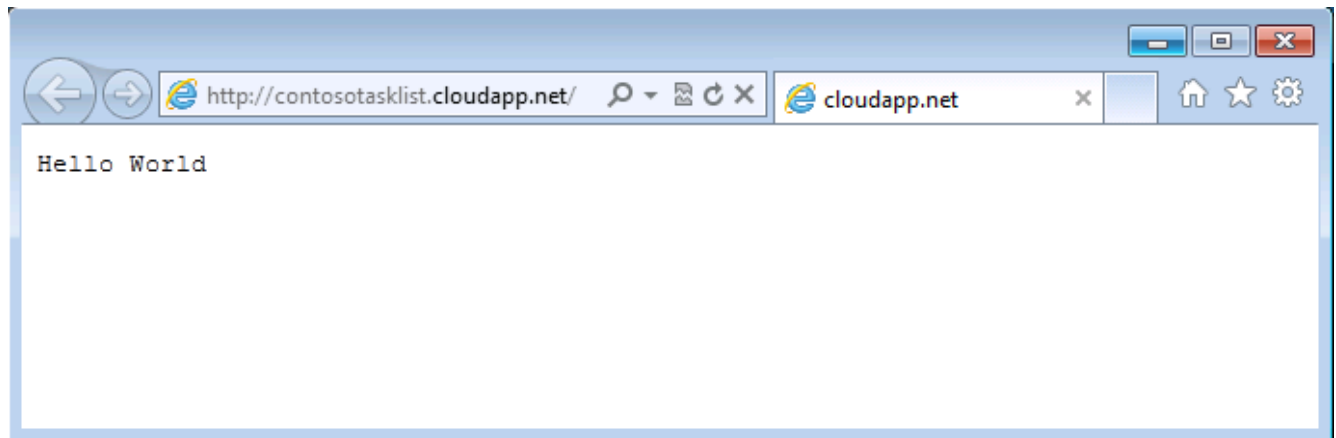
Developing for Windows Azure is easy when using the available tools. This tutorial assumes you have no prior experience using Windows Azure. On completing this guide, you will have an application that uses multiple Windows Azure resources up and running in the cloud.

You will learn:

- How to create a new Windows Azure Node.js application using the Windows PowerShell tools.
- How to run your Node application locally using the Windows Azure compute emulator
- How to publish and re-publish your application to a Cloud Service in Windows Azure.

By following this tutorial, you will build a simple Hello World web application. The application will be hosted in an instance of a web role that, when running in Windows Azure, is itself hosted in a dedicated virtual machine (VM).

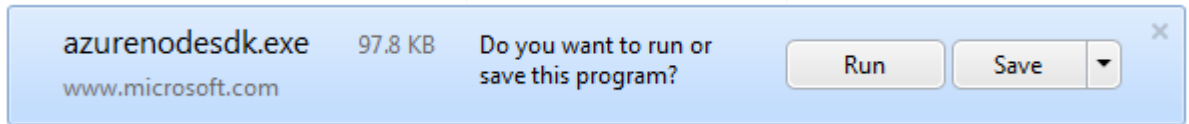
A screenshot of the completed application is below:



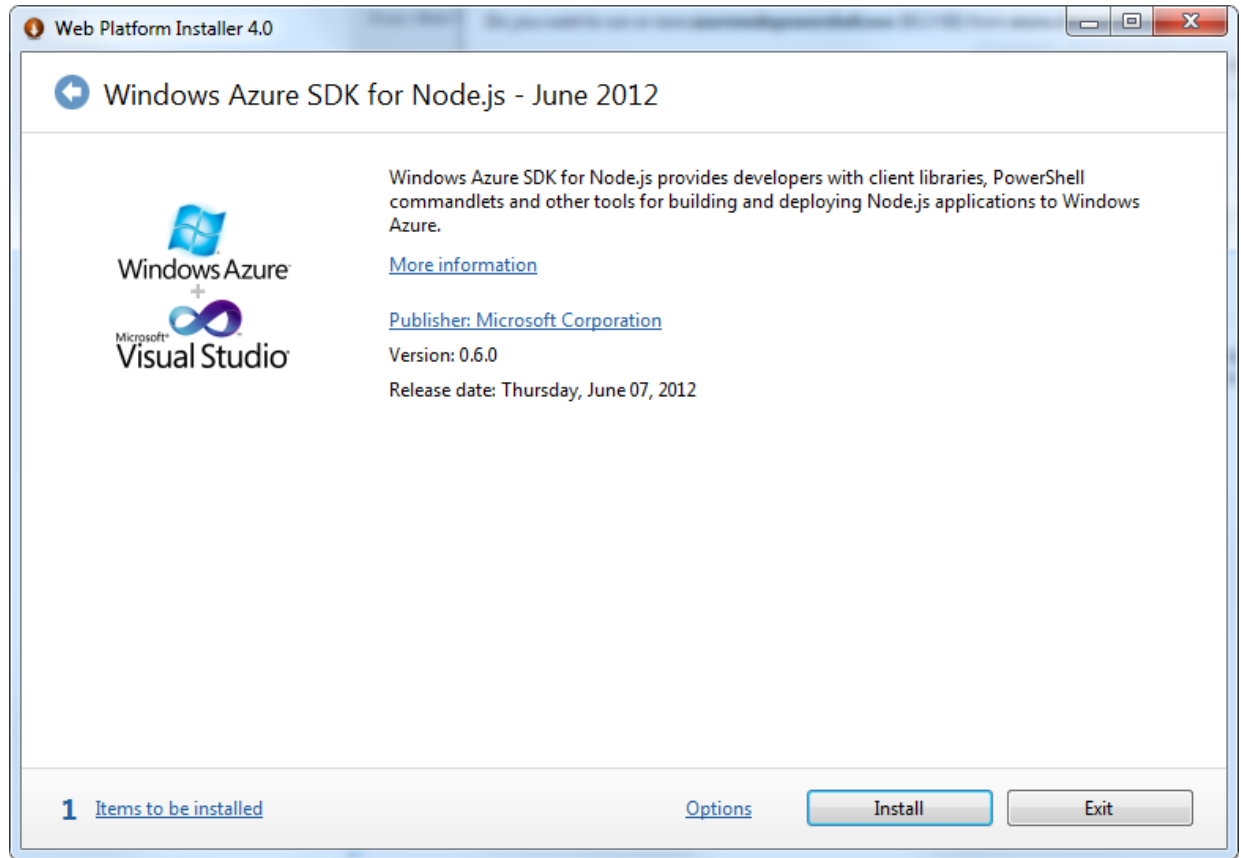
Setting Up the Development Environment

Before you can begin developing your Windows Azure application, you need to get the tools and set up your development environment.

1. To install the Windows Azure SDK for Node.js, click the button below:
[Get Tools and SDK](#)
2. Select **Install Now**, and when prompted to run or save azurenodesdk.exe, click Run:



3. Click **Install** in the installer window and proceed with the installation:



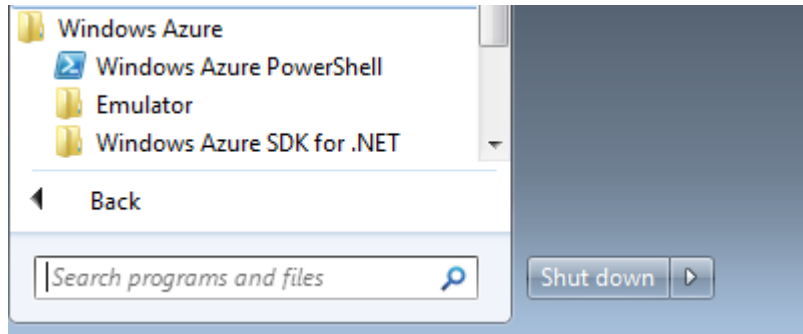
Once the installation is complete, you have everything necessary to start developing. The following components are installed:

- Node.js
- IISNode
- NPM for Windows
- Windows Azure Compute & Storage Emulators
- Windows Azure PowerShell

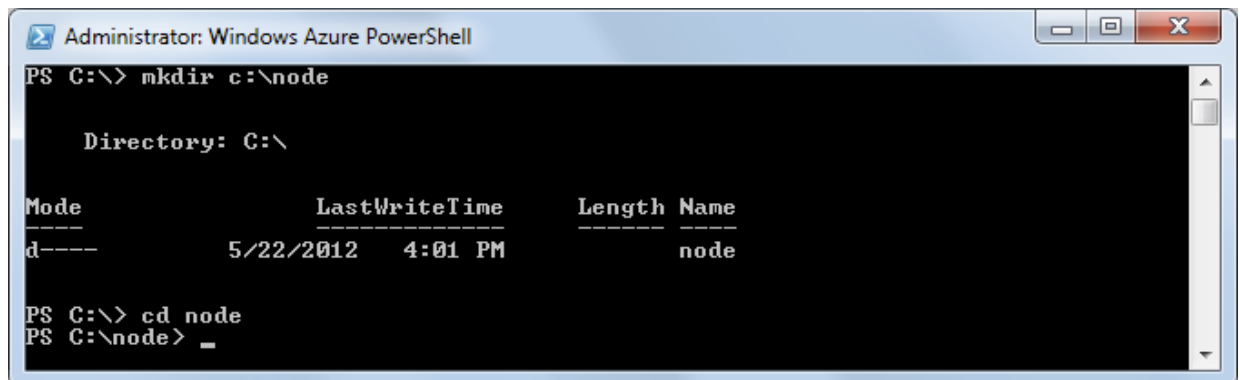
Creating a New Node Application

The Windows Azure SDK for Node.js includes a Windows PowerShell environment that is configured for Windows Azure and Node development. It includes tools that you can use to create and publish Node applications.

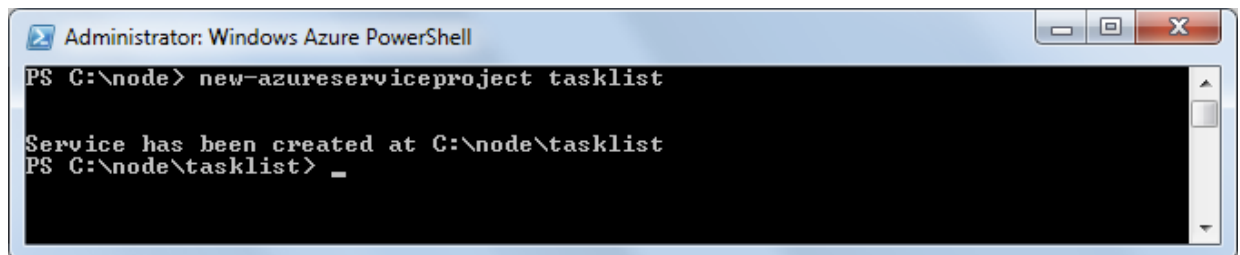
1. On the **Start** menu, click **All Programs, Windows Azure**, right-click **Windows Azure PowerShell**, and then select **Run As Administrator**. Opening your Windows PowerShell environment this way ensures that all of the Node command-line tools are available. Running with elevated privileges avoids extra prompts when working with the Windows Azure Emulator.



2. Create a new **node** directory on your C drive, and change to the c:\node directory:



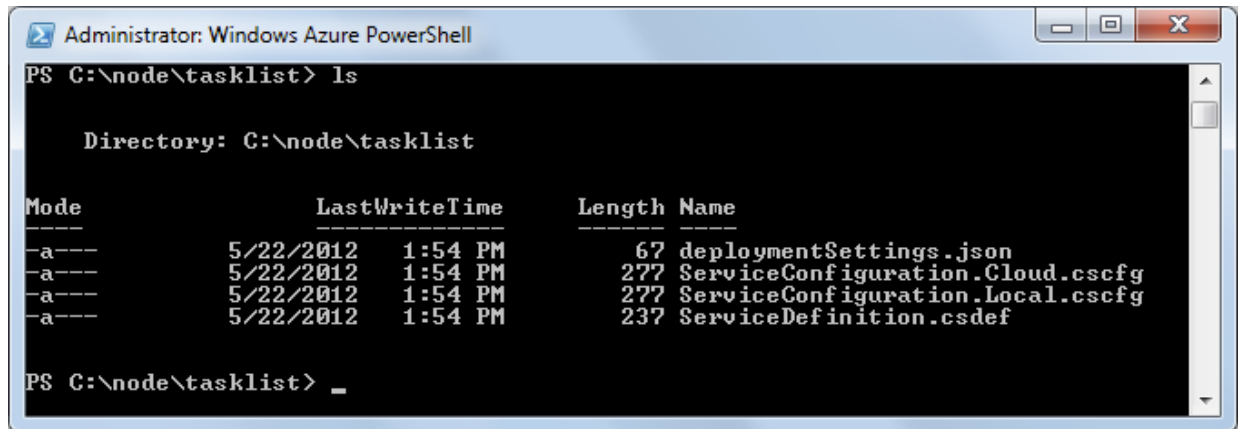
3. Enter the following cmdlet to create a new solution:
PS C:\node> New-AzureServiceProject tasklist
You will see the following response:



The **New-AzureServiceProject** cmdlet generates a basic structure for creating a new Windows Azure Node application which will be published to a Cloud Service. It contains configuration files necessary for publishing to Windows Azure. The cmdlet also changes your working directory to the directory for the service.

Enter the following command to see a listing of the files that were generated:

```
PS C:\node\tasklist> ls
```



```
Administrator: Windows Azure PowerShell
PS C:\node\tasklist> ls

Directory: C:\node\tasklist

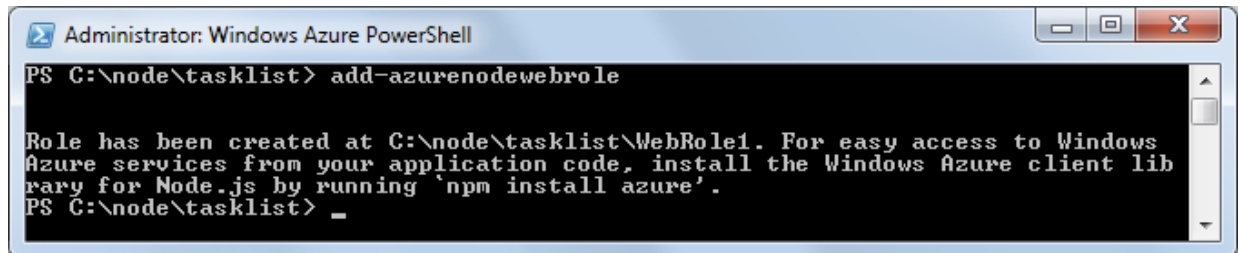
Mode                LastWriteTime         Length Name
----                -
-a---             5/22/2012   1:54 PM           67 deploymentSettings.json
-a---             5/22/2012   1:54 PM          277 ServiceConfiguration.Cloud.cscfg
-a---             5/22/2012   1:54 PM          277 ServiceConfiguration.Local.cscfg
-a---             5/22/2012   1:54 PM          237 ServiceDefinition.csdef

PS C:\node\tasklist> _
```

- ServiceConfiguration.Cloud.cscfg, ServiceConfiguration.Local.cscfg and ServiceDefinition.csdef are Windows Azure-specific files necessary for publishing your application. For more information about these files, see [Overview of Creating a Hosted Service for Windows Azure](#).
 - deploymentSettings.json stores local settings that are used by the Windows Azure PowerShell deployment cmdlets.
4. Enter the following command to add a new web role using the **Add-AzureNodeWebRole** cmdlet:

```
PS C:\node\tasklist> Add-AzureNodeWebRole
```

You will see the following response:



```
Administrator: Windows Azure PowerShell
PS C:\node\tasklist> add-azurenodewebrole

Role has been created at C:\node\tasklist\WebRole1. For easy access to Windows Azure services from your application code, install the Windows Azure client library for Node.js by running 'npm install azure'.
PS C:\node\tasklist> _
```

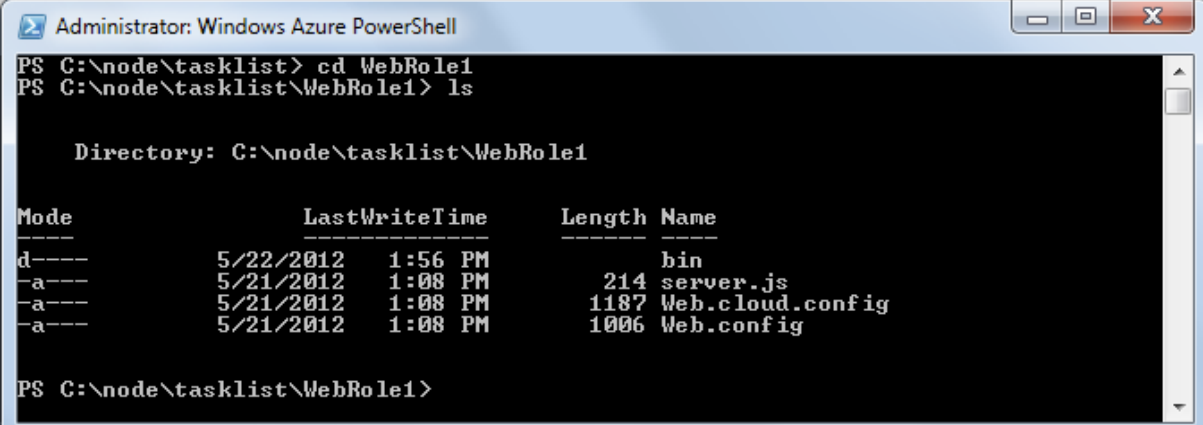
The **Add-AzureNodeWebRole** cmdlet creates a new directory for your application and generates additional files that will be needed when your application is published. In Windows Azure, *roles* define components that can run in the Windows Azure execution environment. A *web role* is customized for web application programming.

By default if you do not provide a role name, one will be created for you i.e. WebRole1. You can provide a name as the first parameter to **Add-AzureNodeWebRole** to override i.e. **Add-AzureNodeWebRole MyRole**

Enter the following commands to change to the newly generated directory and view its contents:

```
PS C:\node\tasklist> cd WebRole1
```

```
PS C:\node\tasklist\WebRole1> ls
```



The screenshot shows a Windows PowerShell window titled "Administrator: Windows Azure PowerShell". The command prompt shows the user has navigated to the directory C:\node\tasklist\WebRole1 and executed the 'ls' command. The output displays a table of files in the directory.

Mode	LastWriteTime	Length	Name
d----	5/22/2012 1:56 PM		bin
-a---	5/21/2012 1:08 PM	214	server.js
-a---	5/21/2012 1:08 PM	1187	Web.cloud.config
-a---	5/21/2012 1:08 PM	1006	Web.config

The command prompt then shows the user has entered the 'notepad server.js' command.

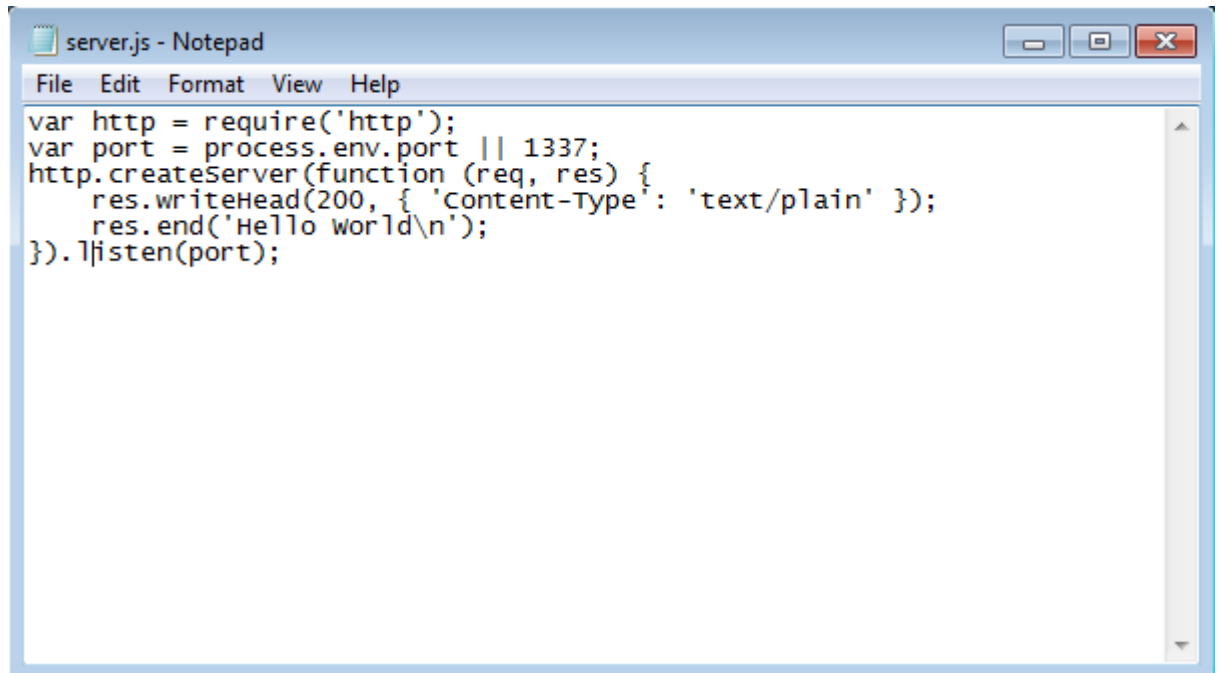
- server.js contains the starter code for your application.

5. Open the server.js file in Notepad. Alternatively, you can open the server.js file in your favorite text editor.

```
PS C:\node\tasklist\WebRole1> notepad server.js
```

This file contains the following starter code that the tools have generated. This code is almost identical to the "Hello World" sample on the nodejs.org website, except:

- The port has been changed to allow IIS to handle HTTP traffic on behalf of the application. IIS Node.js integration provides Node.js applications with a number of benefits when running on-premise or in Windows Azure, including: process management, scalability on multi-core servers, auto-update, side-by-side with other languages, etc.
- Console logging has been removed.



```
server.js - Notepad
File Edit Format View Help
var http = require('http');
var port = process.env.port || 1337;
http.createServer(function (req, res) {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello world\n');
}).listen(port);
```

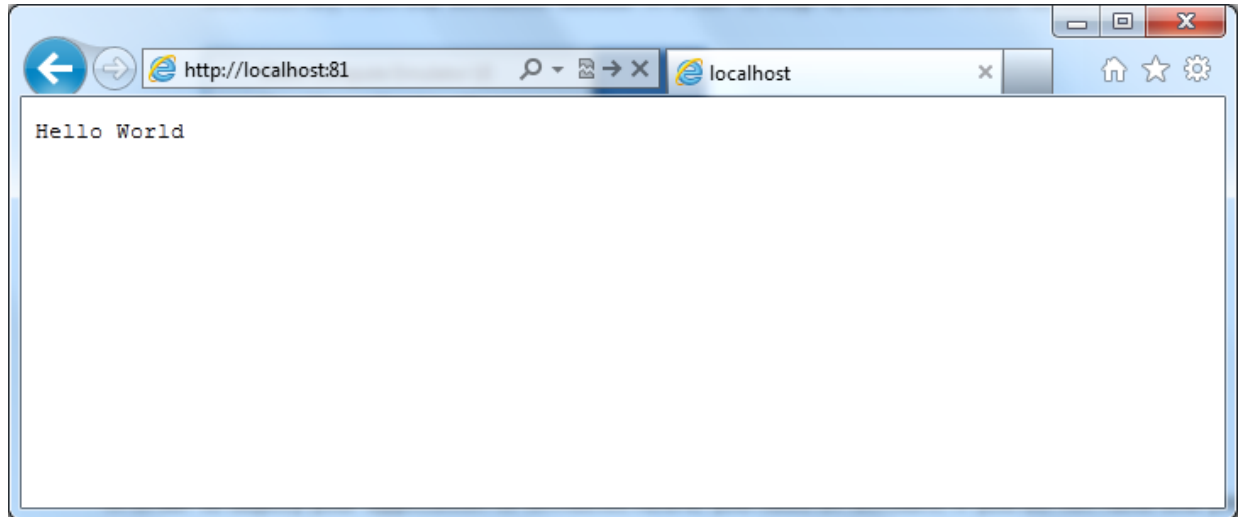
Running Your Application Locally in the Emulator

One of the tools installed by the Windows Azure SDK is the Windows Azure compute emulator, which allows you to test your application locally. The compute emulator simulates the environment your application will run in when it is deployed to the cloud, including providing access to services like Windows Azure Table Storage. This means you can test your application without having to actually deploy it.

1. Close Notepad and switch back to the Windows PowerShell window. Enter the following cmdlet to run your service in the emulator and launch a browser window:

```
PS C:\node\tasklist\WebRole1> Start-AzureEmulator -Launch
```

The **-launch** parameter specifies that the tools should automatically open a browser window and display the application once it is running in the emulator. A browser opens and displays "Hello World," as shown in the screenshot below. This indicates that the service is running in the compute emulator and is working correctly.



2. To stop the compute emulator, use the **Stop-AzureEmulator** command:

```
PS C:\node\tasklist\WebRole1> Stop-AzureEmulator
```

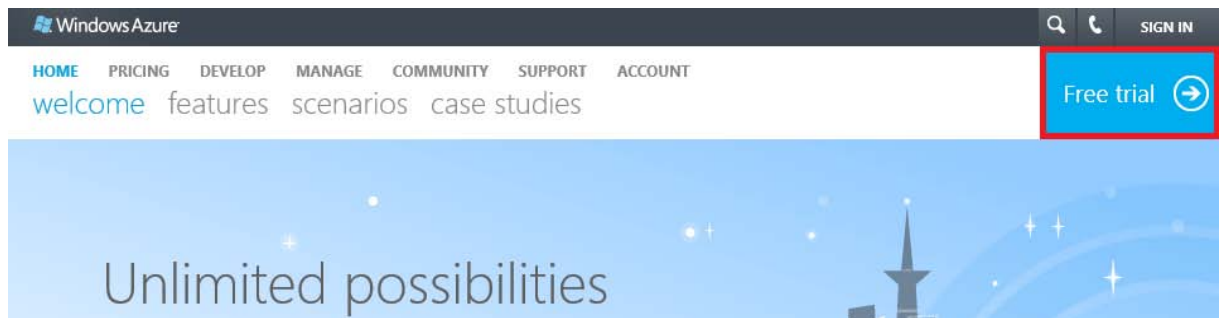
Deploying the Application to Windows Azure

In order to deploy your application to Windows Azure, you need an account. If you do not have one you can create a free trial account. Once you are logged in with your account, you can download a Windows Azure publishing profile. The publishing profile authorizes your computer to publish deployment packages to Windows Azure using the Windows PowerShell cmdlets.

Creating a Windows Azure Account

1. Open a web browser, and browse to <http://www.windowsazure.com>.

To get started with a free account, click on **Free Trial** in the upper right corner and follow the steps.



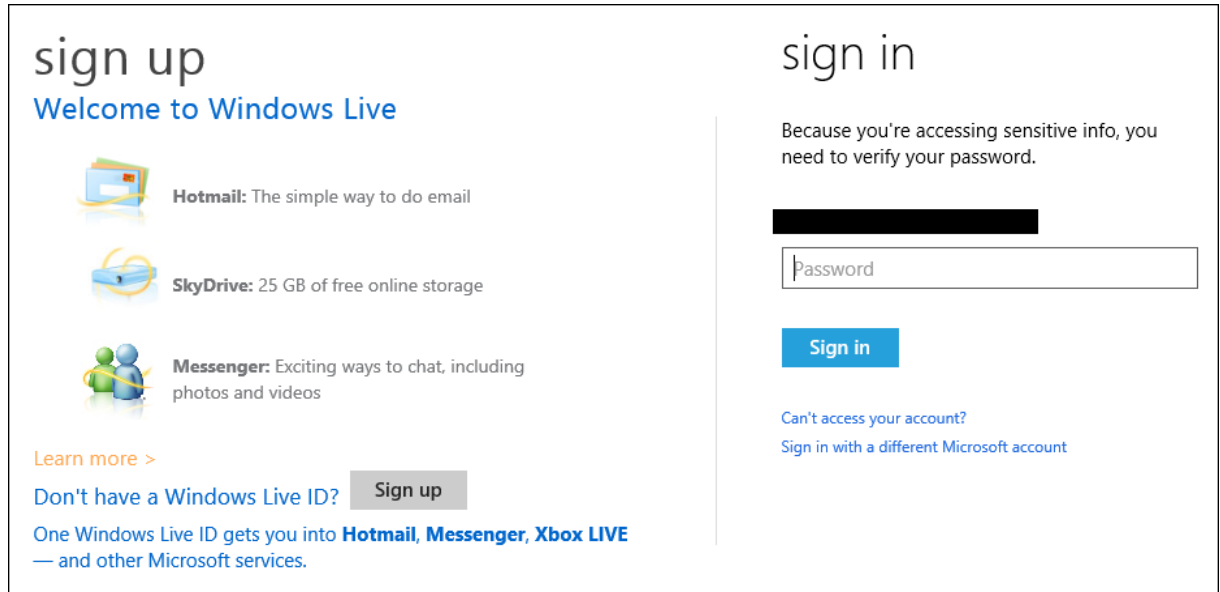
2. Your account is now created. You are ready to deploy your application to Windows Azure!

Downloading the Windows Azure Publishing Settings

1. From the Windows PowerShell window, launch the download page by running the following cmdlet:

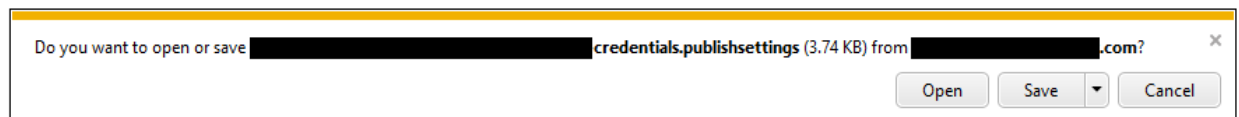
```
PS C:\node\tasklist\WebRole1> Get-AzurePublishSettingsFile
```

This launches the browser for you to log into the Windows Azure Management Portal with your Windows Live ID credentials.



The screenshot shows the Windows Live sign up and sign in page. On the left, under 'sign up', it says 'Welcome to Windows Live' and lists services: Hotmail (The simple way to do email), SkyDrive (25 GB of free online storage), and Messenger (Exciting ways to chat, including photos and videos). There is a 'Learn more >' link and a 'Sign up' button. Below that, it says 'Don't have a Windows Live ID?' and 'One Windows Live ID gets you into Hotmail, Messenger, Xbox LIVE — and other Microsoft services.' On the right, under 'sign in', it says 'Because you're accessing sensitive info, you need to verify your password.' There is a password input field and a 'Sign in' button. Below that, there are links for 'Can't access your account?' and 'Sign in with a different Microsoft account'.

2. Log into the Management Portal. This takes you to the page to download your Windows Azure publishing settings.
3. Save the profile to a file location you can easily access:



4. In the Windows Azure PowerShell window, use the following cmdlet to configure the Windows PowerShell for Node.js cmdlets to use the Windows Azure publishing profile you downloaded:

```
PS C:\node\tasklist\WebRole1> Import-AzurePublishSettingsFile [path to file]
```

After importing the publish settings, consider deleting the downloaded .publishSettings as the file contains information that can be used by others to access your account.

Publishing the Application

1. Publish the application using the **Publish-AzureServiceProject** cmdlet, as shown below.
 - **ServiceName** specifies the name for the service. The name must be unique across all other services in Windows Azure. For example, below, "TaskList" is suffixed with "Contoso," the company name, to make the service name unique. By default if ServiceName is not provided, the project folder name will be used.
 - **Location** specifies the country/region for which the application should be optimized. You can expect faster loading times for users accessing it from this region. Examples of the\ available regions include: North Central US, Anywhere US, Anywhere Asia, Anywhere Europe, North Europe, South Central US, and Southeast Asia.
 - **Launch** specifies to open the browser at the location of the hosted service after publishing has completed.

```
PS C:\node\tasklist\WebRole1> Publish-AzureServiceProject -  
ServiceName TaskListContoso -Location "North Central US" -  
Launch
```

Be sure to use a **unique name**, otherwise the publish process will fail. After publishing succeeds, you will see the following response:



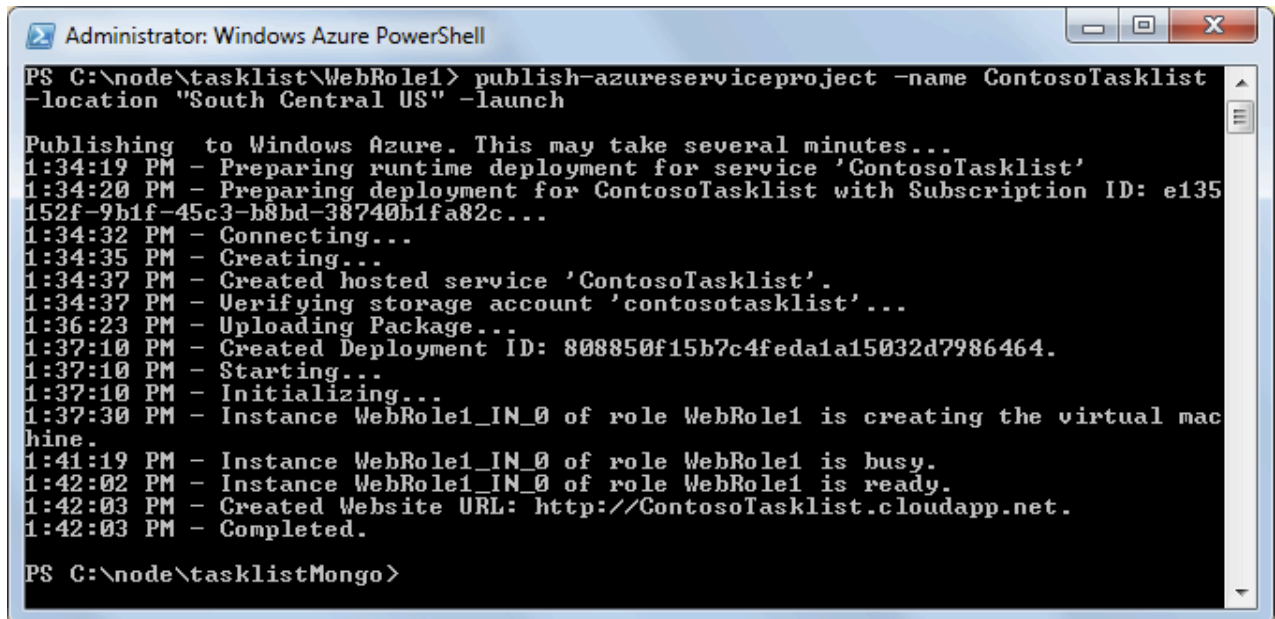
```
Administrator: Windows Azure PowerShell  
PS C:\node\tasklist\WebRole1> publish-azureserviceproject -name ContosoTasklist  
-location "South Central US" -launch  
  
Publishing to Windows Azure. This may take several minutes...  
1:32:18 PM - Preparing runtime deployment for service 'ContosoTasklist'
```

The **Publish-AzureServiceProject** cmdlet performs the following steps:

1. Creates a package that will be deployed to Windows Azure. The package contains all the files in your node.js application folder.
2. Creates a new storage account if one does not exist. The Windows Azure storage account is used in the next section of the tutorial for storing and accessing data.
3. Creates a new hosted service if one does not already exist. A *hosted service* is the container in which your application is hosted when it is deployed to Windows Azure. For more information, see [Overview of Creating a Hosted Service for Windows Azure](#).
4. Publishes the deployment package to Windows Azure.

It can take 5–7 minutes for the application to deploy. Since this is the first time you are publishing, Windows Azure provisions a virtual machine (VM), performs security hardening, creates a web role on the VM to host your application, deploys your code to that web role, and finally configures the load balancer and networking so your application is available to the public.

After the deployment is complete, the following response appears.

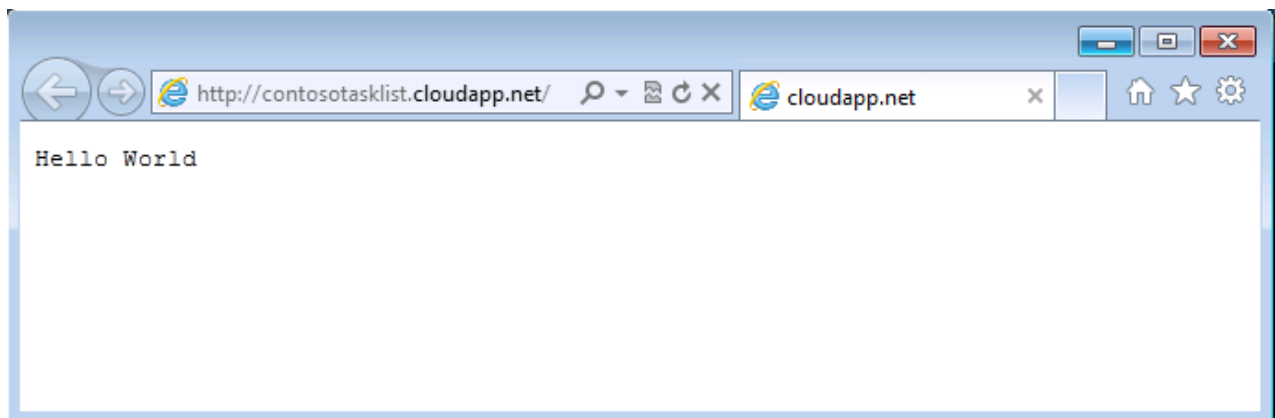


```
Administrator: Windows Azure PowerShell
PS C:\node\tasklist\WebRole1> publish-azureproject -name ContosoTasklist
-location "South Central US" -launch

Publishing to Windows Azure. This may take several minutes...
1:34:19 PM - Preparing runtime deployment for service 'ContosoTasklist'
1:34:20 PM - Preparing deployment for ContosoTasklist with Subscription ID: e135152f-9b1f-45c3-b8bd-38740b1fa82c...
1:34:32 PM - Connecting...
1:34:35 PM - Creating...
1:34:37 PM - Created hosted service 'ContosoTasklist'.
1:34:37 PM - Verifying storage account 'contosotasklist'...
1:36:23 PM - Uploading Package...
1:37:10 PM - Created Deployment ID: 808850f15b7c4feda1a15032d7986464.
1:37:10 PM - Starting...
1:37:10 PM - Initializing...
1:37:30 PM - Instance WebRole1_IN_0 of role WebRole1 is creating the virtual machine.
1:41:19 PM - Instance WebRole1_IN_0 of role WebRole1 is busy.
1:42:02 PM - Instance WebRole1_IN_0 of role WebRole1 is ready.
1:42:03 PM - Created Website URL: http://ContosoTasklist.cloudapp.net.
1:42:03 PM - Completed.

PS C:\node\tasklistMongo>
```

The browser also opens to the URL for your service and display a web page that calls your service.



Your application is now running on Windows Azure! The hosted service contains the web role you created earlier. You can easily scale your application by changing the number of instances allocated to each role in the ServiceConfiguration.Cloud.cscfg file. You may want to use only one instance when deploying for development and test purposes, but multiple instances when deploying a production application.

Stopping and Deleting Your Application

After deploying your application, you may want to disable it so you can avoid costs or build and deploy other applications within the free trial time period.

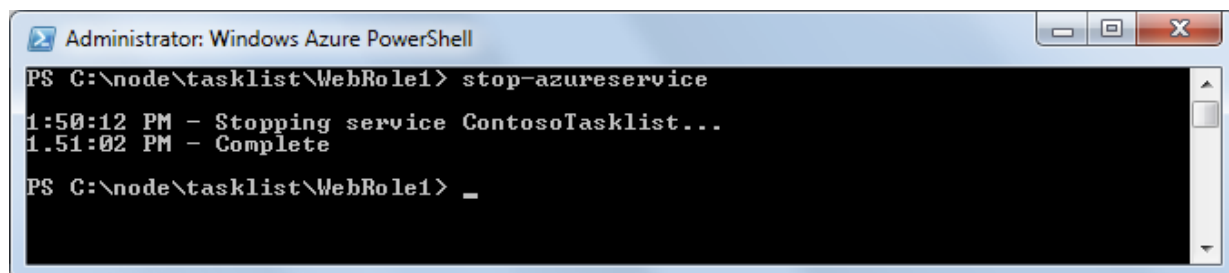
Windows Azure bills web role instances per hour of server time consumed. Server time is consumed once your application is deployed, even if the instances are not running and are in the stopped state.

The following steps show you how to stop and delete your application.

1. In the Windows PowerShell window, stop the service deployment created in the previous section with the following cmdlet:

```
PS C:\node\tasklist\WebRole1> Stop-AzureService
```

Stopping the service may take several minutes. When the service is stopped, you receive a message indicating that it has stopped.



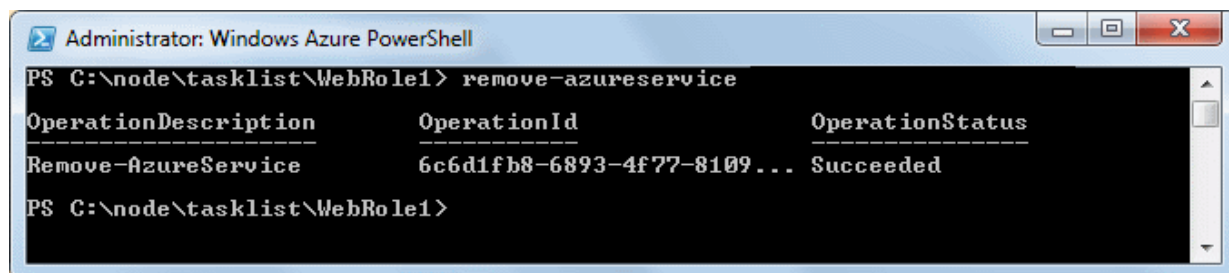
```
Administrator: Windows Azure PowerShell
PS C:\node\tasklist\WebRole1> stop-azureservice
1:50:12 PM - Stopping service ContosoTasklist...
1:51:02 PM - Complete
PS C:\node\tasklist\WebRole1> _
```

2. To delete the service, call the following cmdlet:

```
PS C:\node\tasklist\WebRole1> Remove-AzureService
```

When prompted, enter **Y** to delete the service.

Deleting the service may take several minutes. After the service has been deleted you receive a message indicating that the service was deleted.



```
Administrator: Windows Azure PowerShell
PS C:\node\tasklist\WebRole1> remove-azureservice
OperationDescription      OperationId                OperationStatus
-----
Remove-AzureService       6c6d1fb8-6893-4f77-8109... Succeeded
PS C:\node\tasklist\WebRole1>
```

Note Deleting the service does not delete the storage account that was created when the service was initially published, and you will continue to be billed for storage used. Since storage accounts can be used by multiple deployments, be sure that no other deployed service is using the storage account before you delete it. For more information on deleting a storage account, see [How to Delete a Storage Account from a Windows](#)

[Azure Subscription.](#)

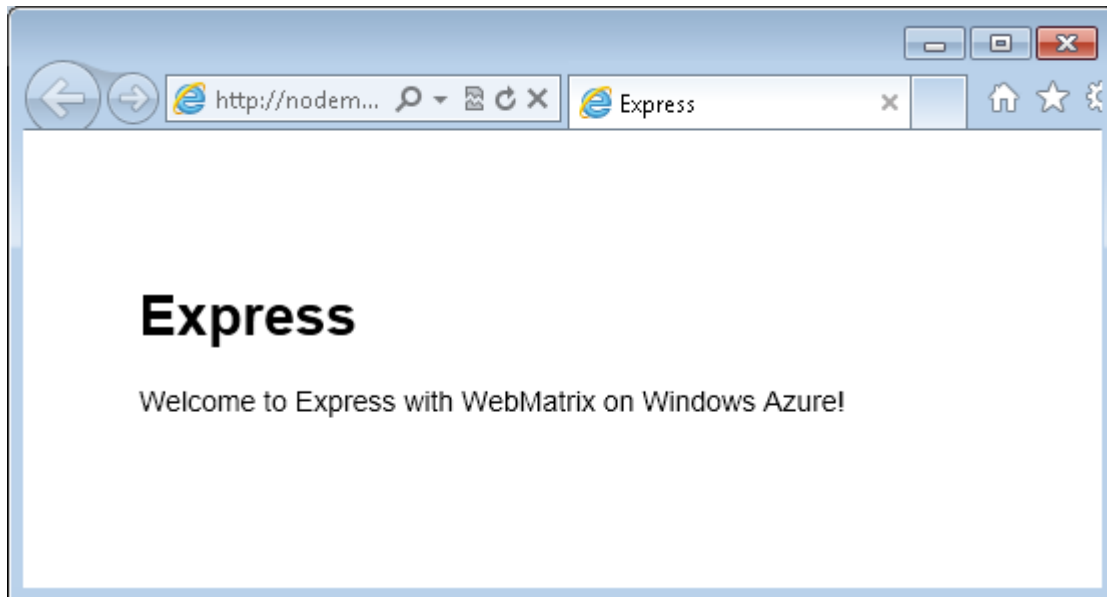
Create and deploy a Node.js application to a Windows Azure Web Site using WebMatrix

This tutorial shows you how to use WebMatrix to develop and deploy a Node.js application to a Windows Azure Website. WebMatrix is a free web development tool from Microsoft that includes everything you need for website development. WebMatrix includes several features that make it easy to use Node.js including code completion, pre-built templates, and editor support for Jade, LESS, and CoffeeScript. Learn more about [WebMatrix for Windows Azure](#).

Upon completing this guide, you will have a node web site running in Windows Azure. You will learn:

- How to create a web site from the Windows Azure Portal.
- How to develop a node application using WebMatrix.
- How to publish and re-publish your application to Windows Azure using WebMatrix.

By following this tutorial, you will build a simple node web application. The application will be hosted in a Windows Azure Web Site. A screenshot of the running application is below:

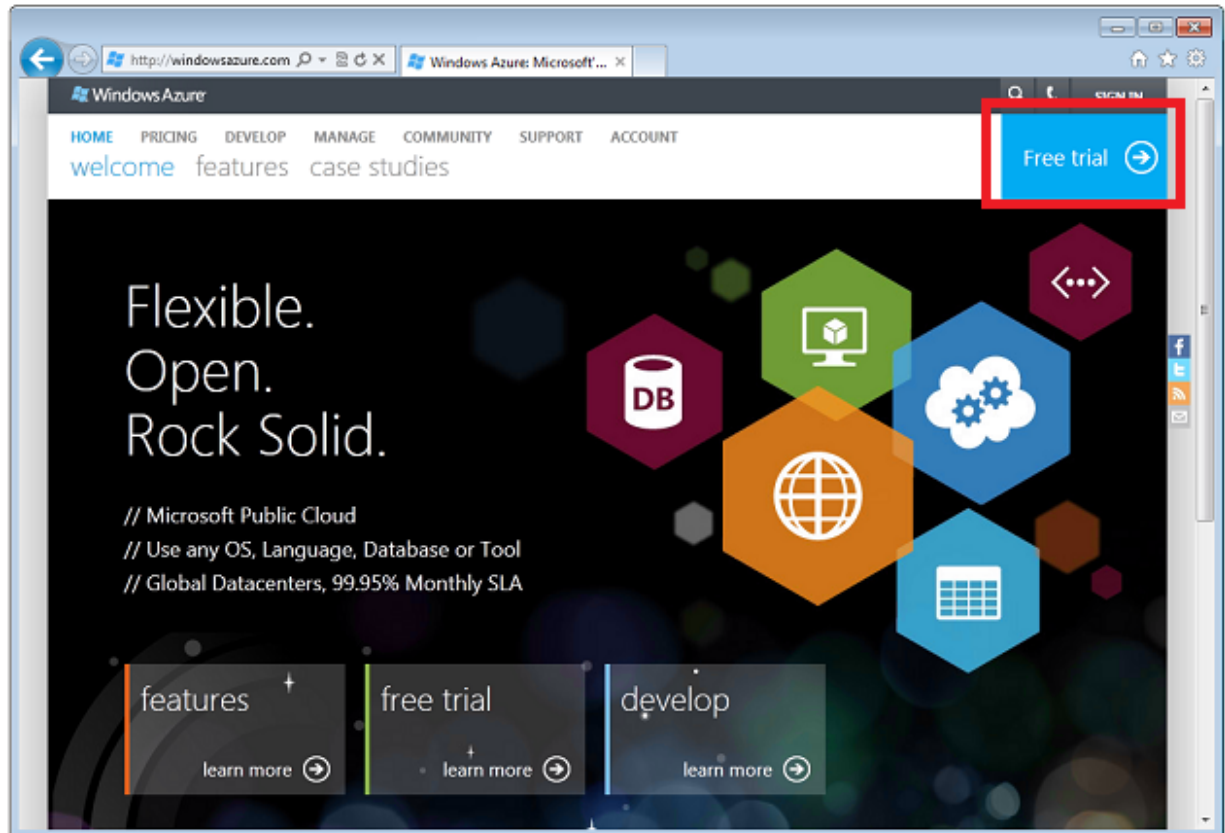


Set up the Windows Azure environment

First, set up the Windows Azure environment. You'll create a Windows Azure account and enable this account to use the Windows Azure Web Sites preview feature.

Create a Windows Azure account

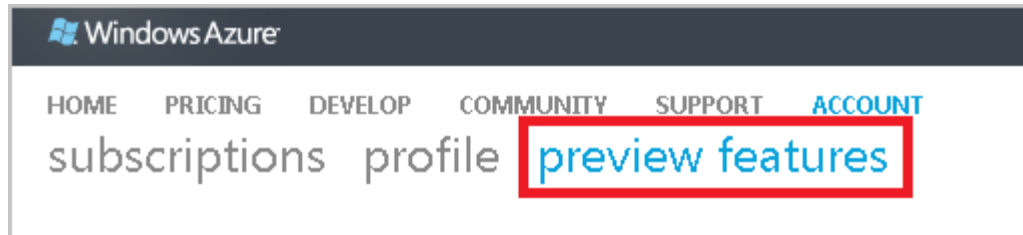
1. Open a web browser and browse to <http://www.windowsazure.com>.
2. To get started with a free account, click **Free Trial** in the upper-right corner and follow the steps. You'll need a credit card number and a mobile phone number for proof of identity, but you will not be billed.



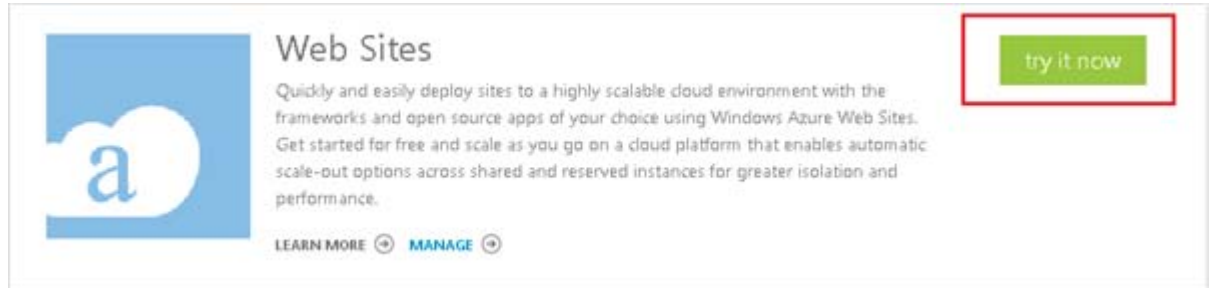
Enable Windows Azure Web Sites

After signing up, follow these steps to enable the Windows Azure Web Site feature.

1. Navigate to <https://account.windowsazure.com/> and sign in with your Windows Azure account.
2. Click **preview features** to view the available previews.




3. Scroll down to **Web Sites** and click **try it now**.



4. Select your subscription and click the check.

×

ADD PREVIEW FEATURE



WEB SITES
FREE

SUBSCRIPTIONS

3-Month Free Trial

▼

You will receive an e-mail when this feature is enabled for your subscription.

By clicking the Submit button, I agree to the Windows Azure [Preview Features Terms of Use](#) and [Privacy Statement](#), including the terms for Preview Releases.

✓

Create a Windows Azure Web Site

Follow these steps to create a Windows Azure Web Site.

1. Login to the [Windows Azure Portal](#).
2. Click the + **NEW** icon on the bottom left of the portal



3. Click **WEB SITE**, then **QUICK CREATE**. Enter a value for **URL** and select the datacenter for your web site in the **REGION** dropdown. Click the checkmark at the bottom of the dialog.

NEW

WEB SITE	QUICK CREATE
WEB SITE	QUICK CREATE
VIRTUAL MACHINE	CREATE WITH DATABASE
CLOUD SERVICE	FROM GALLERY
SQL DATABASE	
STORAGE	
NETWORK	

URL
[Text Field]
.antdf0.antares-test.windows-int.net

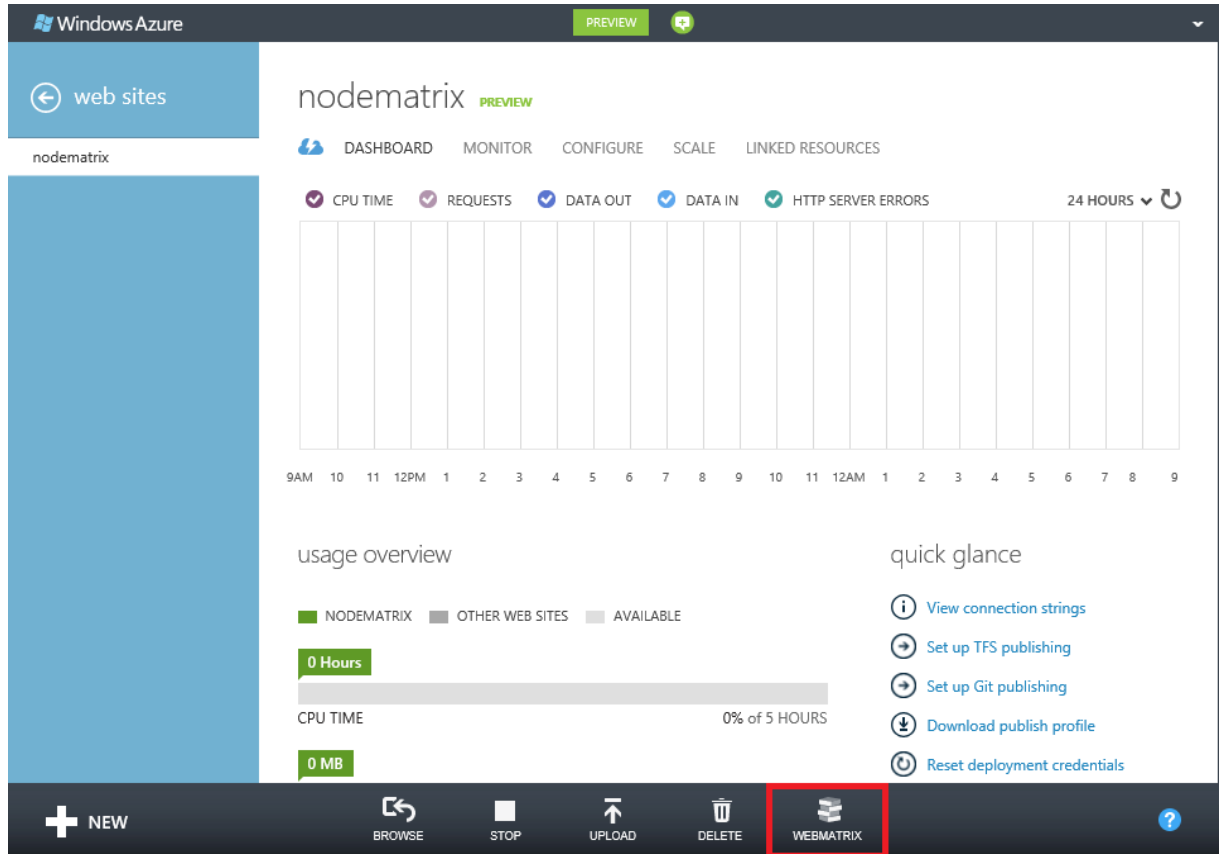
REGION
[Dropdown Menu]
South Central US

CREATE WEB SITE

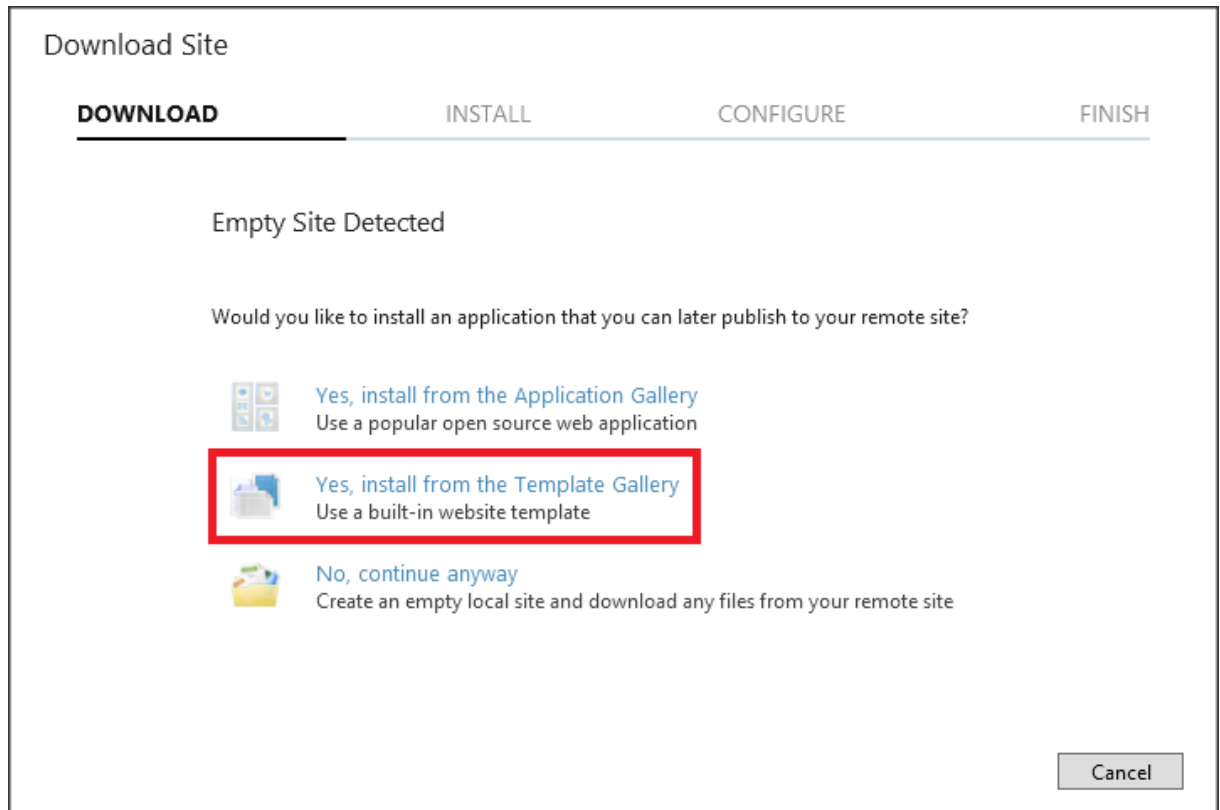
4. Once the web site is created, the portal will display all the web sites associated with your subscription. Verify that the web site you just created has a **Status** of **Running** and then click the web site name to view the **Dashboard** for this web site.

Import the web site into WebMatrix and apply the Express template

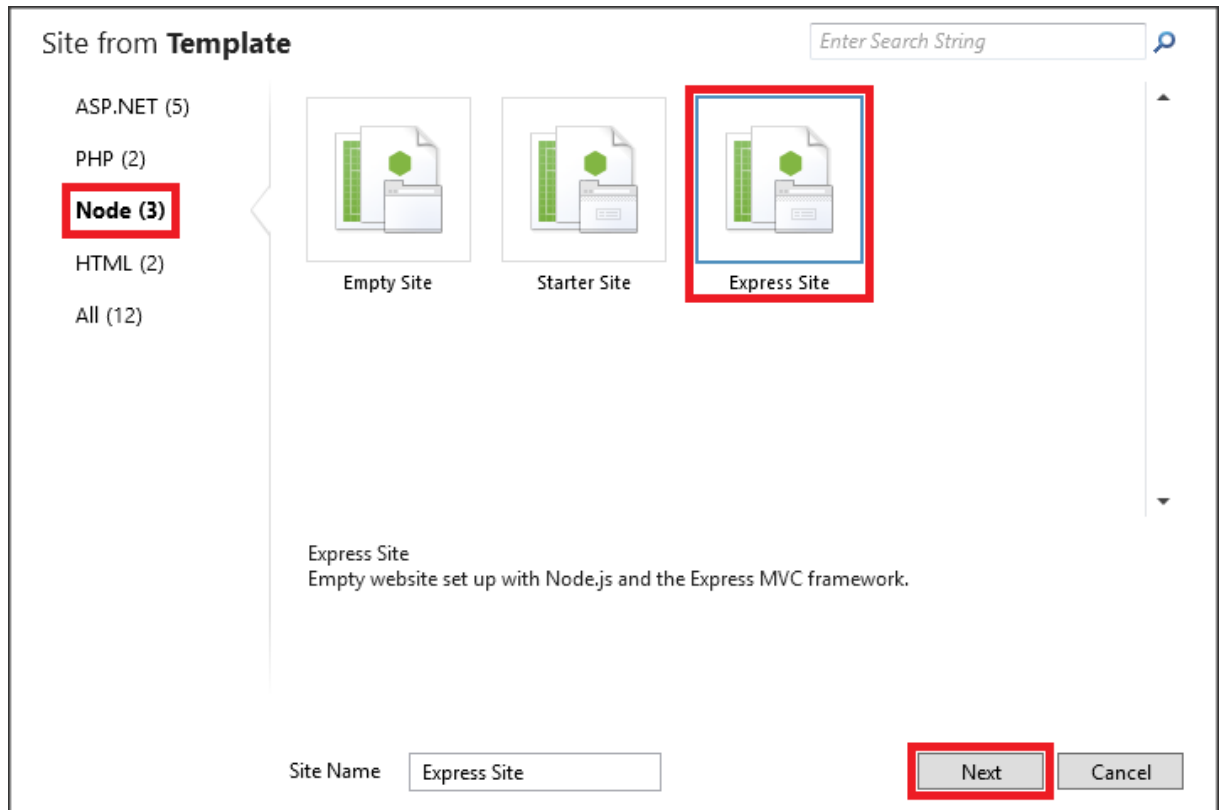
1. From the **Dashboard**, click the WebMatrix icon at the bottom of the page to open the web site in WebMatrix 2.



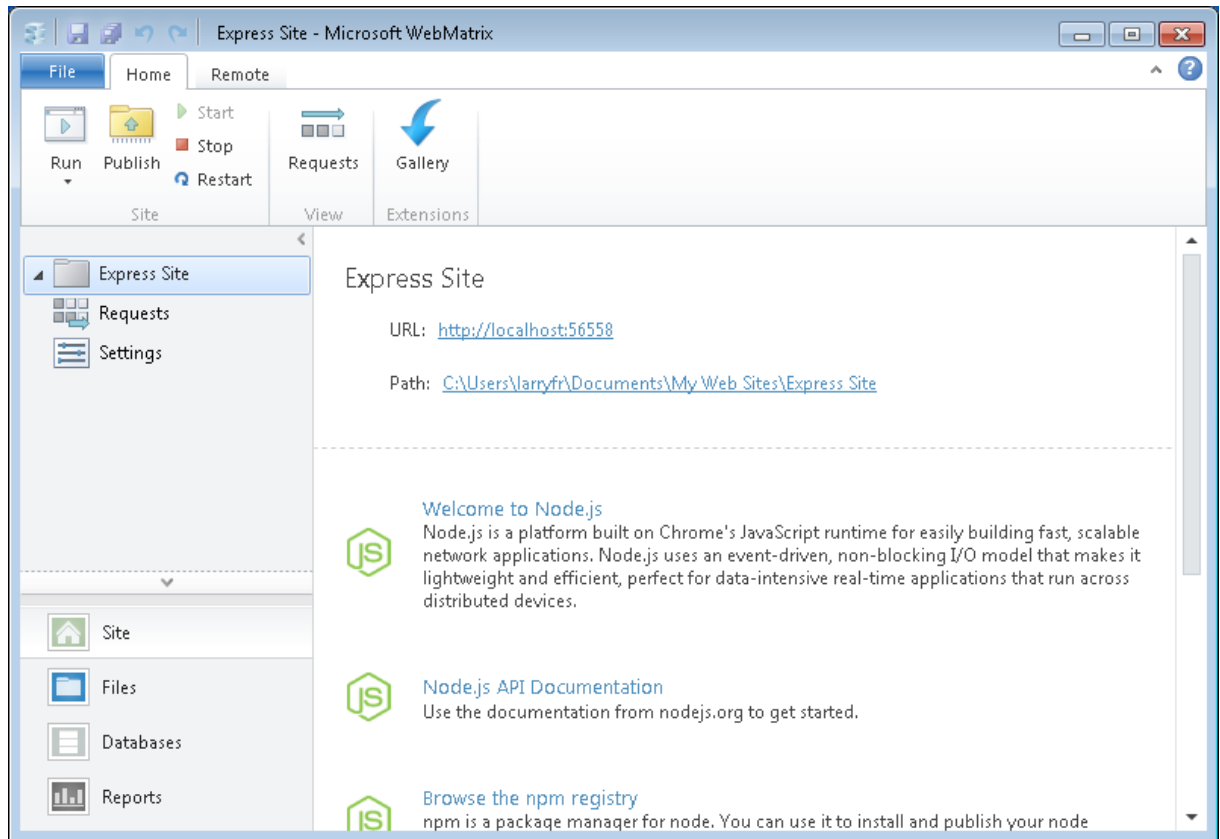
2. If WebMatrix 2 is not installed, Web Platform Installer 4.0 will install Microsoft WebMatrix 2 and all necessary prerequisites. WebMatrix will launch and display a dialog indicating **Empty Site Detected**. Click **Yes, install from the Template Gallery** to select a built-in template.



3. In the **Site from Template** dialog, select **Node** and then select **Express Site**. Finally, click **Next**. If you are missing any prerequisites for the **Express Site** template, you will be prompted to install them.

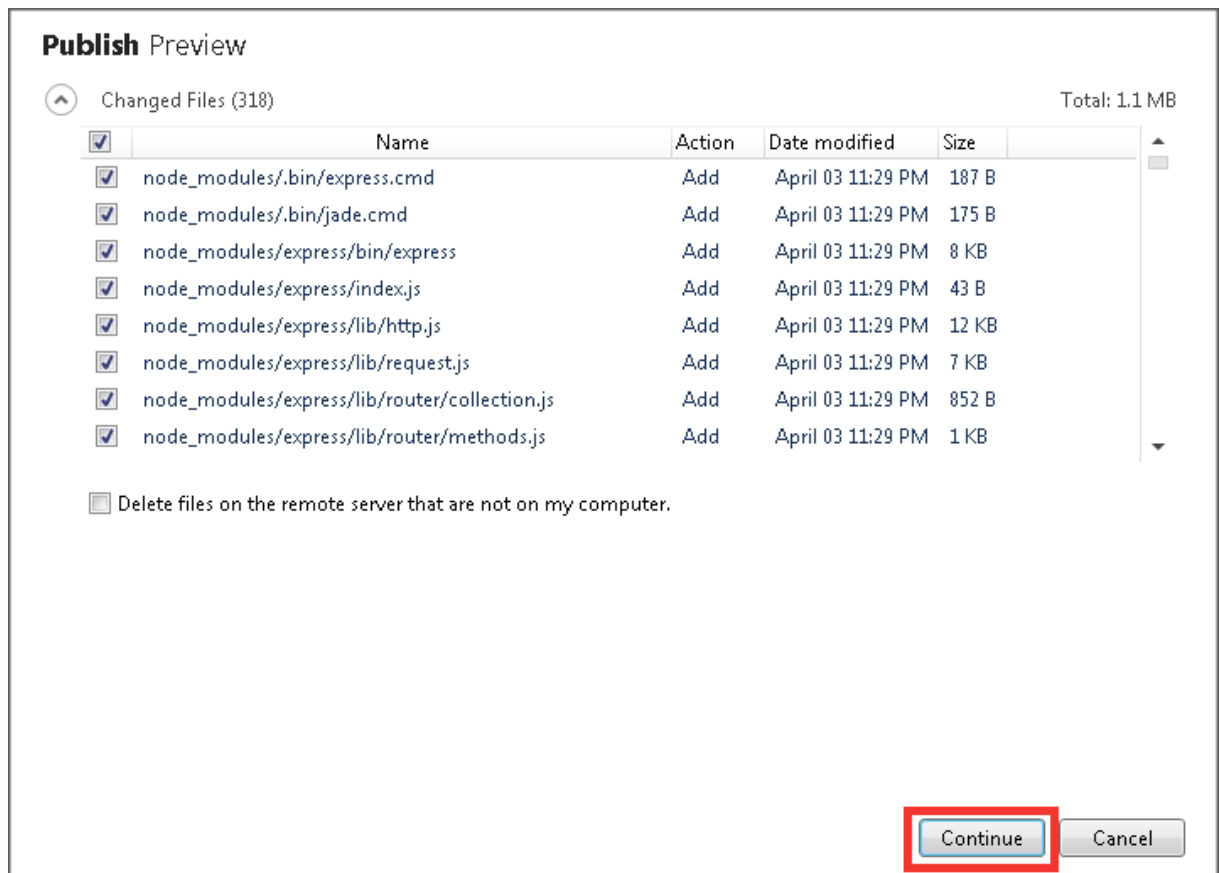


4. After WebMatrix finishes building the web site, the WebMatrix IDE is displayed.

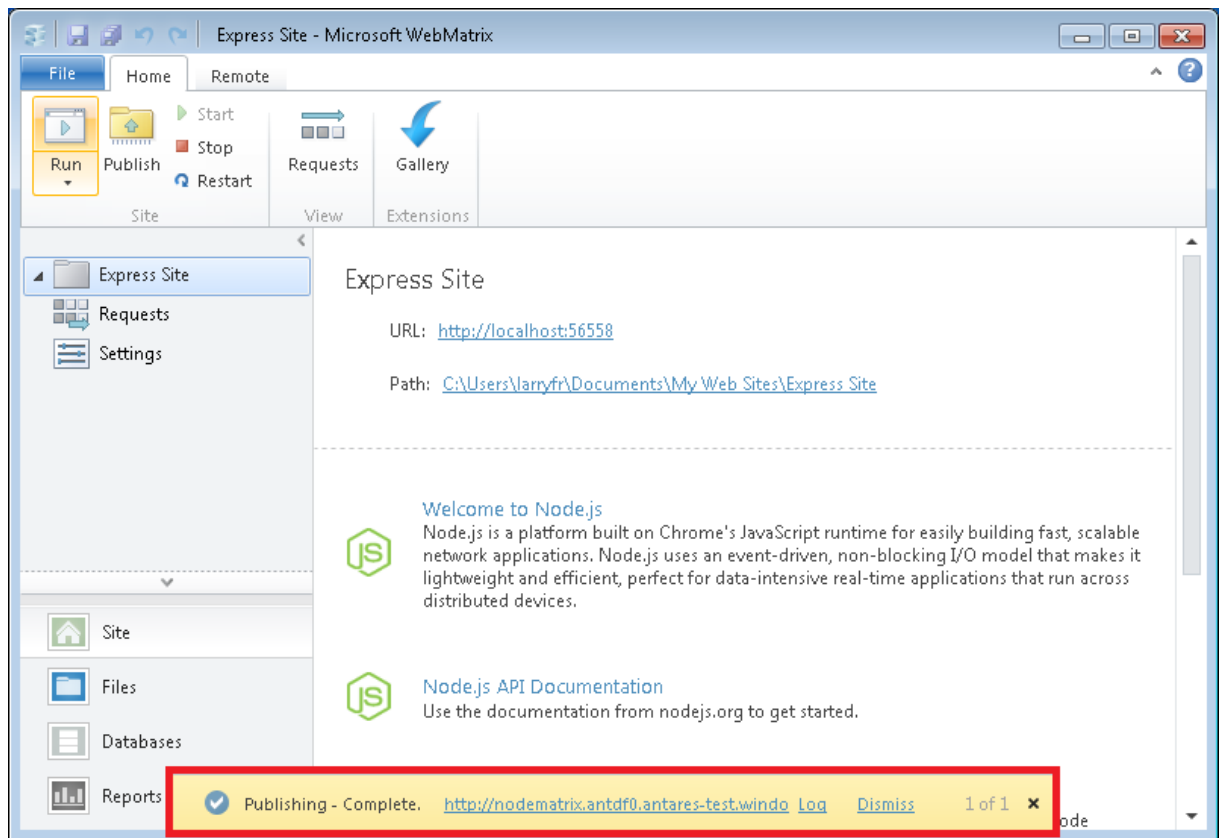


Publish your application to Windows Azure

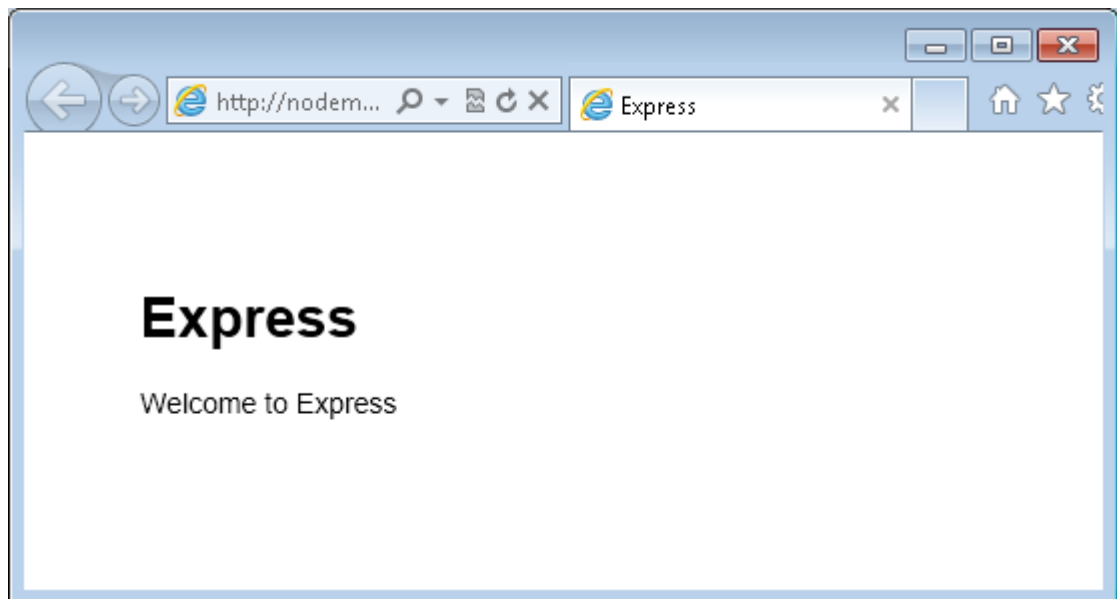
1. In WebMatrix, click **Publish** from the **Home** ribbon to display the **Publish Preview** dialog box for the web site.



2. Click **Continue**. When publishing is complete, the URL for the web site on Windows Azure is displayed at the bottom of the WebMatrix IDE



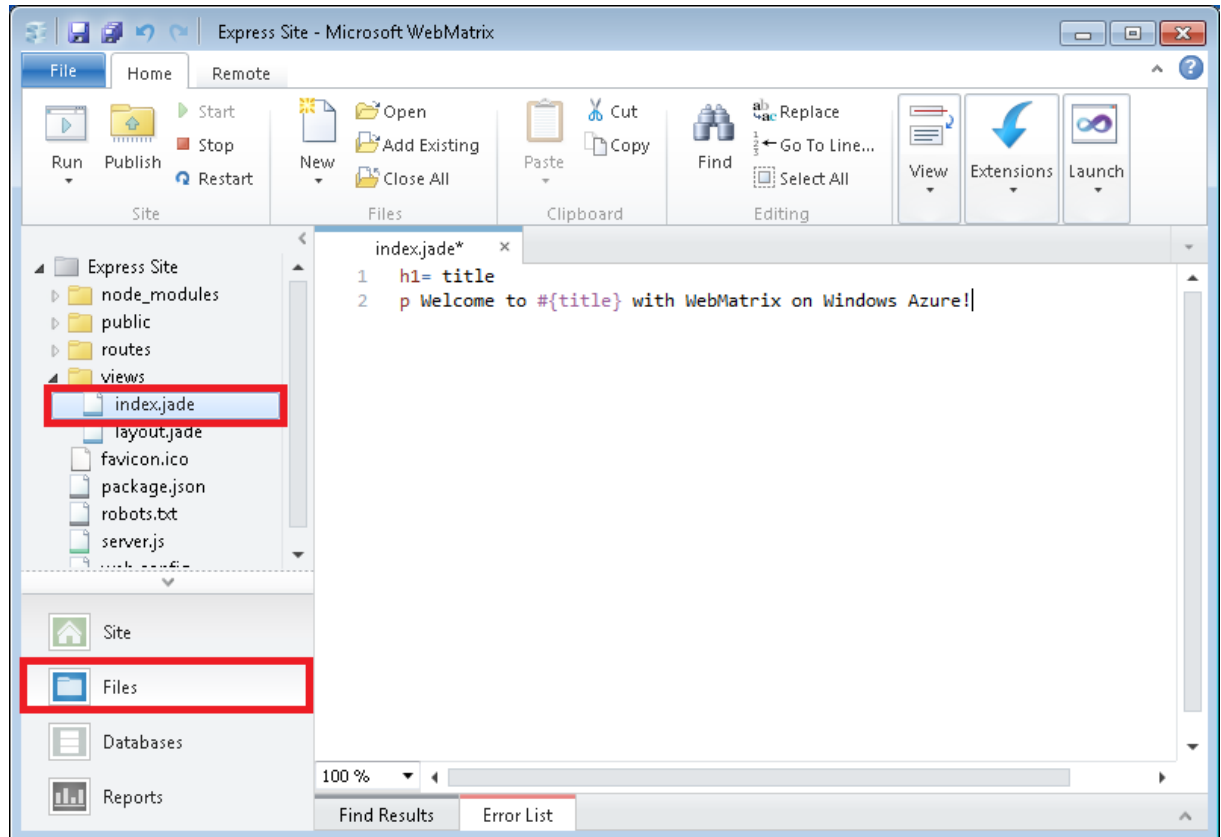
3. Click the link to open the web site in your browser.



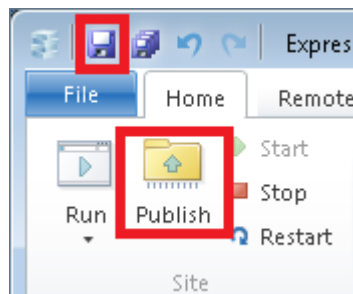
Modify and republish your application

You can easily modify and republish your application. Here, you will make a simple change to the heading in in the **index.jade** file, and republish the application.

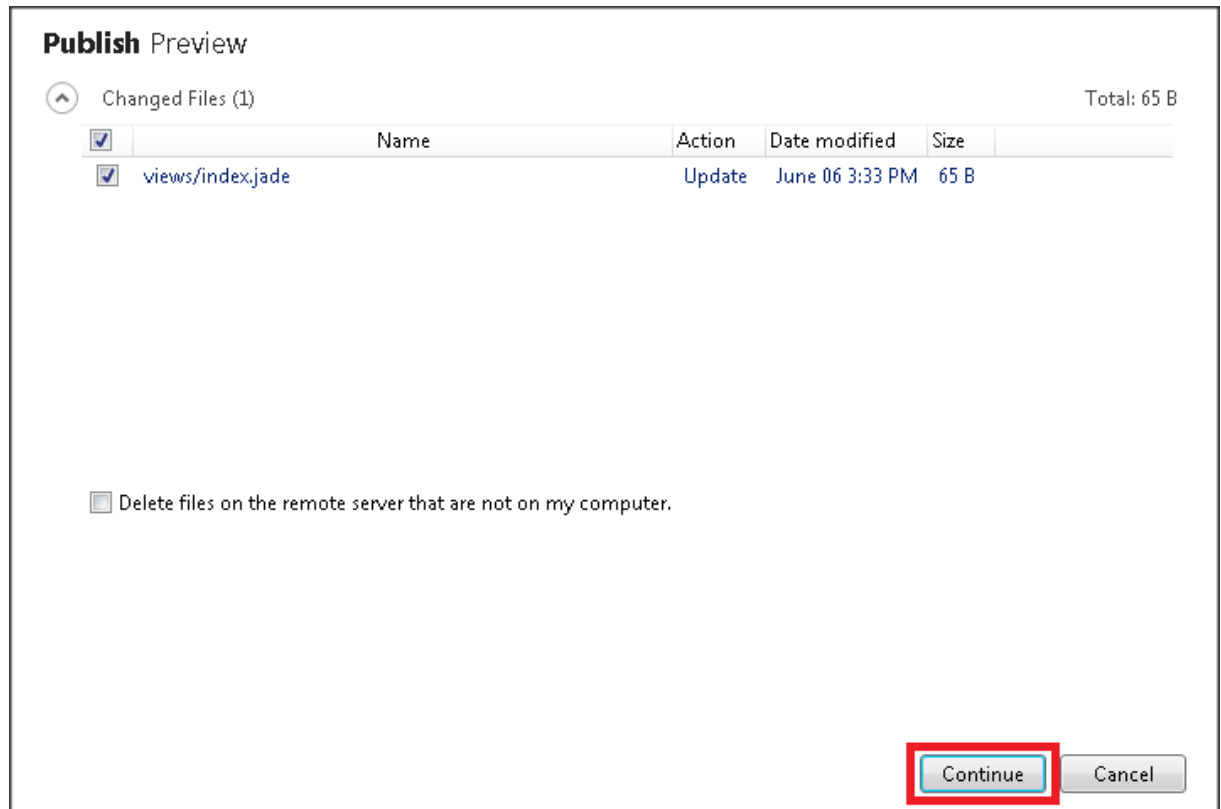
1. In WebMatrix, select **Files**, and then expand the **views** folder. Open the **index.jade** file by double-clicking it.



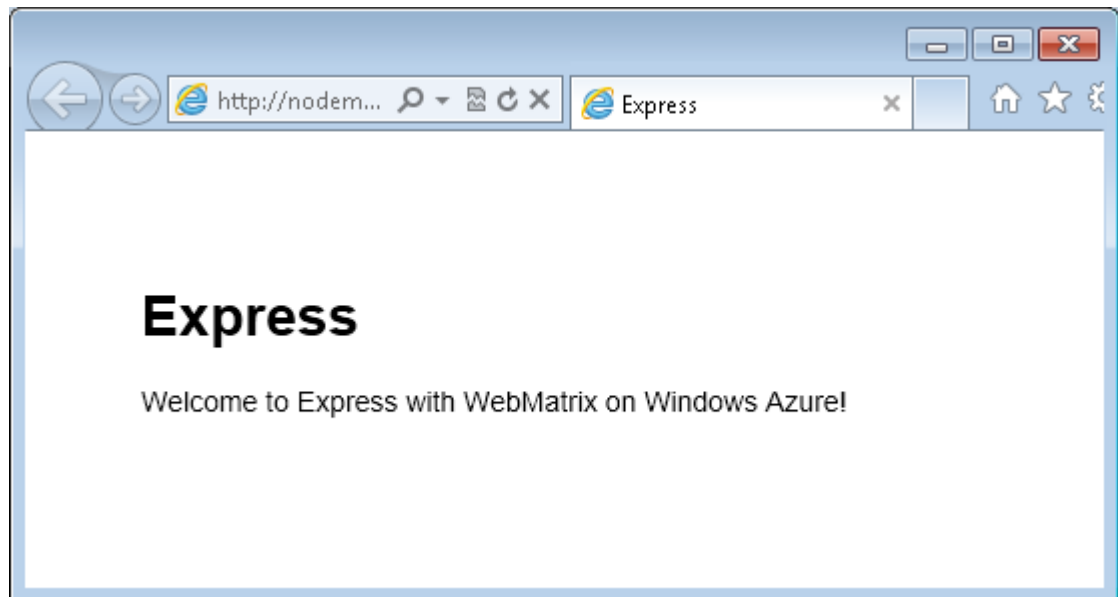
2. Change the second line to the following:
p Welcome to #{title} with WebMatrix on Windows Azure!
3. Click the save icon, and then click the publish icon.



4. Click **Continue** in the **Publish Preview** dialog and wait for the update to be published.



5. When publishing has completed, use the link returned when the publish process is complete to see the updated site.



Next Steps

You've seen how to create and deploy a web site from WebMatrix to Windows Azure. To learn more about WebMatrix, check out these resources:

[WebMatrix for Windows Azure](#)

[WebMatrix website](#)

[Publishing a Windows Azure Web site using Git](#)

Node.js Web Application with Storage on MongoDB

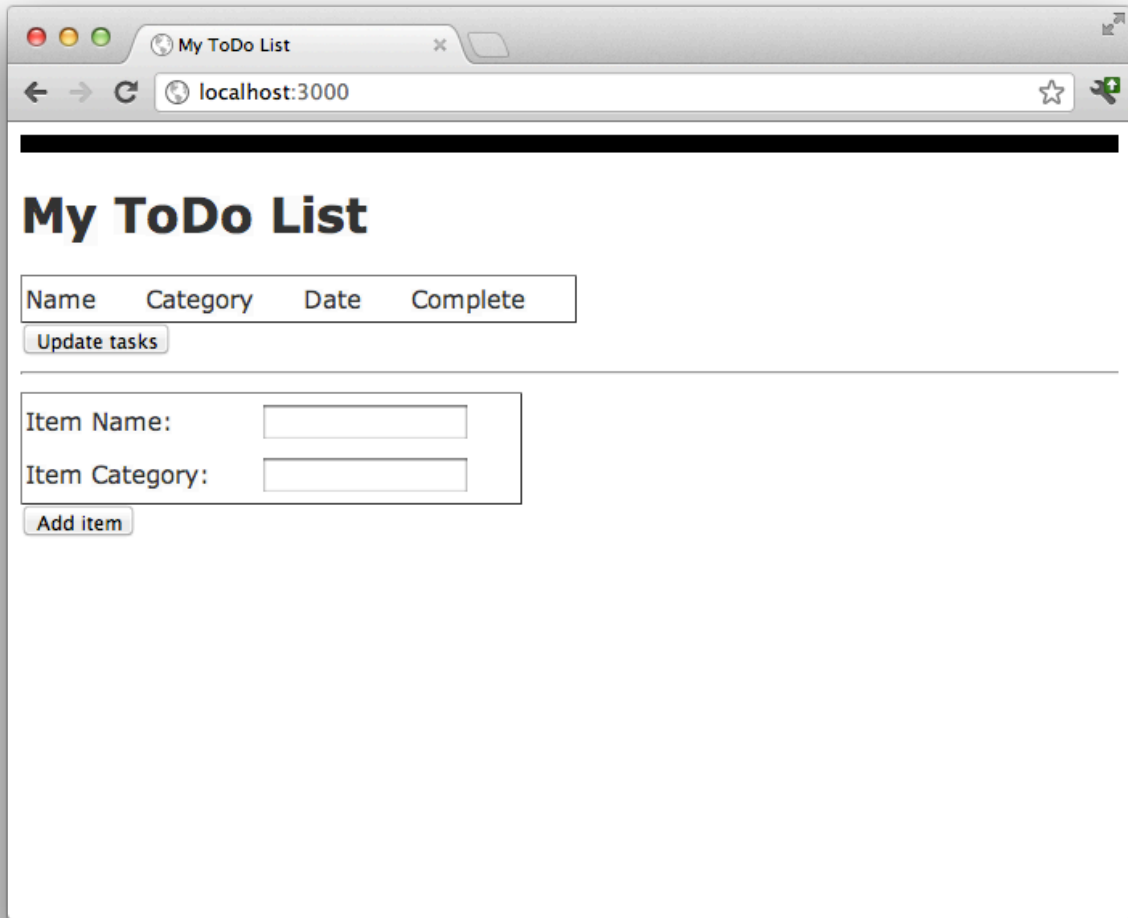
This tutorial shows you how to use [MongoDB](#) to store and access data from a [node](#) application hosted on Windows Azure. [MongoDB](#) is a popular open source, high performance NoSQL database. This tutorial assumes that you have some prior experience using node, MongoDB, and [Git](#).

You will learn:

- How to set up a virtual machine running Linux or Windows and install MongoDB
- How to use npm (node package manager) to install the node modules
- How to access MongoDB from a node application
- How to use the Cross-Platform Tools for Windows Azure to create a Windows Azure Web Site

By following this tutorial, you will build a simple web-based task-management application that allows creating, retrieving and completing tasks. The tasks are stored in MongoDB.

The project files for this tutorial will be stored in a directory named **tasklist** and the completed application will look similar to the following:



Note This tutorial makes reference to the **tasklist** folder. The full path to this folder is omitted, as path semantics differ between operating systems. You should create this folder in a location that is easy for you to access on your local file system, such as **~/node/tasklist** or **c:\node\tasklist**

Note Many of the steps below mention using the command-line. For these steps, use the command-line for your operating system, such as **Windows PowerShell** (Windows) or **Bash** (Unix Shell). On OS X systems you can access the command-line through the Terminal application.

Prerequisites

Before following the instructions in this article, you should ensure that you have the following installed:

- [node](#) recent version
- [Git](#)

Preparation

In this section you will learn how to create a virtual machine in Windows Azure and install MongoDB, set up your development environment, and install the MongoDB C# driver.

Create a virtual machine and install MongoDB

This tutorial assumes you have created a virtual machine in Windows Azure. After creating the virtual machine you need to install MongoDB on the virtual machine:

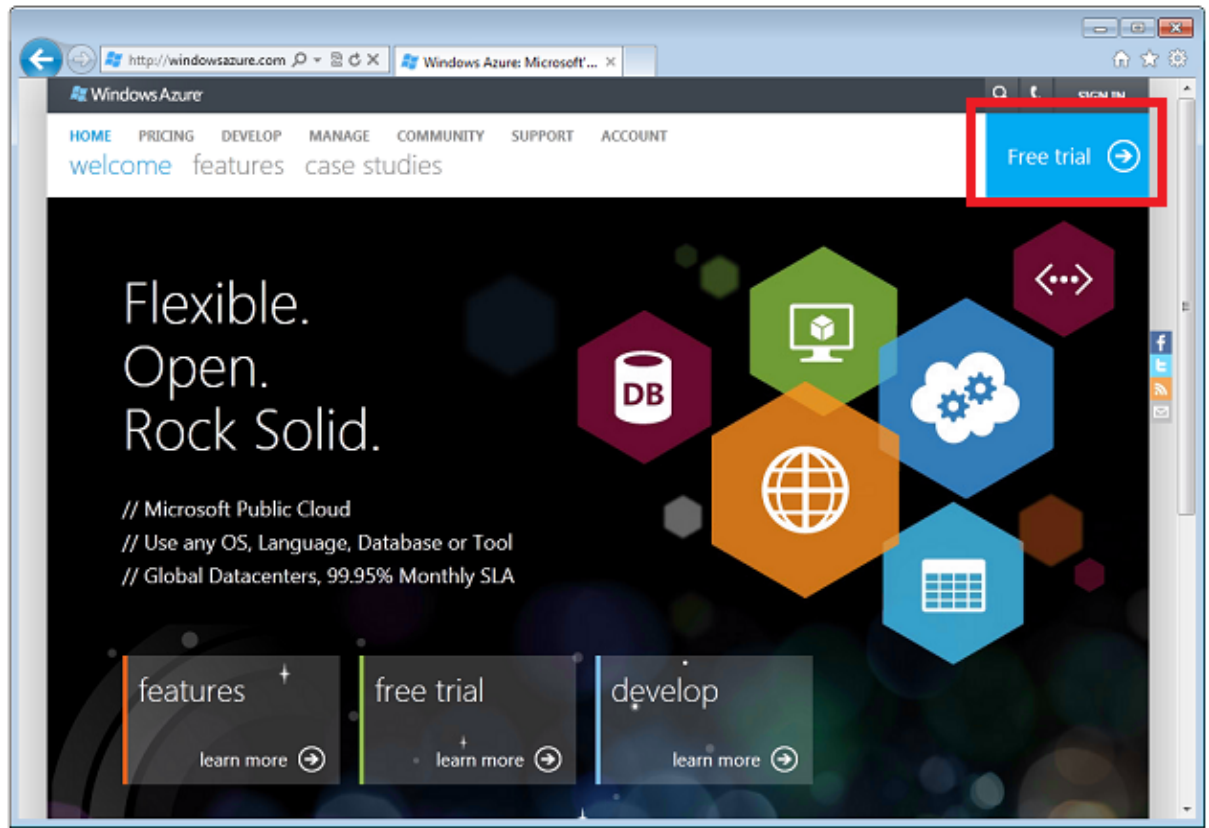
- To create a Linux virtual machine and install MongoDB, see [Installing MongoDB on a Linux Virtual machine](#).

After you have created the virtual machine in Windows Azure and installed MongoDB, be sure to remember the DNS name of the virtual machine ("testlinuxvm.cloudapp.net", for example) and the external port for MongoDB that you specified in the endpoint. You will need this information later in the tutorial.

Sign up for the Windows Azure Web Sites preview feature

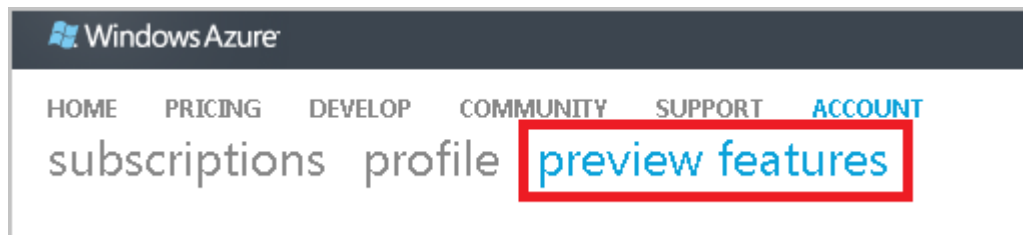
You will need to sign up for the Windows Azure Web Sites preview feature in order to create a Windows Azure web site. You can also sign up for a free trial account if you do not have a Windows Azure account.

1. Open a web browser and browse to <http://www.windowsazure.com>.
2. To get started with a free account, click **Free Trial** in the upper-right corner and follow the steps. You'll need a credit card number and a mobile phone number for proof of identity, but you will not be billed.




Enable Windows Azure Web Sites

1. Navigate to <https://account.windowsazure.com/> and sign in with your Windows Azure account.
2. Click **preview features** to view the available previews.



3. Scroll down to **Web Sites** and click **try it now**.



Web Sites


Quickly and easily deploy sites to a highly scalable cloud environment with the frameworks and open source apps of your choice using Windows Azure Web Sites. Get started for free and scale as you go on a cloud platform that enables automatic scale-out options across shared and reserved instances for greater isolation and performance.

[LEARN MORE](#) [MANAGE](#)

try it now

4. Select your subscription and click the check.

ADD PREVIEW FEATURE



WEB SITES
FREE

SUBSCRIPTIONS

3-Month Free Trial

You will receive an e-mail when this feature is enabled for your subscription.

By clicking the Submit button, I agree to the Windows Azure [Preview Features Terms of Use](#) and [Privacy Statement](#), including the terms for Preview Releases.

Install modules and generate scaffolding

In this section you will create a new Node application and use npm to add module packages. For the task-list application you will use the [Express](#) and [Mongoose](#) modules. The Express module provides a Model View Controller framework for node, while Mongoose is a driver for communicating with MongoDB.

Install express and generate scaffolding

1. From the command-line, change directories to the **tasklist** directory. If the **tasklist** directory does not exist, create it.
2. Enter the following command to install express.

```
sudo npm install express -g
```

Note When using the '-g' parameter on some operating systems, you may receive an error of **Error: EPERM, chmod '/usr/local/bin/express'** and a request to try running the account as an administrator. If this occurs, use the **sudo** command to run npm at a higher privilege level.

The output of this command should appear similar to the following:

```
express@2.5.9 /usr/local/lib/node_modules/express
├─ mime@1.2.4
├─ mkdirp@0.3.0
├─ qs@0.4.2
└─ connect@1.8.7
```

Note The '-g' parameter used when installing the express module installs it globally. This is done so that we can access the **express** command to generate web site scaffolding without having to type in additional path information.

3. To create the scaffolding which will be used for this application, use the **express** command:

```
express
```

The output of this command should appear similar to the following:

```
create : .
create : ./package.json
create : ./app.js
create : ./public
create : ./public/javascripts
create : ./public/images
create : ./public/stylesheets
create : ./public/stylesheets/style.css
create : ./routes
create : ./routes/index.js
create : ./views
create : ./views/layout.jade
```

```
create : ./views/index.jade
```

don't forget to install dependencies:

```
$ cd . && npm install
```

After this command completes, you should have several new directories and files in the **tasklist** directory.

Install additional modules

The **package.json** file is one of the files created by the **express** command. This file contains a list of additional modules that are required for an Express application. Later, when you deploy this application to a Windows Azure Web Site, this file will be used to determine which modules need to be installed on Windows Azure to support your application.

1. From the command-line, change directories to the **tasklist** folder and enter the following to install the modules described in the **package.json** file:

```
npm install
```

The output of this command should appear similar to the following:

```
express@2.5.8 ./node_modules/express
```

```
├─ mime@1.2.4
```

```
├─ qs@0.4.2
```

```
├─ mkdirp@0.3.0
```

```
└─ connect@1.8.7
```

```
jade@0.26.0 ./node_modules/jade
```

```
├─ commander@0.5.2
```

```
└─ mkdirp@0.3.0
```

This installs all of the default modules that Express needs.

2. Next, enter the following command to install the Mongoose module locally as well as to save an entry for it to the **package.json** file:

```
npm install mongoose --save
```

The output of this command should appear similar to the following:

```
mongoose@2.6.5 ./node_modules/mongoose
```

```
├─ hooks@0.2.1
```

Note You can safely ignore any message about installing the C++ bson parser.

Using MongoDB in a node application

In this section you will extend the basic application created by the **express** command by adding a **task.js** file which contains the model for your tasks. You will also modify the existing **app.js** and create a new **tasklist.js** controller file to make use of the model.

Create the model

1. In the **tasklist** directory, create a new directory named **models**.
2. In the **models** directory, create a new file named **task.js**. This file will contain the model for the tasks created by your application.
3. At the beginning of the **task.js** file, add the following code to reference required libraries:

```
var mongoose = require('mongoose')
    , Schema = mongoose.Schema;
```

4. Next, you will add code to define and export the model. This model will be used to perform interactions with the MongoDB database.

```
var TaskSchema = new Schema({
  itemName      : String
  , itemCategory : String
  , itemCompleted : { type: Boolean, default: false }
  , itemDate     : { type: Date, default: Date.now }
});
```

```
module.exports = mongoose.model('TaskModel', TaskSchema)
```

5. Save and close the **task.js** file.

Create the controller

1. In the **tasklist/routes** directory, create a new file named **tasklist.js** and open it in a text editor.
2. Add the following code to **tasklist.js**. This loads the mongoose module and the task model defined in **task.js**. The TaskList function is used to create the connection to the MongoDB server based on the **connection** value:

```
var mongoose = require('mongoose')
    , task = require('../models/task.js');
```

```
module.exports = TaskList;
```

```
function TaskList(connection) {  
  mongoose.connect(connection);  
}
```

3. Continue adding to the **tasklist.js** file by adding the methods used to **showTasks**, **addTask**, and **completeTasks**:

```
TaskList.prototype = {  
  showTasks: function(req, res) {  
    task.find({itemCompleted: false}, function  
foundTasks(err, items) {  
      res.render('index',{title: 'My ToDo List ', tasks:  
items})  
    });  
  },
```

```
  addTask: function(req,res) {  
    var item = req.body.item;  
    newTask = new task();  
    newTask.itemName = item.name;  
    newTask.itemCategory = item.category;  
    newTask.save(function savedTask(err){  
      if(err) {  
        throw err;  
      }  
    });  
    res.redirect('home');  
  },
```

```
  completeTask: function(req,res) {  
    var completedTasks = req.body;  
    for(taskId in completedTasks) {  
      if(completedTasks[taskId]=='true') {  
        var conditions = { _id: taskId };  
        var updates = { itemCompleted:  
completedTasks[taskId] };  
        task.update(conditions, updates, function  
updatedTask(err) {  
          if(err) {  
            throw err;  
          }  
        });  
      }  
    }  
  }  
};
```

```

    }
  }
  res.redirect('home');
}
}

```

4. Save the **tasklist.js** file.

Modify app.js

1. In the **tasklist** directory, open the **app.js** file in a text editor. This file was created earlier by running the **express** command.
2. Replace the content after the `//Routes` comment with the following code. This will initialize **TaskList** with the connection string for the MongoDB server and add the functions defined in **tasklist.js** as routes:

```

var TaskList = require('./routes/tasklist');
var taskList = new
TaskList('mongodb://mongodbserver/tasks');

app.get('/', taskList.showTasks.bind(taskList));
app.post('/addtask', taskList.addTask.bind(taskList));
app.post('/completetask',
taskList.completeTask.bind(taskList));

app.listen(process.env.port || 3000);

```

Note You must replace the connection string above with the connection string for the MongoDB server you created earlier. For example, **`mongodb://mymongodb.cloudapp.net/tasks`**

3. Save the **app.js** file.

Modify the index view

1. Change directories to the **views** directory and open the **index.jade** file in a text editor.
2. Replace the contents of the **index.jade** file with the code below. This defines the view for displaying existing tasks, as well as a form for adding new tasks and marking existing ones as completed.

```

h1= title
form(action="/completetask", method="post")

```

```

table(border="1")
  tr
    td Name
    td Category
    td Date
    td Complete
  each task in tasks
    tr
      td #{task.itemName}
      td #{task.itemCategory}
      - var day    = task.itemDate.getDate();
      - var month  = task.itemDate.getMonth() + 1;
      - var year   = task.itemDate.getFullYear();
      td #{month + "/" + day + "/" + year}
      td
        input(type="checkbox", name="#{task._id}",
value="#{!task.itemCompleted}",
checked=task.itemCompleted)
        input(type="submit", value="Update tasks")
  hr
  form(action="/addtask", method="post")
    table(border="1")
      tr
        td Item Name:
        td
          input(name="item[name]", type="textbox")
      tr
        td Item Category:
        td
          input(name="item[category]", type="textbox")
          input(type="submit", value="Add item")

```

3. Save and close **index.jade** file.

Run your application locally

To test the application on your local machine, perform the following steps:

1. From the command-line, change directories to the **tasklist** directory.
2. Use the following command to launch the application locally:

```
node app.js
```
3. Open a web browser and navigate to <http://localhost:3000>. This should display a web page similar to the following:

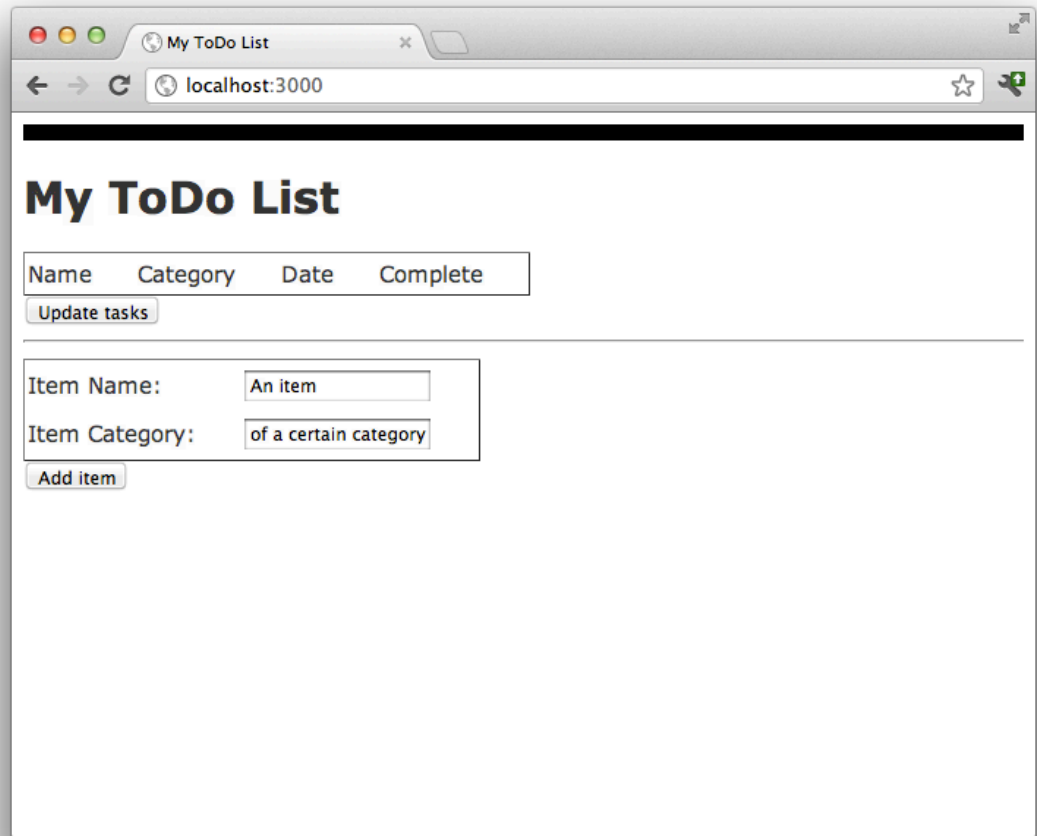
The screenshot shows a web browser window with the title 'My ToDo List' and the address bar displaying 'localhost:3000'. The page features a main heading 'My ToDo List' and a table with columns: Name, Category, Date, and Complete. Below the table is an 'Update tasks' button. A form section contains two input fields labeled 'Item Name:' and 'Item Category:', followed by an 'Add item' button.

Name	Category	Date	Complete
------	----------	------	----------

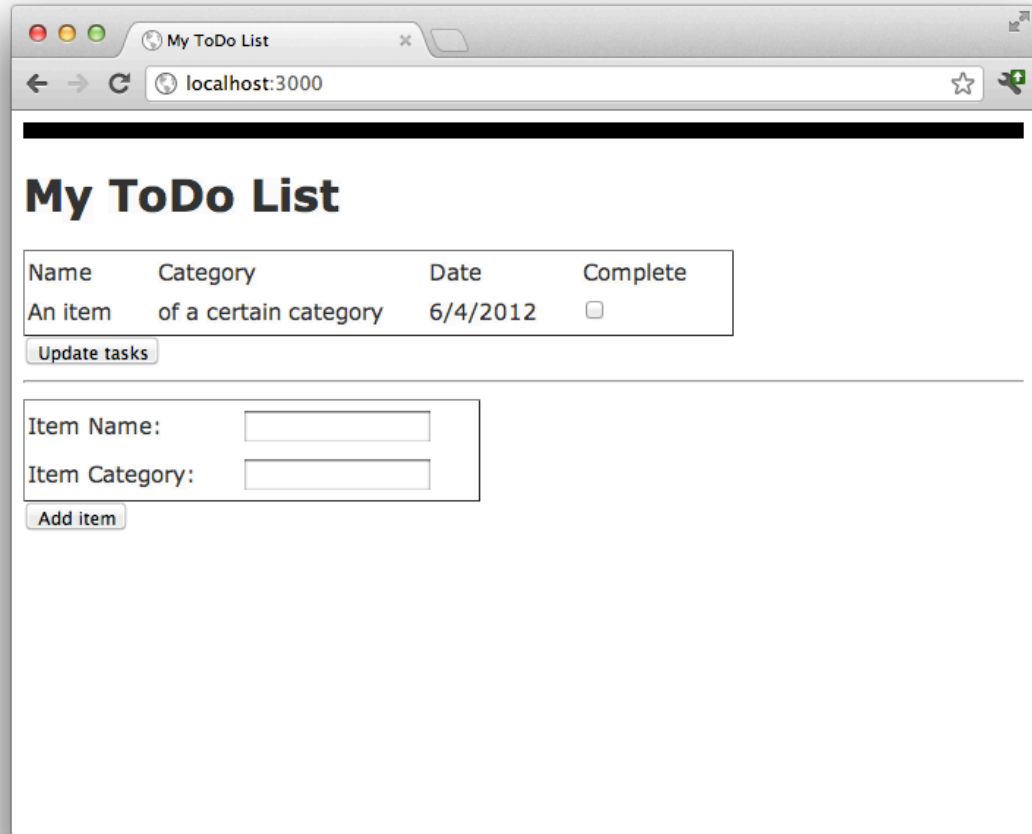
Item Name:

Item Category:

4. Use the provided fields for **Item Name** and **Item Category** to enter information, and then click **Add item**.



5. The page should update to display the item in the ToDo List table.



6. To complete a task, simply check the checkbox in the Complete column, and then click **Update tasks**. While there is no visual change after clicking **Update tasks**, the document entry in MongoDB has now been marked as completed.
7. To stop the node process, go to the command-line and press the **CTRL** and **C** keys.

Deploy your application to Windows Azure

The steps in this section use the Windows Azure command-line tools to create a new Windows Azure Web Site, and then use Git to deploy your application. To perform these steps you must have a Windows Azure subscription.

Note These steps can also be performed by using the Windows Azure portal. For steps on using the Windows Azure portal to deploy a Node.js application, see [Create and deploy a Node.js application to a Windows Azure Web Site](#).

Note If this is the first Windows Azure Web Site you have created, you must use the

Windows Azure portal to deploy this application.

Install the Windows Azure command-line tool for Mac and Linux

To install the command-line tools, use the following command:

```
sudo npm install azure -g
```

Note If you have already installed the **Windows Azure SDK for Node.js** from the [Windows Azure Developer Center](#), then the command-line tools should already be installed. For more information, see [Windows Azure command-line tool for Mac and Linux](#).

Note While the command-line tools were created primarily for Mac and Linux users, they are based on Node.js and should work on any system capable of running Node.

Import publishing settings

Before using the command-line tools with Windows Azure, you must first download a file containing information about your subscription. Perform the following steps to download and import this file.

1. From the command-line, enter the following command to launch the browser and navigate to the download page. If prompted, login with the account associated with your subscription.

```
azure account download
```



Your .publishSettings file has been generated, and the download should have already started. Click [here](#) if the download does not start.

Configure the *azure* shell command to use your account by following these steps:

- 1) [Sign up for Azure preview features](#) such as Web Sites and Virtual Machines.

- 2) Save the .publishSettings file locally.

Security note: *This file contains an encoded management certificate that will serve as your credentials to administer all aspects of your subscriptions and related services. Store this file in a secure location, or delete it after you follow these steps.*

- 3) Import the downloaded .publishSettings file by running the command:

```
azure account import <publishSettings-file>
```

- 4) Create an Azure website by running the following command in an empty folder:

```
azure site create <unique-website-name> --git
```

The file download should begin automatically; if it does not, you can click the link at the beginning of the page to manually download the file.

2. After the file download has completed, use the following command to import the settings:

```
azure account import <path-to-file>
```

Specify the path and file name of the publishing settings file you downloaded in the previous step. Once the command completes, you should see output similar to the following:

```
info:    Executing command account import
info:    Found subscription: subscriptionname
info:    Setting default subscription to: subscriptionname
warn:    The '/Users/user1/.azure/publishSettings.xml' file
contains sensitive information.
warn:    Remember to delete it now that it has been
imported.
info:    Account publish settings imported successfully
info:    account iomport command OK
```

3. Once the import has completed, you should delete the publish settings file as it is no longer needed and contains sensitive information regarding your Windows Azure subscription.

Create a Windows Azure Web Site

1. From the command-line, change directories to the **tasklist** directory.
2. Use the following command to create a new Windows Azure Web Site. Replace 'myuniquesitename' with a unique site name for your web site. This value is used as part of the URL for the resulting web site.

```
azure site create myuniquesitename --git
```

You will be prompted for the datacenter that the site will be located in. Select the datacenter geographically close to your location.

The `--git` parameter will create a Git repository locally in the **tasklist** folder if none exists. It will also create a [Git remote](#) named 'azure', which will be used to publish the application to Windows Azure. It will create an [iisnode.yml](#), which contains settings used by Windows Azure to host node applications. Finally it will also create a .gitignore file to exclude the node-modules folder for being published to .git.

Note If this command is ran from a directory that already contains a Git repository, it will not re-initialize the directory.

Note If the '`--git`' parameter is omitted, yet the directory contains a Git repository, the 'azure' remote will still be created.

Once this command has completed, you will see output similar to the following. Note that the line beginning with **Created website at** contains the URL for the web site.

```
info:   Executing command site create
info:   Using location southcentraluswebspace
info:   Executing `git init`
info:   Creating default web.config file
info:   Creating a new web site
info:   Created website at
mongodbtasklist.azurewebsites.net
info:   Initializing repository
info:   Repository initialized
info:   Executing `git remote add azure
http://username@mongodbtasklist.azurewebsites.net/mongodbtas
klist.git`
info:   site create command OK
```

Note If this is the first Windows Azure Web Site for your subscription, you will be instructed to use the portal to create the web site. For more information, see [Create and deploy a Node.js application to Windows Azure Web Sites](#).

Publish the application

1. In the Terminal window, change directories to the **tasklist** directory if you are not already there.
2. Use the following commands to add, and then commit files to the local Git repository:

```
git add .
git commit -m "adding files"
```

3. When pushing the latest Git repository changes to the Windows Azure Web Site, you must specify that the target branch is **master** as this is used for the web site content.

```
git push azure master
```

You will see output similar to the following. As the deployment takes place Windows Azure will download all npm modules.

```
Counting objects: 17, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (13/13), done.
Writing objects: 100% (17/17), 3.21 KiB, done.
```

```
Total 17 (delta 0), reused 0 (delta 0)
remote: New deployment received.
remote: Updating branch 'master'.
remote: Preparing deployment for commit id 'ef276f3042'.
remote: Preparing files for deployment.
remote: Running NPM.
...
remote: Deploying Web.config to enable Node.js activation.
remote: Deployment successful.
To
https://username@mongodbtasklist.azurewebsites.net/MongoDBT
asklist.git
* [new branch]      master -> master
```

4. Once the push operation has completed, browse to the web site by using the `azure site browse` command to view your application.

Next steps

While the steps in this article describe using MongoDB to store information, you can also use the Windows Azure Table Service. See [Node.js Web Application with the Windows Azure Table Service](#) for more information.

Additional resources

[Windows Azure command-line tool for Mac and Linux](#)

[Create and deploy a Node.js application to Windows Azure Web Sites](#)

[Publishing to Windows Azure Web Sites with Git](#)

Node.js Web Application using the Windows Azure SQL Database

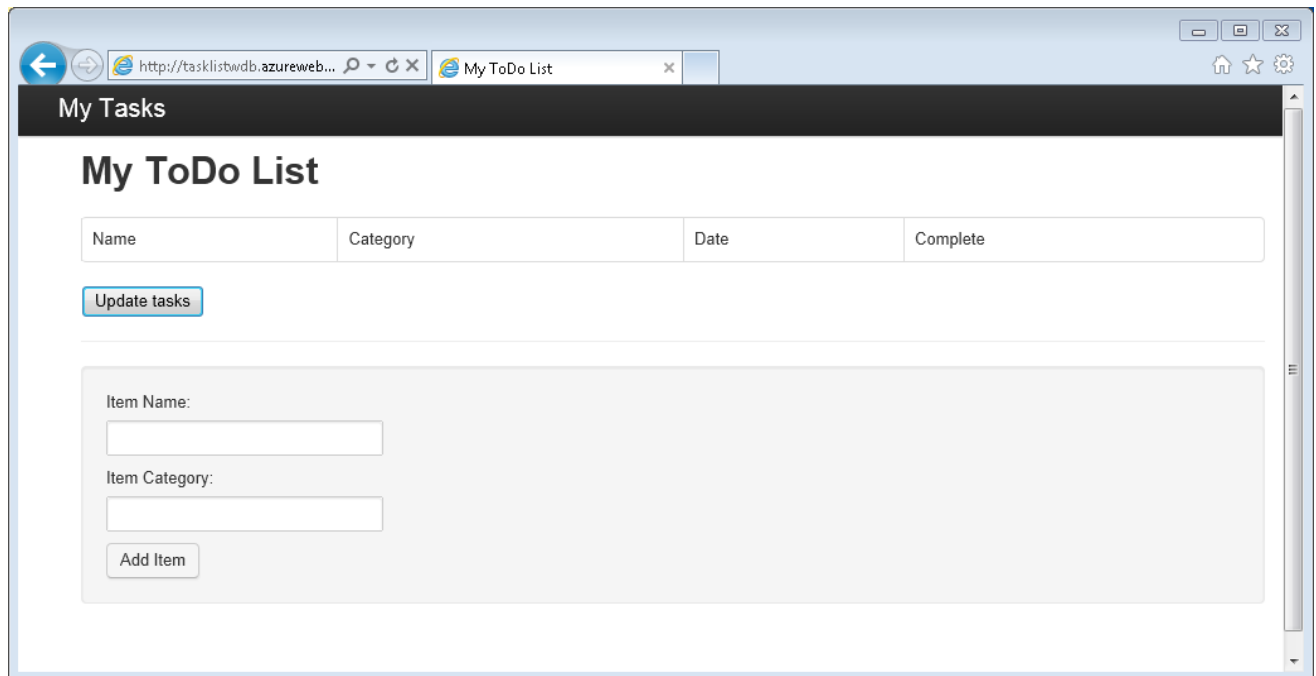
This tutorial shows you how to use SQL Database provided by Windows Azure Data Management to store and access data from a [node](#) application hosted on Windows Azure. This tutorial assumes that you have some prior experience using node and [Git](#).

You will learn:

- How to use the Windows Azure preview portal to create a Windows Azure Web Site and SQL Database
- How to use npm (node package manager) to install the node modules
- How to work with a SQL Database using the node-sqlserver module
- How to use app settings to specify run-time values for an application

By following this tutorial, you will build a simple web-based task-management application that allows creating, retrieving and completing tasks. The tasks are stored in SQL Database.

The project files for this tutorial will be stored in a directory named **tasklist** and the completed application will look similar to the following:



Note The Microsoft Driver for node.js for SQL Server used in this tutorial is currently available as a preview release, and relies on run-time components that are only available on the Microsoft Windows and Windows Azure operating systems.

Note This tutorial makes reference to the **tasklist** folder. The full path to this folder is omitted, as path semantics differ between operating systems. You should create this folder in a location that is easy for you to access on your local file system, such as ~/node/tasklist or c:\node\tasklist

Note Many of the steps below mention using the command-line. For these steps, use the command-line for your operating system, such as cmd.exe (Windows) or bash (UNIX shell). On OS X systems you can access the command-line through the terminal application.

Prerequisites

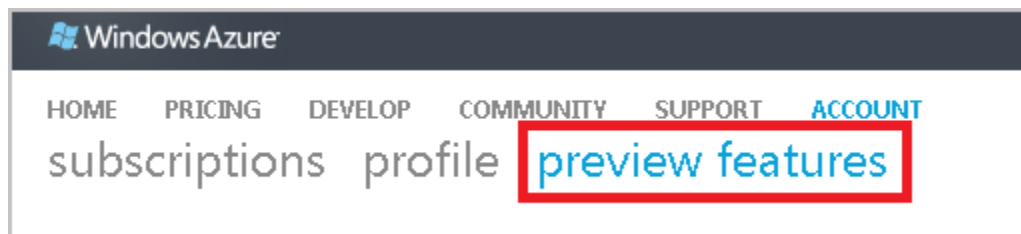
Before following the instructions in this article, you should ensure that you have the following installed:

- [node](#) version 0.6.14 or higher
- [Git](#)
- Microsoft SQL Server Native Client libraries - available as part of the [Microsoft SQL Server 2012 Feature Pack](#)
- A text editor
- A web browser

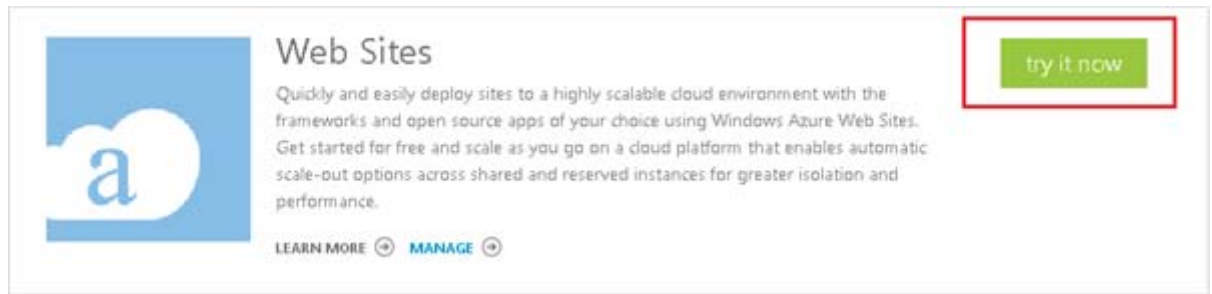
Enable the Windows Azure Web Site feature

If you do not already have a Windows Azure subscription, you can sign up [for free](#). After signing up, follow these steps to enable the Windows Azure Web Site feature.

1. Navigate to <https://account.windowsazure.com/> and sign in with your Windows Azure account.
2. Click **preview features** to view the available previews.



3. Scroll down to **Web Sites** and click **try it now**.



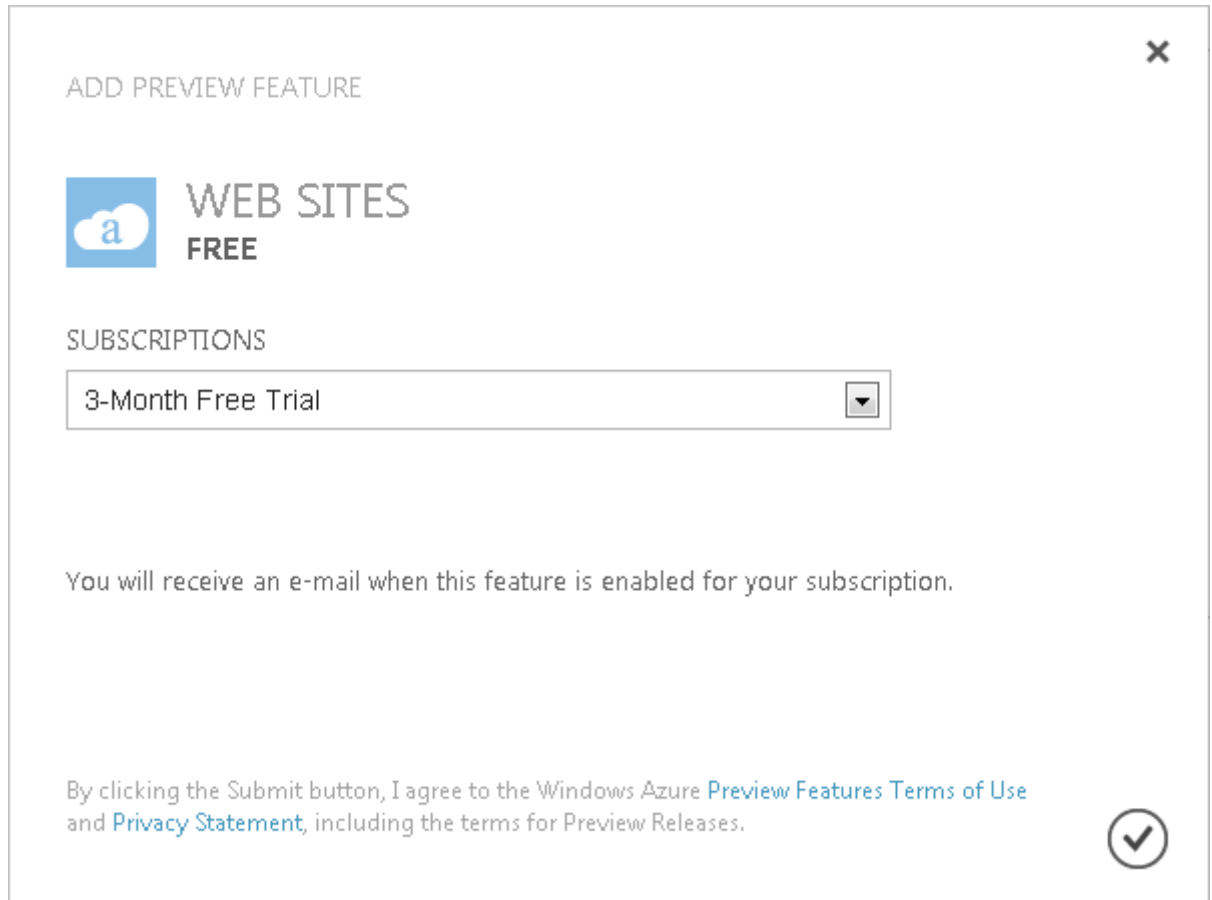
Web Sites

Quickly and easily deploy sites to a highly scalable cloud environment with the frameworks and open source apps of your choice using Windows Azure Web Sites. Get started for free and scale as you go on a cloud platform that enables automatic scale-out options across shared and reserved instances for greater isolation and performance.


[LEARN MORE](#) [MANAGE](#)

try it now

4. Select your subscription and click the check.



ADD PREVIEW FEATURE

 **WEB SITES**
FREE

SUBSCRIPTIONS

3-Month Free Trial

You will receive an e-mail when this feature is enabled for your subscription.

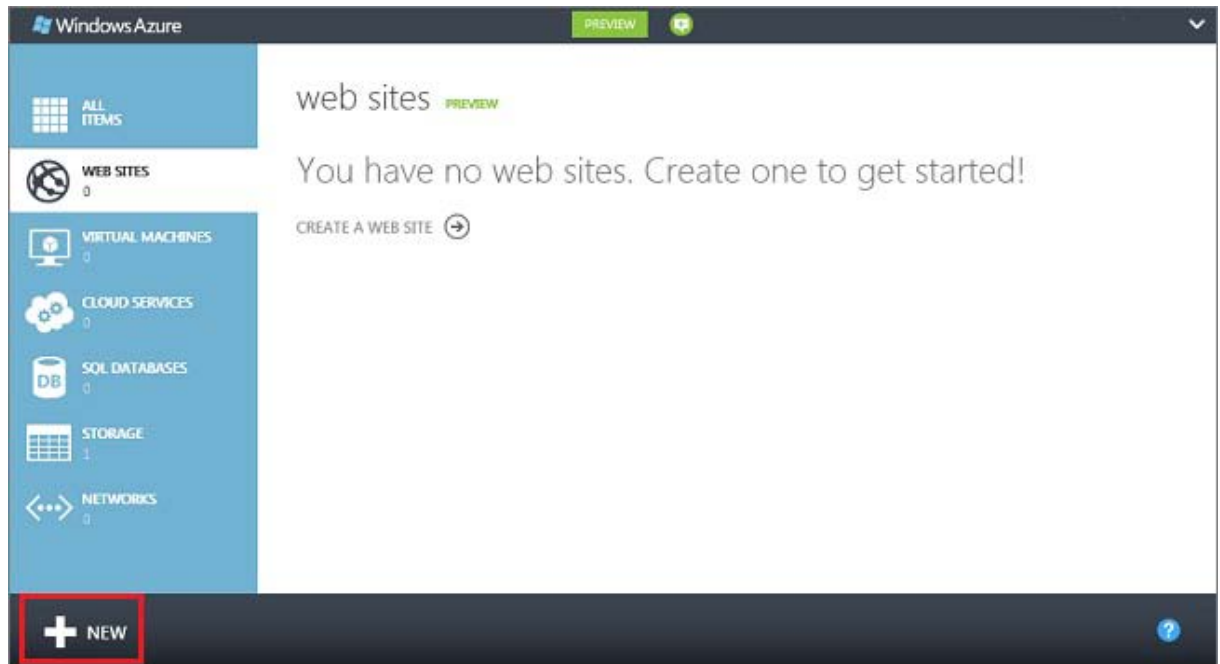
By clicking the Submit button, I agree to the Windows Azure [Preview Features Terms of Use](#) and [Privacy Statement](#), including the terms for Preview Releases.

☒

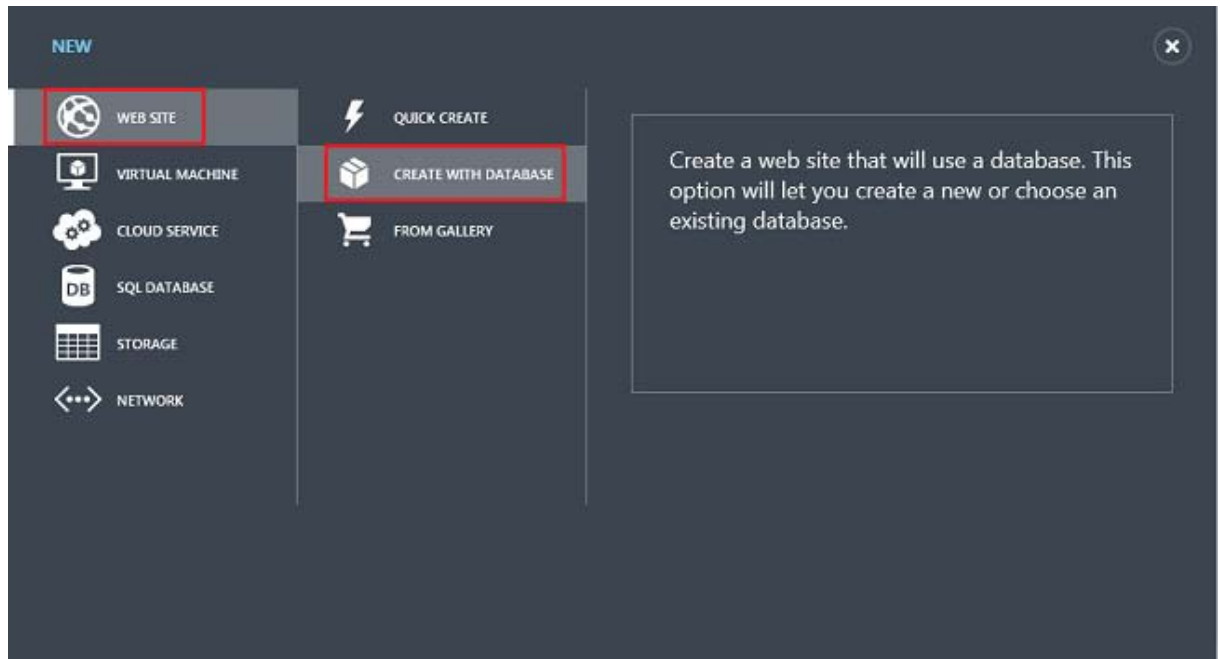
Create a web site with database

Follow these steps to create a Windows Azure Web Site and a SQL Database:

1. Login to the [Preview Management Portal](#).
2. Click the + **New** icon on the bottom left of the portal.



3. Click WEB SITE, then CREATE WITH DATABASE.



Enter a value for **URL**, select **Create a New SQL Database** from the **DATABASE** dropdown, and select the data center for your web site in the **REGION** dropdown. Click the arrow at the bottom of the dialog.

NEW WEB SITE - CREATE WITH DATABASE ×

Create web site

URL

testsite ✓

.antdf0.antares-test.windows-int.net

DATABASE

Create a new SQL database ▼

REGION

South Central US ▼

➔

2 3

4. Enter a value for the **NAME** of your database, select the **EDITION** ([WEB](#) or [BUSINESS](#)), select the **MAX SIZE** for your database, choose the **COLLATION**, and select **NEW SQL Database server**. Click the arrow at the bottom of the dialog.

NEW WEB SITE - CREATE WITH DATABASE

Specify database settings

NAME

EDITION

☒ WEB ☐ BUSINESS

MAXIMUM SIZE

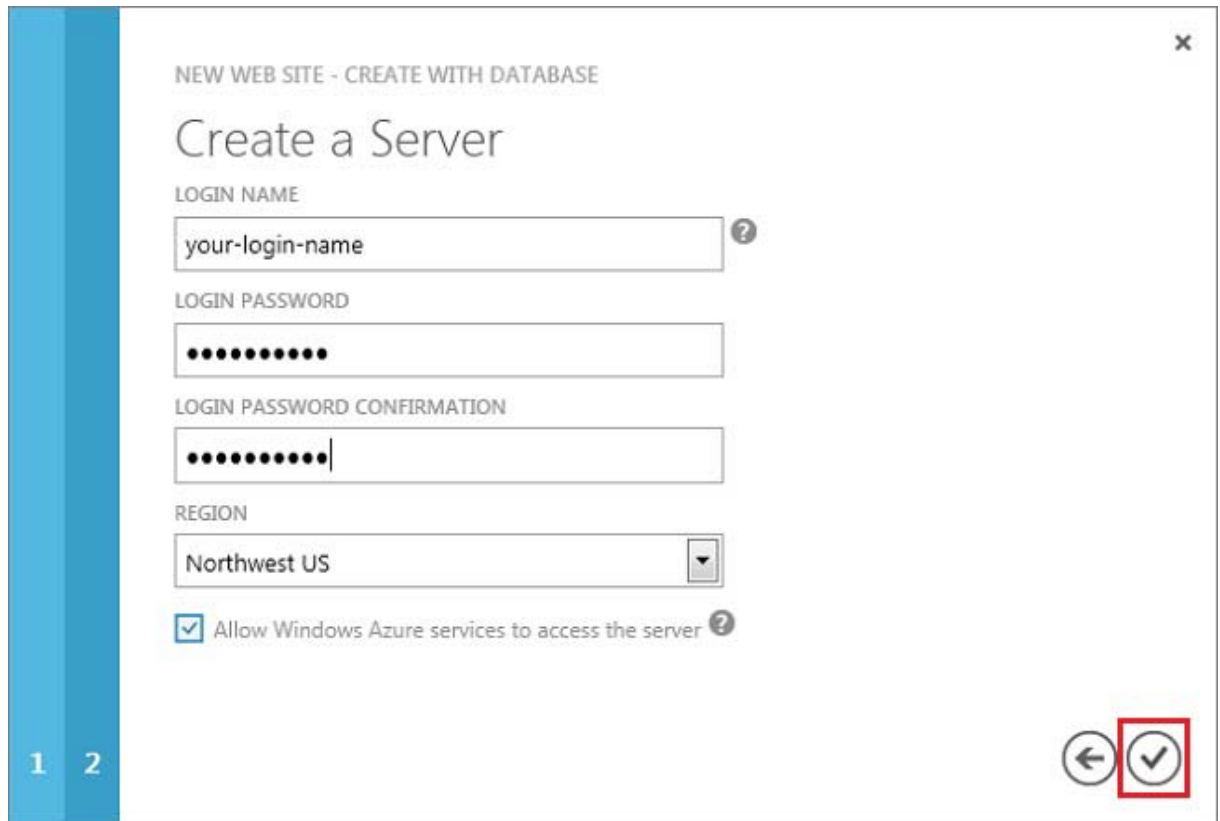
COLLATION

SERVER

1 3

← →

5. Enter an administrator name and password (and confirm the password), choose the region in which your new SQL Database server will be created, and check the **Allow Windows Azure Services to access the server** box.



NEW WEB SITE - CREATE WITH DATABASE

Create a Server

LOGIN NAME

 ?

LOGIN PASSWORD

LOGIN PASSWORD CONFIRMATION

REGION

 ▼

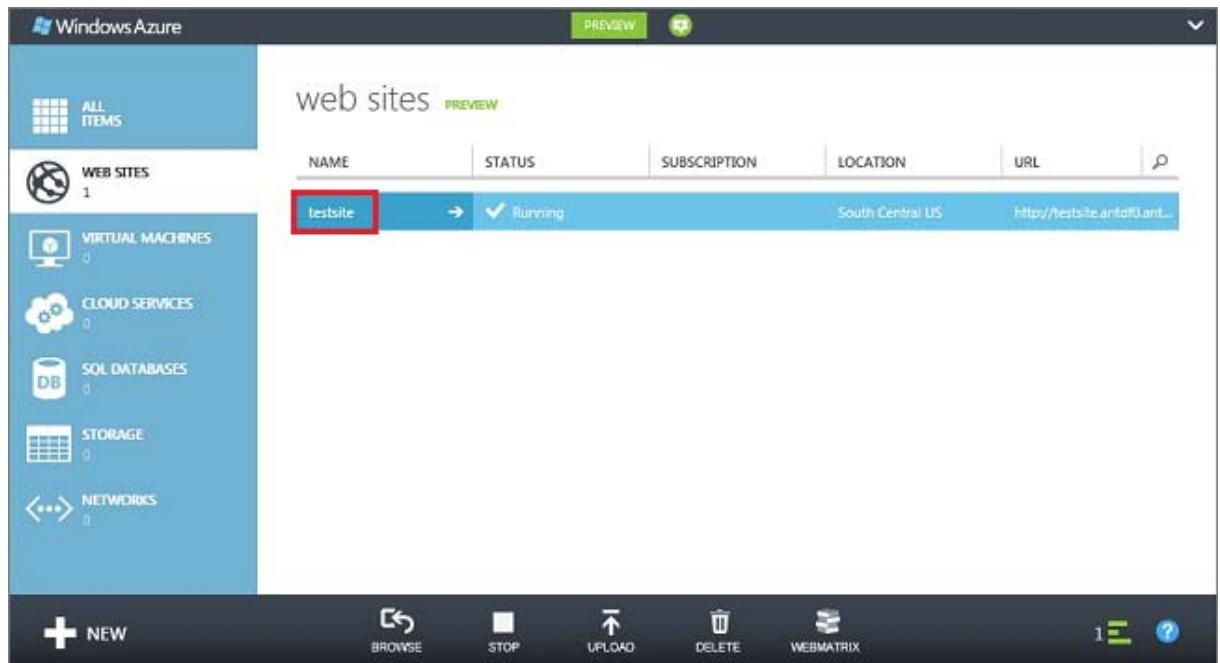
☒ Allow Windows Azure services to access the server ?

1 2

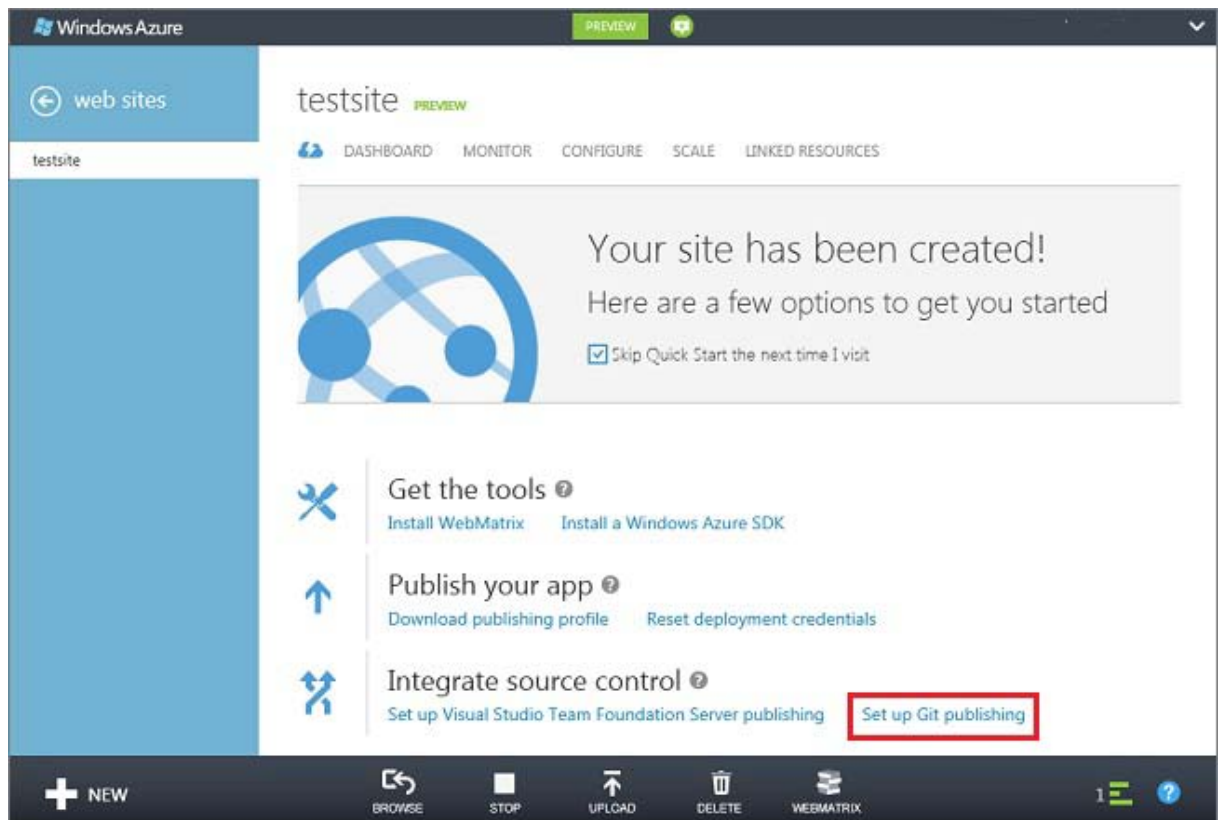
⏪ ⏩

When the web site has been created you will see the text **Creation of Web Site '[SITENAME]' completed successfully**. Now, you can enable Git publishing.

6. Click the name of the web site displayed in the list of web sites to open the web site's Quick Start dashboard.



7. At the bottom of the Quick Start page, click **Set up Git publishing**.



8. To enable Git publishing, you must provide a user name and password. Make a note of the user name and password you create. (If you have set up a Git repository before, this step will be skipped.)

×

DEPLOYMENT CREDENTIALS

Enter Username and Password



Deployment using GIT or FTP requires authentication using a username and password.

You only need to set this up once.

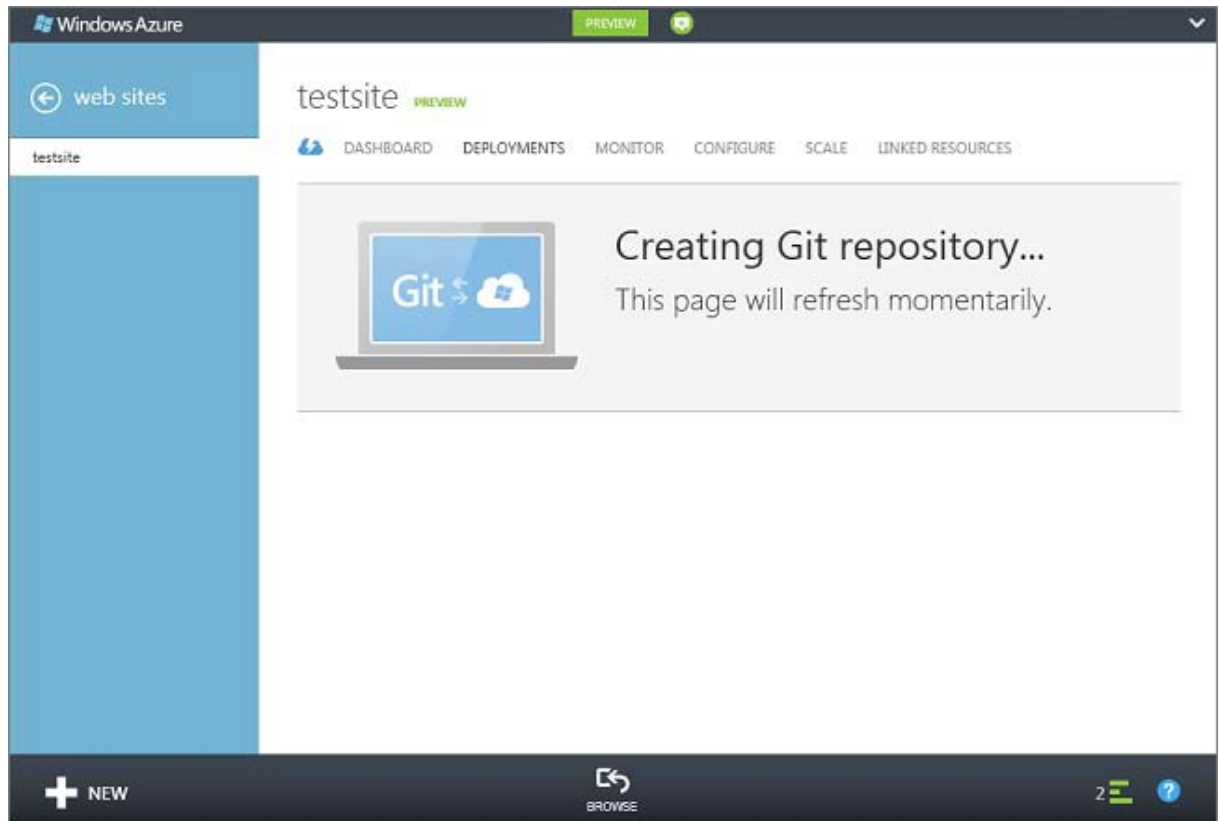
USER NAME

ENTER PASSWORD

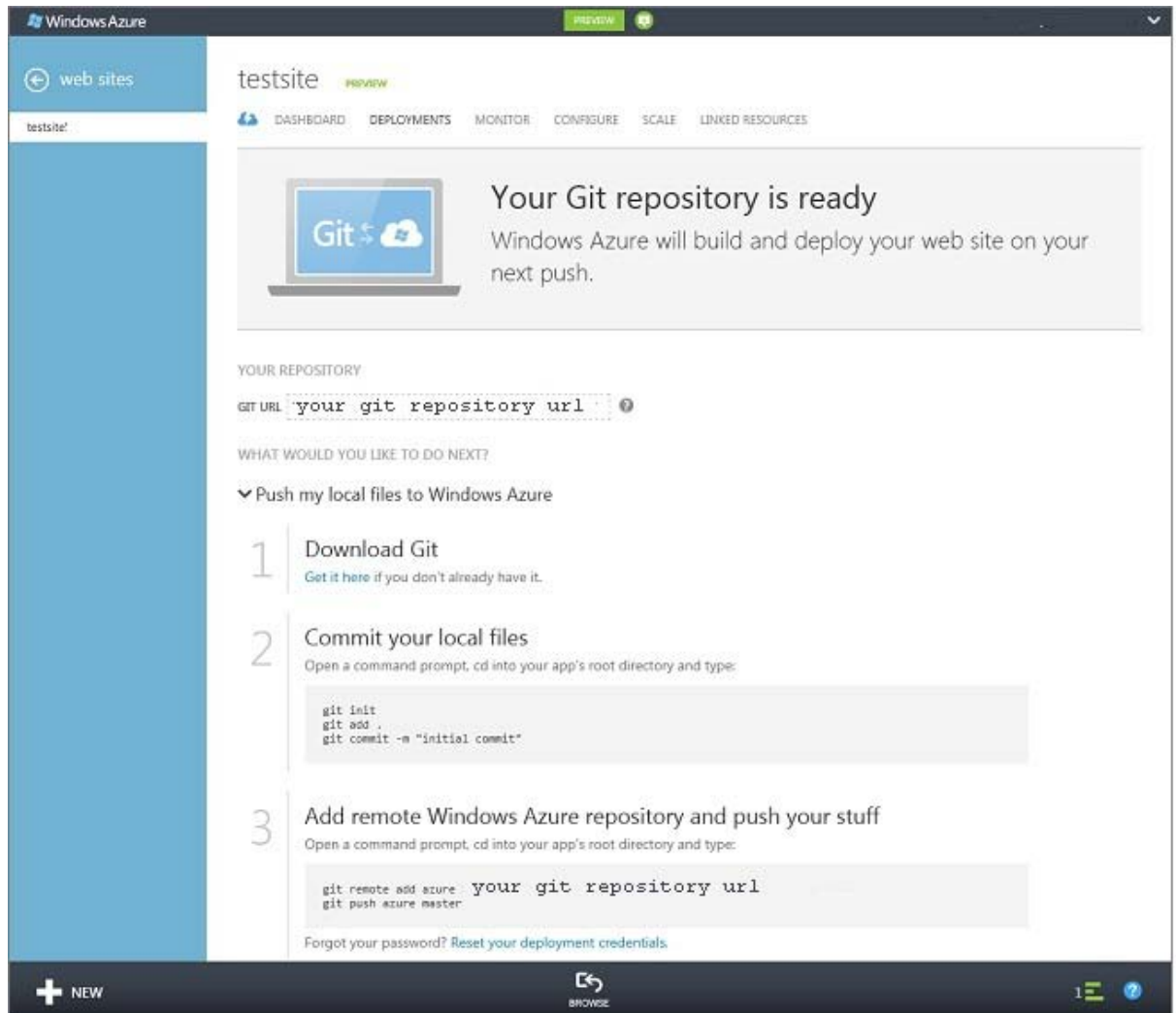
CONFIRM PASSWORD

It will take a few seconds to set up your repository.



9. When your repository is ready, you will see instructions for pushing your application files to the repository. Make note of these instructions - they will be needed later.



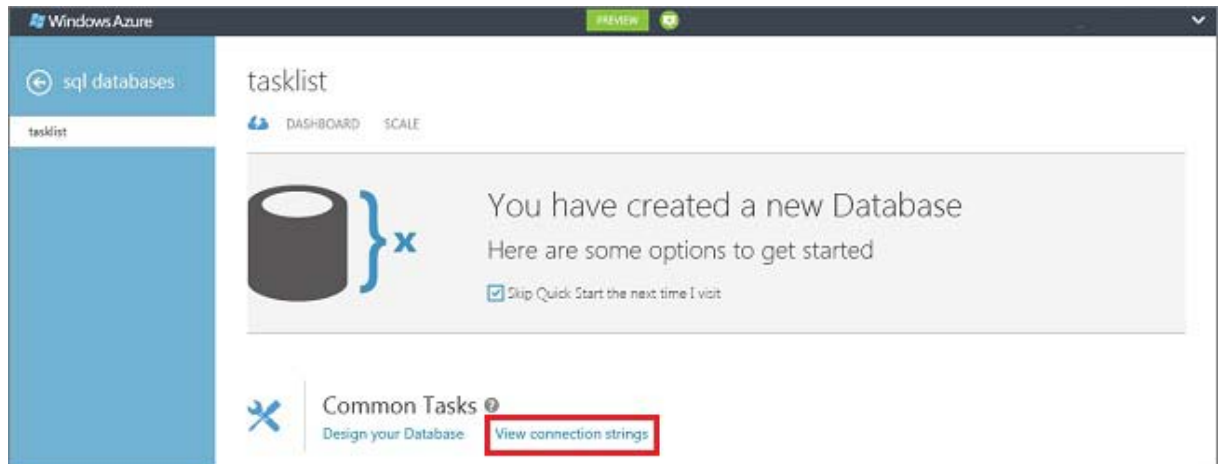
Get SQL Database connection information

To connect to the SQL Database instance that is running in Windows Azure Web Sites, you will need the connection information. To get SQL Database connection information, follow these steps:

1. From the Preview Management Portal, click **LINKED RESOURCES**, then click the database name.



2. Click View connection strings.

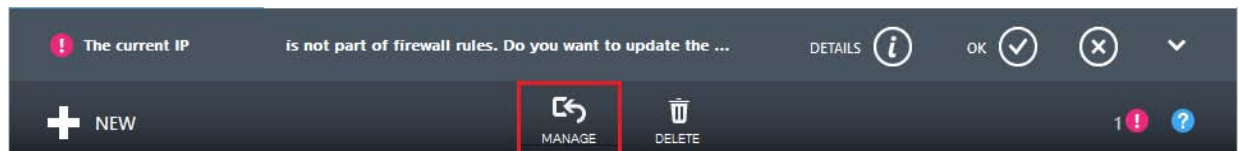


3. From the **ODBC** section of the resulting dialog, make note of the connection string as this will be used later.

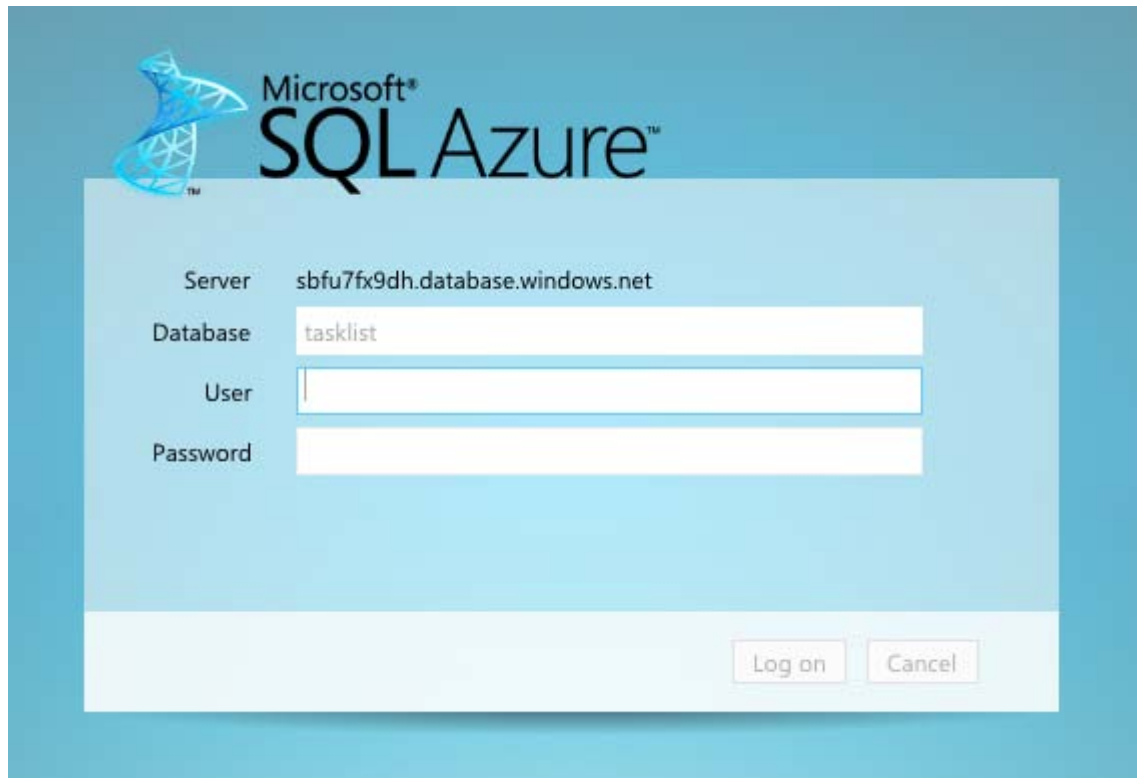
Design the task table

To create the database table used to store items for the tasklist application, perform the following steps:

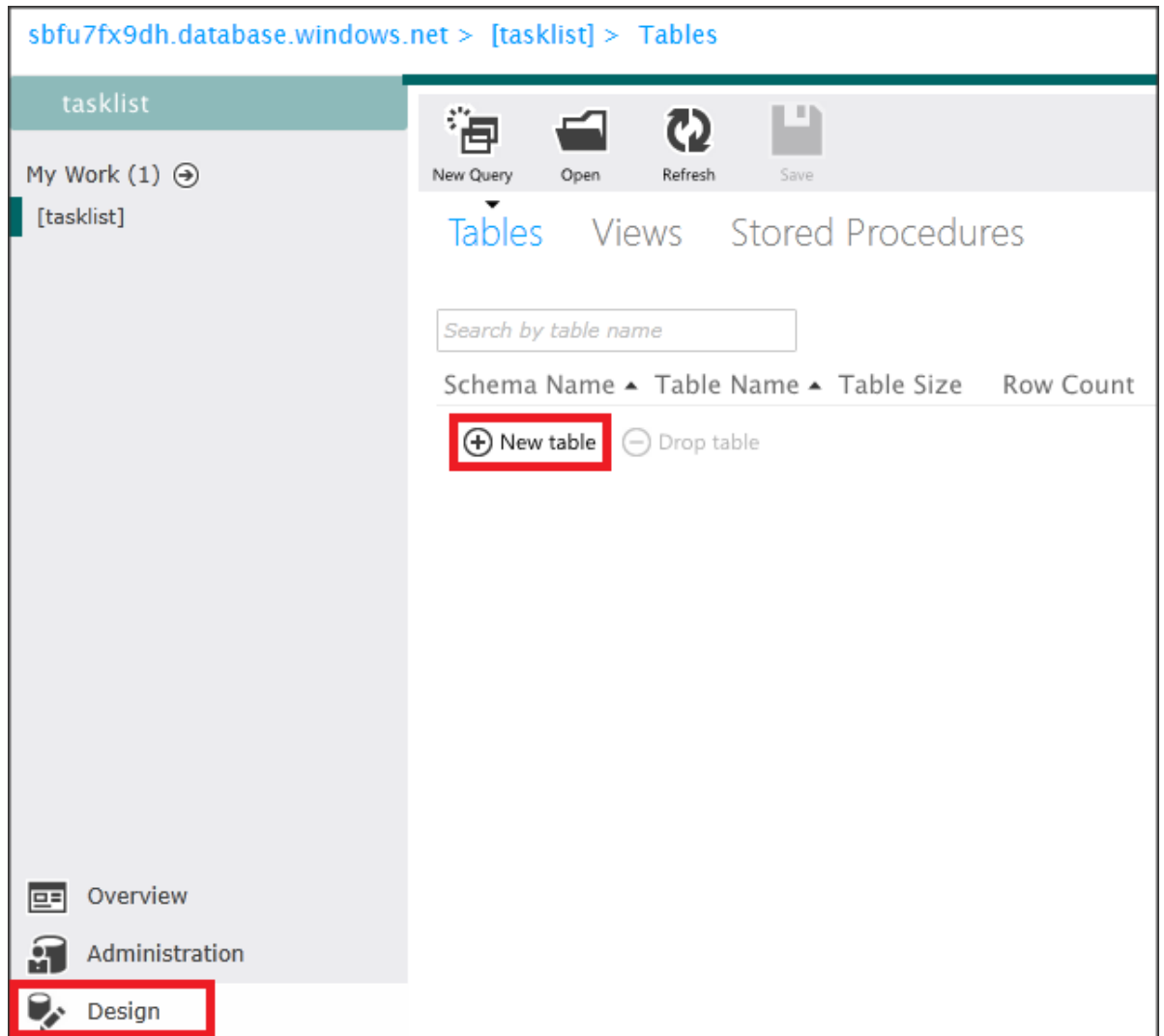
1. From the Preview Management Portal, select your SQL Database and then click **MANAGE** from the bottom of the page. If you receive a message stating that the current IP is not part of the firewall rules, select **OK** to add the IP address.



2. Login using the login name and password you selected when creating the database server earlier.



3. From the bottom left of the page, select **Design** and then select **New Table**.



4. Enter 'tasks' as the **Table Name** and check **Is Identity?** for the **ID** column.

sbfu7fx9dh.database.windows.net > [tasklist] > Tables > [dbo].[Table1]* User: larryfr Log off

tasklist

My Work (2) [dbo].[Table1] [tasklist]

New Query Open Refresh Save

Columns Indexes And Keys Data

Schema: dbo Table Name: **tasks**

Column	Select type	Default Value	Is Identity?	Is Required?	In Primary Key?
ID	int		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Column1	nvarchar	50	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Column2	nvarchar	15	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

+ Add column - Delete column

- Change **Column1** to **name** and **Column2** to **category**. Add two new columns by clicking the **Add column** button. The first new column should be named **created** and have a type of **date**. The second new column should be named **completed** and have a type of **bit**. Both new columns should be marked as **Is Required?**.

Column	Select type	Default Value	Is Identity?	Is Required?	In Primary Key?
ID	int		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
name	nvarchar	50	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
category	nvarchar	15	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
created	date		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
completed	bit		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

+ Add column - Delete column

- Click the **Save** button to save your changes to the table. You can now close the SQL Database management page.

Install modules and generate scaffolding

In this section you will create a new Node application and use npm to add module packages. For the task-list application you will use the [express](#) and [node-sqlserver](#) modules. The Express module provides a Model View Controller framework for node, while the node-sqlserver module provides connectivity to Windows Azure SQL Database.

Install express and generate scaffolding

- From the command-line, change directories to the **tasklist** directory. If the **tasklist** directory does not exist, create it.

2. Enter the following command to install express.

```
npm install express -g
```

Note When using the '-g' parameter on some operating systems, you may receive an error of error: eperm, chmod '/usr/local/bin/express' and a request to try running the account as an administrator. if this occurs, use the sudo command to run npm at a higher privilege level.

The output of this command should appear similar to the following:

```
express@2.5.9 /usr/local/lib/node_modules/express
├─ mime@1.2.4
├─ mkdirp@0.3.0
├─ qs@0.4.2
└─ connect@1.8.7
```

Note The '-g' parameter used when installing the express module installs it globally. This is done so that we can access the express command to generate web site scaffolding without having to type in additional path information.

3. To create the scaffolding which will be used for this application, use the **express** command:

```
express
```

The output of this command should appear similar to the following:

```
create : .
create : ./package.json
create : ./app.js
create : ./public
create : ./public/javascripts
create : ./public/images
create : ./public/stylesheets
create : ./public/stylesheets/style.css
create : ./routes
create : ./routes/index.js
create : ./views
create : ./views/layout.jade
create : ./views/index.jade
```

don't forget to install dependencies:

```
$ cd . && npm install
```

After this command completes, you should have several new directories and files in the **tasklist** directory.

Install additional modules

1. From the command-line, change directories to the **tasklist** folder and enter the following to install the modules described in the **package.json** file:

```
npm install
```

The output of this command should appear similar to the following:

```
express@2.5.8 ./node_modules/express
```

```
├─ mime@1.2.4
```

```
├─ qs@0.4.2
```

```
├─ mkdirp@0.3.0
```

```
└─ connect@1.8.7
```

```
jade@0.26.0 ./node_modules/jade
```

```
├─ commander@0.5.2
```

```
└─ mkdirp@0.3.0
```

This installs all of the default modules that Express needs.

2. Next, use the following command to add the **nconf** module. This module will be used by the application to read the database connection string from a configuration file.

```
npm install nconf --save
```

3. Next, download the binary version of the Microsoft Driver for Node.JS for SQL Server from the [download center](#).
4. Extract the archive to the **tasklist\node_modules** directory.
5. Run the **node-sqlserver-install.cmd** file in the **tasklist\node_modules** directory. This will create a **node-sqlserver** subdirectory under **node_modules** and move the driver files into this new directory structure.
6. Delete the **node-sqlserver-install.cmd** file, as it is no longer needed.

Use SQL Database in a node application

In this section you will extend the basic application created by the **express** command modifying the existing **app.js** and create a new **index.js** files to use the database created earlier.

Modify the controller

1. In the **tasklist/routes** directory, open the **index.js** file in a text editor.
2. Replace the existing code in the **index.js** file with the following code. This loads the node-sqlserver, and nconf modules, then uses nconf to load the connection string from either an environment variable named **SQL_CONN** or an **SQL_CONN** value in the **config.json** file.

```
var sql = require('node-sqlserver')
    , nconf = require('nconf');
```

```
nconf.env()
    .file({ file: 'config.json' });
var conn = nconf.get("SQL_CONN");
```

3. Continue adding to the **index.js** file by adding the **index** and **updateItem** methods. The **index** method returns all uncompleted tasks from the database, while **updateItem** will mark selected tasks as completed.

```
exports.index = function(req, res) {
    var select = "select * from tasks where completed = 0";
    sql.query(conn, select, function(err, items) {
        if(err)
            throw err;
        res.render('index', { title: 'My ToDo List ',
tasks: items });
    });
};
```

```
exports.updateItem = function(req, res) {
    var item = req.body.item;
    if(item) {
```

```

        var insert = "insert into tasks (name, category,
created, completed) values (?, ?, GETDATE(), 0)";
        sql.query(conn, insert, [item.name, item.category],
function(err) {
            if(err)
                throw err;
            res.redirect('home');
        });
    } else {
        var completed = req.body.completed;
        if(!completed.forEach)
            completed = [completed];
        var update = "update tasks set completed = 1 where
id in (" + completed.join(",") + ")";
        sql.query(conn, update, function(err) {
            if(err)
                throw err;
            res.redirect('home');
        });
    }
}
}

```

4. Save the **index.js** file.

Modify app.js

1. In the **tasklist** directory, open the **app.js** file in a text editor. This file was created earlier by running the **express** command.
2. Replace the content after the `//Routes` comment with the following code. This will add a new route to the **updateItem** method you added previously in the **index.js** file and listen on the port specified in `process.env.PORT`. The port value will be used once the application is deployed to Windows Azure.

```
// Routes
```

```

app.get('/', routes.index);
app.post('/', routes.updateItem);

```


- ```
app.listen(process.env.PORT || 3000);
```
3. Save the **app.js** file.

## Modify the index view

1. Change directories to the **views** directory and open the **index.jade** file in a text editor.
2. Replace the contents of the **index.jade** file with the code below. This defines the view for displaying existing tasks, as well as a form for adding new tasks and marking existing ones as completed.

```
h1= title
br

form(action="/", method="post")
 table(class="table table-striped table-bordered")
 thead
 tr
 td Name
 td Category
 td Date
 td Complete
 tbody
 each task in tasks
 tr
 td #{task.name}
 td #{task.category}
 td #{task.created}
 td
 input(type="checkbox", name="completed",
value="#{task.ID}", checked=task.completed == 1)
 button(type="submit") Update tasks
hr
```

- ```

form(action="/", method="post", class="well")
  label Item Name:
  input(name="item[name]", type="textbox")
  label Item Category:
  input(name="item[category]", type="textbox")
  br
  button(type="submit", class="btn") Add Item

```
3. Save and close **index.jade** file.

Modify the global layout

The **layout.jade** file in the **views** directory is used as a global template for other **.jade** files. In this step you will modify it to use [Twitter Bootstrap](#), which is a toolkit that makes it easy to design a nice looking web site.

1. Download and extract the files for [Twitter Bootstrap](#). Copy the **bootstrap.min.css** file from the **bootstrap\css** folder to the **public\stylesheets** directory of your tasklist application.
2. From the **views** folder, open the **layout.jade** in your text editor and replace the contents with the following:

```

!!!html
html
  head
    title= title
    meta(http-equiv='X-UA-Compatible', content='IE=10')
    link(rel='stylesheet',
href='/stylesheets/style.css')
    link(rel='stylesheet',
href='/stylesheets/bootstrap.min.css')
  body(class='app')
    div(class='navbar navbar-fixed-top')
      .navbar-inner
        .container
          a(class='brand', href='/') My Tasks
    .container!= body

```

3. Save the **layout.jade** file.

Create configuration file

The **config.json** file contains the connection string used to connect to the SQL Database, and is read by the **index.js** file at run-time. To create this file, perform the following steps:

1. In the **tasklist** directory, create a new file named **config.json** and open it in a text editor.
2. The contents of the **config.json** file should appear similar to the following:

```
{  
  "SQL_CONN" : "connection_string"  
}
```

Replace the **connection_string** with the ODBC connection string value returned earlier.

3. Save the file.

Run your application locally

To test the application on your local machine, perform the following steps:

1. From the command-line, change directories to the **tasklist** directory.
2. Use the following command to launch the application locally:
`node app.js`
3. Open a web browser and navigate to `http://127.0.0.1:3000`. This should display a web page similar to the following:

My Tasks

My ToDo List

Name	Category	Date	Complete
------	----------	------	----------

[Update tasks](#)

Item Name:

Item Category:

[Add Item](#)

4. Use the provided fields for **Item Name** and **Item Category** to enter information, and then click **Add item**.
5. The page should update to display the item in the ToDo List.

My Tasks

My ToDo List

Name	Category	Date	Complete
A thing	An item	2012-06-15	<input type="checkbox"/>

[Update tasks](#)

Item Name:

Item Category:

[Add Item](#)

6. To complete a task, simply check the checkbox in the Complete column, and then click **Update tasks**.

7. To stop the node process, go to the command-line and press the **CTRL** and **C** keys.

Deploy your application to Windows Azure

In this section, you will use the deployment steps you received after creating the web site to publish your application to Windows Azure.

Publish the application

1. At the command-line, change directories to the **tasklist** directory if you are not already there.
2. Use the following commands to initialize a local git repository for your application, add the application files to it, and finally push the files to Windows Azure

```
git init
git add .
git commit -m "adding files"
git remote add azure [URL for remote repository]
git push azure master
```

At the end of the deployment, you should see a statement similar to the following:

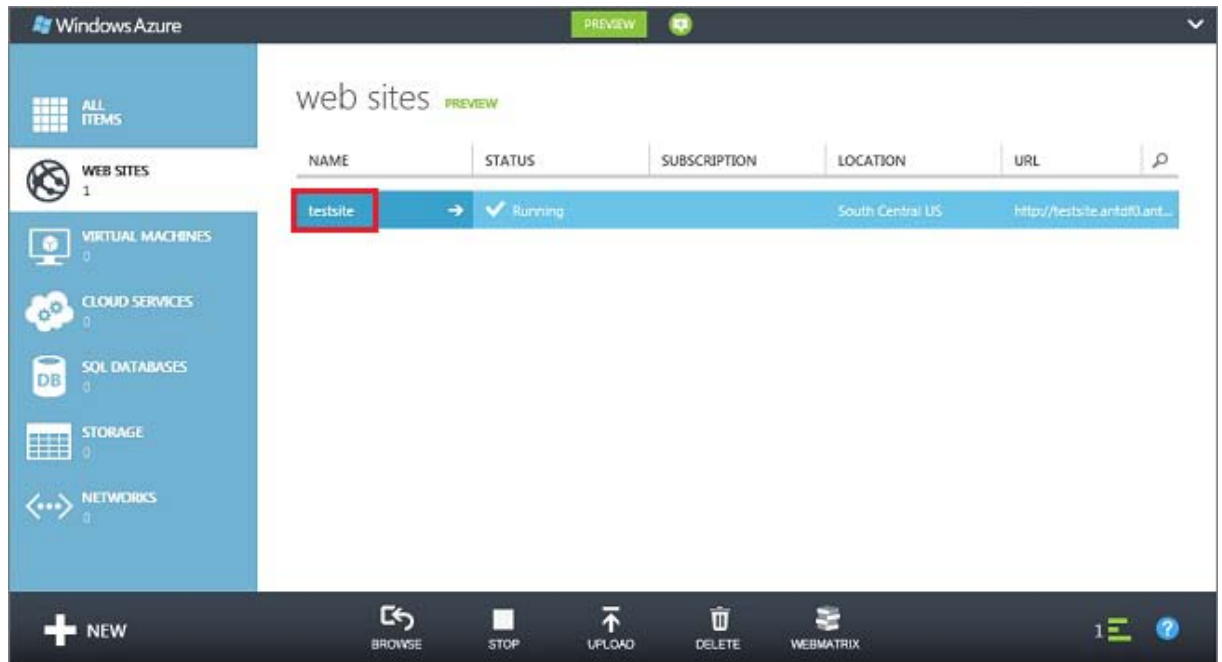
```
To
https://username@tabletasklist.azurewebsites.net/TableTaskl
ist.git
* [new branch]      master -> master
```

3. Once the push operation has completed, browse to **http://[site name].azurewebsites.net/** to view your application.

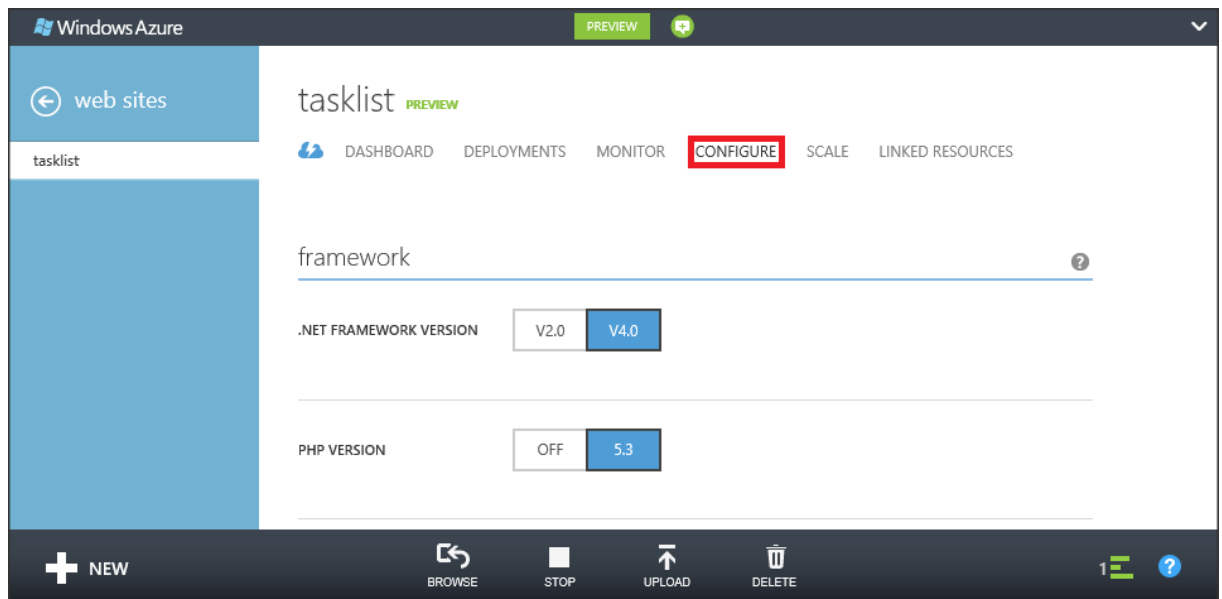
Switch to an environment variable

Earlier we implemented code that looks for a **SQL_CONN** environment variable for the connection string or loads the value from the **config.json** file. In the following steps you will create a key/value pair in your web site configuration that the application real access through an environment variable.

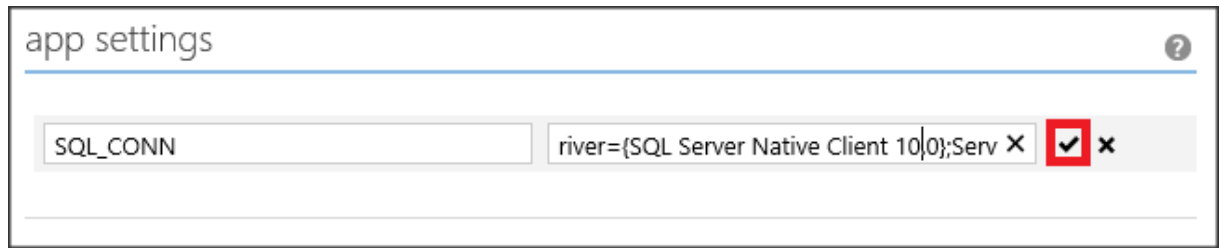
1. From the Preview Management Portal, click **Web Sites** and then select your web site.



2. Click **CONFIGURE** and then find the **app settings** section of the page.



3. In the **app settings** section, enter **SQL_CONN** in the **KEY** field, and the ODBC connection string in the **VALUE** field. Finally, click the checkmark.

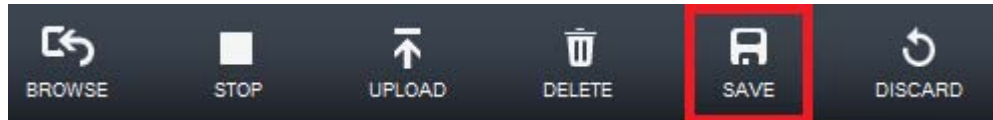


app settings

SQL_CONN

river={SQL Server Native Client 10|0};Serv X ✓ X

4. Finally, click the **SAVE** icon at the bottom of the page to commit this change to the run-time environment.



5. From the command-line, change directories to the **tasklist** directory and enter the following command to remove the **config.json** file:

```
git rm config.json
git commit -m "Removing config file"
```

6. Perform the following command to deploy the changes to Windows Azure:

```
git push azure master
```

Once the changes have been deployed to Windows Azure, your web application should continue to work as it is now reading the connection string from the **app settings** entry. To verify this, change the value for the **SQL_CONN** entry in **app settings** to an invalid value. Once you have saved this value, the web site should fail due to the invalid connection string.

Next steps

[Node.js Web Application with MongoDB](#)

[Node.js Web Application with Table Storage](#)

Additional resources

[Windows Azure command-line tool for Mac and Linux](#)