**Azure Web Apps**

**Overview**

[Azure App Service](https://azure.microsoft.com/en-us/documentation/articles/app-service-value-prop-what-is/) is a set of services provided by Microsoft Azure to enable developers to easily build and deploy Web apps and mobile apps for various platforms and devices. Included in the App Service family are [Azure Web Apps](https://azure.microsoft.com/en-us/documentation/articles/app-service-web-overview/), which allow you to quickly and easily deploy Web sites built with tools and languages you’re already familiar with; [Azure Mobile Apps](https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-value-prop/), which provide data services, syncing services, notification services, and other back-end services for popular mobile operating systems; [Azure API Apps](https://azure.microsoft.com/en-us/documentation/articles/app-service-api-apps-why-best-platform/), which simplify the writing, publishing, and consuming of cloud APIs; and [Azure Logic Apps](https://azure.microsoft.com/en-us/documentation/articles/app-service-logic-what-are-logic-apps/), which are great for automating business processes.

Azure Web Apps makes deploying Web sites extraordinarily easy – and not just Web sites built using the Microsoft stack. You can deploy PHP apps that use MySQL just as easily as ASP.NET apps that use SQL Server. You can select from a wide variety of Web App templates or build templates of your own. You can configure Web Apps to auto-scale as traffic increases to ensure that your customers aren’t left waiting during periods of peak demand. You can publish apps to staging locations and test them in the cloud before taking them live, and then swap staging deployments for production deployments with the click of a button. You can even create WebJobs – programs or scripts that run continuously or on a schedule to handle billing and other time-critical tasks. In short, Azure Web Apps takes the pain out of publishing and maintaining Web apps and are just as suitable for a personal photo-sharing site as they are for enterprise-grade sites serving millions of customers.

In this lab, you will use the cross-platform [Visual Studio Code](https://code.visualstudio.com/) editor to build a Web site that uses [PHP](http://php.net/) server-side scripting. The site will allow you to upload, browse, and display photos, and it will store photos in a [MySQL](http://www.mysql.com/) database. You will then provision a new Azure Web App to host the site. Finally, you will upload the site's content to the newly provisioned Web App and view it in your browser.

**Objectives**

In this hands-on lab, you will learn how to:

* Use Visual Studio Code to build a PHP and MySQL Web site
* Provision an Azure Web App to host the Web site
* Deploy the Web site using FTP

**Prerequisites**

The following are required to complete this hands-on lab:

* An active Microsoft Azure subscription. If you don't have one, [sign up for a free trial](http://aka.ms/WATK-FreeTrial).
* [Visual Studio Code](https://code.visualstudio.com/download)
* [PHP for Windows](http://windows.php.net/download/)

**Exercises**

This hands-on lab includes the following exercises:

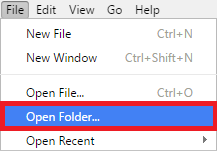
* [Exercise 1: Build a Web site with Visual Studio Code](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Azure%20Web%20Apps%20HOL.md#Exercise1)
* [Exercise 2: Provision a MySQL Web App](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Azure%20Web%20Apps%20HOL.md#Exercise2)
* [Exercise 3: Deploy the Web site](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Azure%20Web%20Apps%20HOL.md#Exercise3)

Estimated time to complete this lab: **45** minutes.

**Exercise 1: Build a Web site with Visual Studio Code**

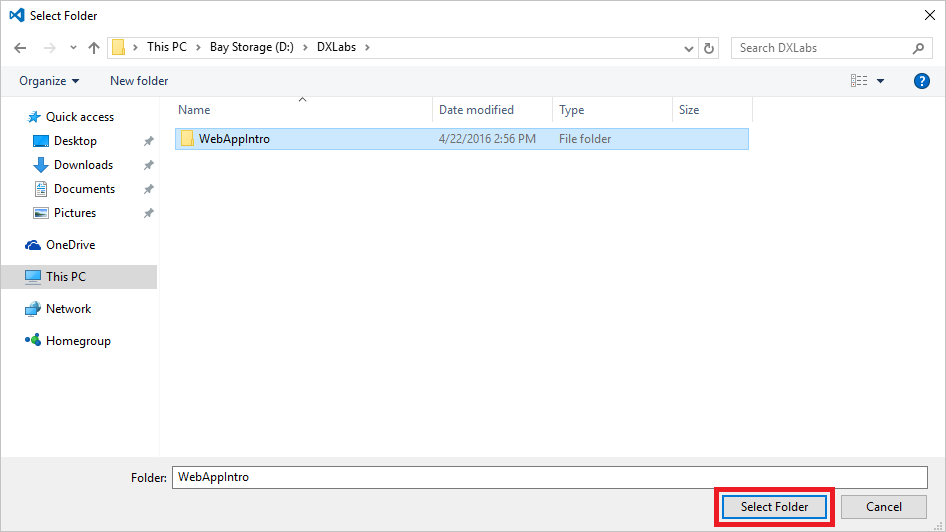
In this exercise, you will use [Visual Studio Code](https://code.visualstudio.com/) to build a Web site. Visual Studio Code is a free, cross-platform code editor available for Windows, macOS, and Linux that supports a variety of programming languages, both natively and via extensions. It features built-in Git support as well as syntax highlighting and code completion via [IntelliSense](https://code.visualstudio.com/#meet-intellisense).

1. Create a project directory named "WebAppIntro" in the location of your choice — for example, "C:\DXLabs\WebAppIntro."
2. Start Visual Studio Code. Open the **File** menu and select **Open Folder**.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/build-codeopenfolder.png)

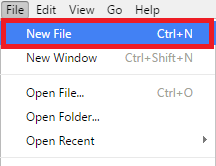
*Starting the project*

1. Select the project folder that you created in Step 1. Then click the **Select Folder** button.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/build-codeselectfolder.png)

*Selecting a project folder*

1. From the **File** menu, select **New File**.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/build-codenewfile.png)

*Adding a file to the project*

1. Add the following code and markup to the new file to serve as the main page for your Web site:

<!DOCTYPE html>

<html>

<head>

<title>My Images</title>

<!-- Latest compiled and minified CSS -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"

integrity="sha384-1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"

crossorigin="anonymous">

<!-- Optional theme -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css"

integrity="sha384-fLW2N01lMqjakBkx3l/M9EahuwpSfeNvV63J5ezn3uZzapT0u7EYsXMjQV+0En5r"

crossorigin="anonymous">

<link rel="stylesheet" href="content/styles.css" type='text/css'>

<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>

<!-- Latest compiled and minified JavaScript -->

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"

integrity="sha384-0mSbJDEHialfmuBBQP6A4Qrprq5OVfW37PRR3j5ELqxss1yVqOtnepnHVP9aJ7xS"

crossorigin="anonymous"></script>

</head>

<body>

<div class="container-fluid bg-primary">

<div class="row col-md-12">

<h3>My Images</h3>

</div>

</div>

<div class="navbar navbar-default">

<form class="navbar-form navbar-left" enctype="multipart/form-data" action="<?php echo $\_SERVER['PHP\_SELF']; ?>" method="post">

<div class="form-group">

<label class="sr-only" for="imageToUpload">Image to Upload</label>

<div class="input-group">

<span class="input-group-btn">

<span class="btn btn-default btn-file" type="button">

Image File:<input type="file" id="imageToUpload" name="imageToUpload">

</span>

</span>

<input type="text" class="form-control" placeholder="Select a file to upload..." id="selectedFileName" readonly>

</div>

</div>

<button type="submit" class="btn btn-default navbar-btn">Upload</button>

<?php

if (isset($\_FILES['imageToUpload'])) {

include "images.php";

try {

$msg = Images::Upload(); // this will upload the image

echo "<p class='navbar-text navbar-right'>".$msg."</p>"; // Message showing success or failure.

}

catch (Exception $e) {

echo "<p class='navbar-text navbar-right text-danger'>"."Sorry, could not upload file".$e->getMessage()."</p>";

}

}

?>

</form>

</div>

<div class="container-fluid">

<div class="row">

<?php

include "images.php";

$images = Images::GetImages();

foreach ($images as $image) {

?>

<div class='col-lg-2 col-md-4 col-sm-6 col-xs-12'>

<?php

echo "<a href='image\_display.php?id=".$image->id."' target='\_blank'>";

echo "<img class='img-responsive' src='image\_display.php?id=".$image->id."&width=192' alt='' />";

echo "</a>";

?>

</div>

<?php

}

?>

</div>

</div>

<script type="text/javascript" language="javascript">

// Show name of selected image file in the text display in the custom UI element

$(document).ready(function () {

$(document).on('change', '.btn-file :file', function () {

var input = $(this),

numFiles = input.get(0).files ? input.get(0).files.length : 1,

label = input.val().replace(/**\\**/g, '/').replace(/.\***\/**/, '');

input.trigger('fileselect', [numFiles, label]);

})

$('.btn-file :file').on('fileselect', function (event, numFiles, label) {

console.log(numFiles);

console.log(label);

$("#selectedFileName").val(label);

});

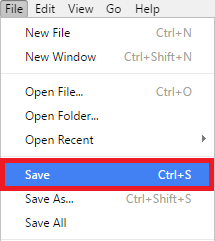
});

</script>

</body>

</html>

1. Use the **File -> Save** command to save the file. Name it **index.php**.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/build-codesavefile.png)

*Saving the file*

1. Repeat Steps 3 through 5 to add a file named **image\_display.php** containing the following code to the project. This is the code that returns images requested by the browser over HTTP.

<?php

if ((isset($\_GET['id']) && is\_numeric($\_GET['id'])) === FALSE) die;

include "images.php";

$imageId = $\_GET['id'];

$image = Images::GetImage($imageId);

// get the source image attributes

$srcImage = $image->image;

$srcSize = getImageSizeFromString($srcImage);

$srcWidth = $srcSize[0];

$srcHeight = $srcSize[1];

$srcType = $srcSize[2];

$srcMime = $srcSize['mime'];

$srcImageResource = imageCreateFromString($srcImage);

// set the header for the image

header("Content-type: ".$srcMime);

if ((isset($\_GET['width']) && is\_numeric($\_GET['width'])) === FALSE) {

// no width requested - just return the source

echo $srcImage;

exit;

}

// resize/resample the image to the requested size

$destWidth = $\_GET['width'];

$destHeight = $destWidth \* $srcSize[1] / $srcSize[0];

$destImageResource = imageCreateTrueColor($destWidth, $destHeight);

imagealphablending($destImageResource, false);

imagesavealpha($destImageResource, true);

imageCopyResampled($destImageResource, $srcImageResource, 0,0,0,0, $destWidth, $destHeight, $srcWidth, $srcHeight);

// export the image

switch ($srcType) {

case IMAGETYPE\_JPEG:

imageJPEG($destImageResource);

break;

case IMAGETYPE\_PNG:

imagePNG($destImageResource);

break;

default:

imageJPEG($destImageResource);

break;

}

imageDestroy($destImageResource);

?>

1. Repeat Steps 3 through 5 to add a file named **images.php** containing the following code to the project. This is the code used to upload, store, and retrieve images.

<?php

include "database.php";

class Images {

public static function Upload() {

$maxsize = 10000000; // set to approx 10 MB

// check associated error code

if ($\_FILES['imageToUpload']['error'] == UPLOAD\_ERR\_OK) {

// check whether file is uploaded with HTTP POST

if (is\_uploaded\_file($\_FILES['imageToUpload']['tmp\_name'])) {

// check size of uploaded image on server side

if ( $\_FILES['imageToUpload']['size'] < $maxsize) {

// check whether uploaded file is of image type

$finfo = finfo\_open(FILEINFO\_MIME\_TYPE);

if (strpos(finfo\_file($finfo, $\_FILES['imageToUpload']['tmp\_name']), "image") === 0) {

// open the image file for insertion

$imagefp = fopen($\_FILES['imageToUpload']['tmp\_name'], 'rb');

// put the image in the db...

$database = new Database();

$id = $database->UploadImage($\_FILES['imageToUpload']['name'], $imagefp);

$msg = 'Image successfully saved in database with id = ' . $id;

}

else {

$msg = "Uploaded file is not an image.";

}

}

else {

// if the file is not less than the maximum allowed, print an error

$msg = '<div>File exceeds the Maximum File limit</div>

<div>Maximum File limit is '.$maxsize.' bytes</div>

<div>File '.$\_FILES['imageToUpload']['name'].' is '.$\_FILES['imageToUpload']['size'].

' bytes</div><hr />';

}

}

else

$msg = "File not uploaded successfully.";

}

else {

$msg = file\_upload\_error\_message($\_FILES['imageToUpload']['error']);

}

return $msg;

}

public static function GetImages() {

$database = new Database();

$images = $database->GetAllImages();

return $images;

}

public static function GetImage($id) {

$database = new Database();

$image = $database->FindImage($id);

return $image;

}

}

?>

1. Repeat Steps 3 through 5 to add a file named **database.php** containing the following code to the project. This is the code used to interact with the MySQL database. Observe that the database connection string isn't embedded in the code, but is instead retrieved from an environment variable named *MYSQLCONNSTR\_defaultConnection*. The value of this approach will be explained in the next exercise.

<?php

class Database {

private $link;

public function \_\_construct() {

// Notice that private connection information is \*NOT\* part of the source

// and therefore does not end up in public repos, etc.

$connectionString = getenv("MYSQLCONNSTR\_defaultConnection");

$varsString = str\_replace(";","&", $connectionString);

parse\_str($varsString);

$host = $Data\_Source;

$user = $User\_Id;

$pass = $Password;

$db = $Database;

try{

$this->link = new PDO("mysql:host=".$host.";dbname=".$db, $user, $pass);

$this->link->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

}

catch (PDOException $e){

echo "Error: Unable to connect to MySQL: ". $e->getMessage();

die;

}

$this->InitializeImageTable();

}

public function \_\_destruct() {

$this->link = null;

}

public function UploadImage($imageName, $imageFP) {

$sql = $this->link->prepare("INSERT INTO images (name, image) VALUES (:name, :image);");

$sql->bindParam(":name", $imageName);

$sql->bindParam(":image", $imageFP, PDO::PARAM\_LOB);

$sql->execute();

return $this->link->lastInsertId();

}

public function GetAllImages() {

$sql = $this->link->prepare("SELECT \* FROM images;");

$sql->execute();

$results = $sql->fetchAll(PDO::FETCH\_OBJ);

return $results;

}

public function FindImage($id) {

$sql = $this->link->prepare("SELECT \* FROM images WHERE id = :id;");

$sql->bindParam(":id", $id, PDO::PARAM\_INT);

$sql->execute();

$result = $sql->fetch(PDO::FETCH\_OBJ);

return $result;

}

private function InitializeImageTable() {

// Check to see if the table needs to be created

$results = $this->link->query("SHOW TABLES LIKE 'images';");

if ($results == TRUE && $results->rowCount() > 0) {

return;

}

// create table

$sql = "CREATE TABLE images (

id INT(10) NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL DEFAULT '',

image LONGBLOB NOT NULL

);";

if ($this->link->query($sql) != TRUE) {

die("Error creating image table: " . $this->link->error);

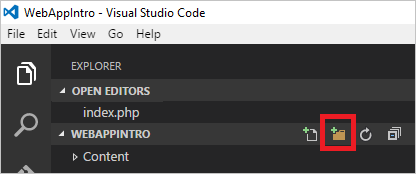
}

}

}

?>

1. Open the **View** menu and select **Explorer**. In the panel that appears, hover the mouse over the project folder and click the **New Folder** icon to create a new folder in the project folder. Name the new folder **Content**.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/build-codenewfolder.png)

*Adding a Content folder to the project*

1. Repeat Steps 3 through 5 to add a file named **styles.css** containing the following CSS to the **Content** folder that you just created:

/\* Custom background color for the nav-bar element. \*/

.navbar {

background-color: #d9edf7; /\*matches bg-info\*/

background-image: none;

background-repeat: no-repeat;

filter: none;

}

/\* Support for custom-looking File Selection button. \*/

.btn-file {

position: relative;

overflow: hidden;

}

.btn-file input[type=file] {

position: absolute;

top: 0;

right: 0;

min-width: 100%;

min-height: 100%;

font-size: 100px;

text-align: right;

filter: alpha(opacity=0);

opacity: 0;

outline: none;

background: white;

cursor: inherit;

display: block;

}

/\* Styling/framing for the image elements. \*/

img {

width: 192px;

background-color: white;

padding: 5px 5px 30px 5px;

cursor: pointer;

}

/\* Custom queries for force line breaks depending on the resolution. \*/

@media (min-width: 1200px) {

.col-lg-2:nth-child(6n+1) {

clear: both;

}

}

@media (min-width: 992pxpx) and (max-width: 1200px) {

.col-md-4:nth-child(3n+1) {

clear: both;

}

}

@media (min-width: 768px) and (max-width: 992px) {

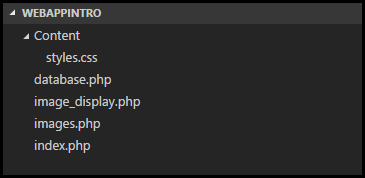
.col-sm-6:nth-child(2n+1) {

clear: both;

}

}

The files listed in the Explorer panel for your project should now look like this:

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/build-codefinalfolder.png)

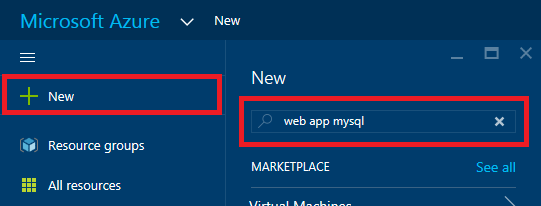
*Project content*

That's it! Your photo-sharing Web site is built. The next step is to provision an Azure Web App to host it.

**Exercise 2: Provision a MySQL Web App**

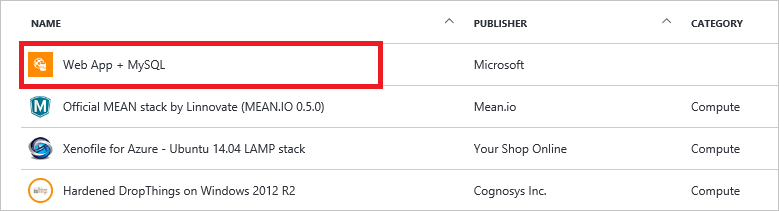
There are several ways to provision an Azure Web App. In this exercise, you will use the Azure Portal to do it. Provisioning is quick and easy and involves little more than answering a few questions and clicking a few buttons. Once the Web App is provisioned, you can view it in your browser just as you would any other Web site.

1. Open the [Azure Portal](https://portal.azure.com/) in your browser. If you are asked to log in, do so using your Microsoft account.
2. Click **+ New**. In the "New" blade that opens, type "web app mysql" (without quotation marks) into the search box and press **Enter**.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/find-webappmysql.png)

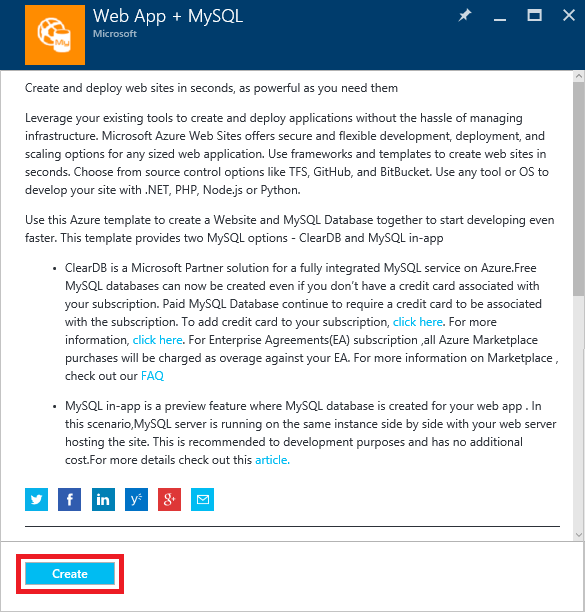
*Finding the "Web App + MySQL" template*

1. Two new blades named "Marketplace" and "Everything" open in the portal. The former represents the [Microsoft Azure Marketplace](https://azure.microsoft.com/en-us/marketplace/), which is an online store containing thousands of free templates for deploying apps, services, virtual machines, and more, preconfigured for Azure and provisioned with popular tools such as WordPress, CakePHP, and Django. In the "Everything" blade, click **Web App + MySQL**.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/select-webappmysql.png)

*Selecting the "Web App + MySQL" template*

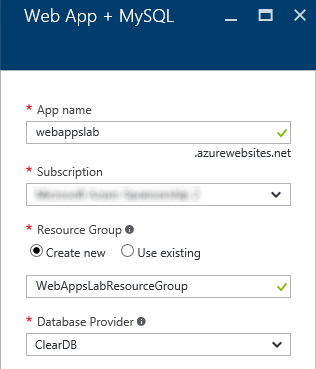
1. In the "Web App + MySQL" blade that opens, take a moment to review the text and learn what the template provisions. Then click the **Create** button at the bottom of the blade.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/create-web-app.png)

*Creating a Web App with MySQL*

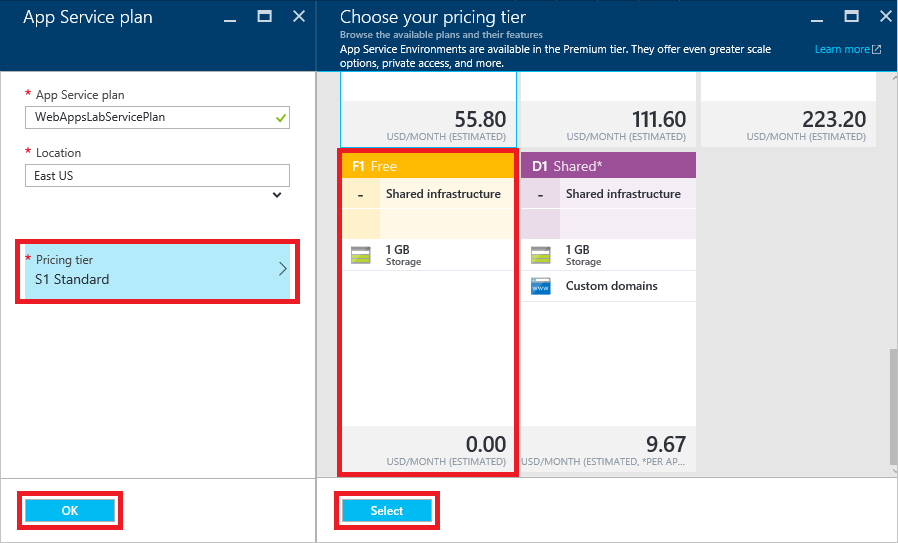
1. Enter a name for your Web app in the **App name** field. *This name must be unique within Azure, so make sure a green check mark appears next to it*. Select **Create new** under **Resource Group** and enter the resource-group name "WebAppsLabResourceGroup" (without quotation marks). Then select **ClearDB** as the **Database Provider**.

Resource groups are a powerful construct for grouping resources such as storage accounts, databases, and virtual machines together so they can be managed as a unit. Deleting a resource group deletes everything inside it and prevents you from having to delete those resources one by one.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/web-app-parameters-1.png)

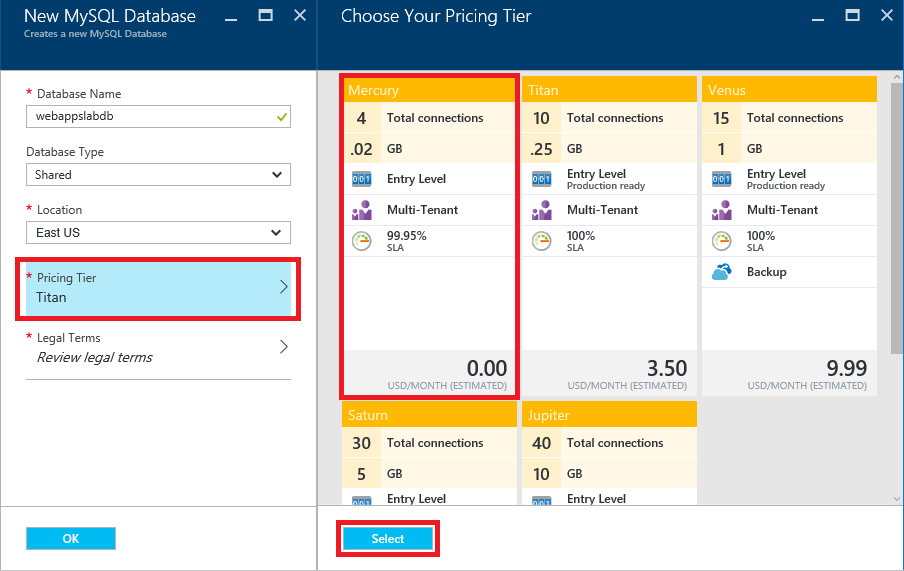
*Configuring the Web App*

1. Click **App Service plan/location**, and then click **Create New** to create a new App Service plan for your Web app. In the "App Service plan" blade, enter "WebAppsLabServicePlan" (without quotation marks) as the service-plan name and select the location nearest you. Then click **Pricing tier** and select the **F1 Free** tier. (You will have to click **View all** in the "Choose your pricing tier" blade and scroll to the bottom to see the **F1 Free** tier.) Click **Select** to finalize your tier selection, and then click **OK** in the "App Service plan" blade to create the new service plan.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/web-app-parameters-2.png)

*Creating an App Service plan*

1. Click **Database** in the "Web App + MySQL" blade, and then click **Create New** to create a new MySQL database to go with your Web app. Enter "webappslabdb" (without quotation marks) for the **Database Name**. Set **Database Type** to **Shared**and select the location nearest you. Then click **Pricing Tier**, select the **Mercury** tier, and click **Select** at the bottom of the blade.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/web-app-parameters-3.png)

*Creating a MySQL database*

1. Click **Legal Terms** in the "New MySQL Database" blade. Review the legal terms and click **Purchase** at the bottom of the "Purchase" blade. Finish up by clicking **OK** at the bottom of the "New MySQL Database" blade.

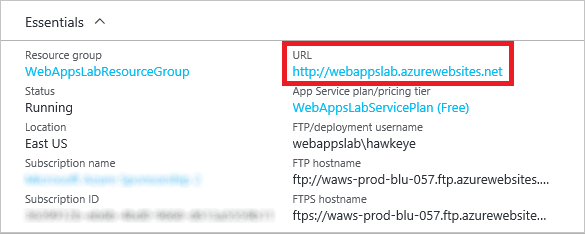
Nothing is actually being purchased since you selected the Mercury database tier, which is free. In order to provision the MySQL database for you, Azure requires that you acknowledge the terms of use.

1. Check **Pin to Dashboard** at the bottom of the "Web App + MySQL" blade, and then click the **Create** button to create your Web app.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/web-app-parameters-4.png)

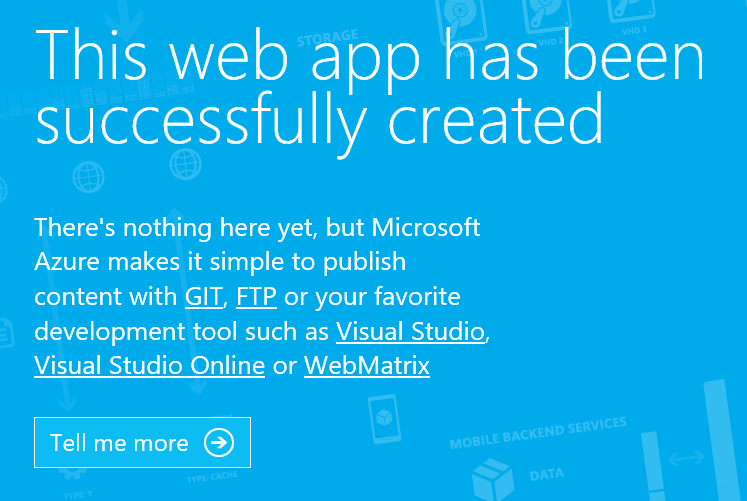
*Creating the Web App*

1. Once the Web App has been created (it usually takes about one minute), click the tile that was created for it on the dashboard.
2. Click the Web site URL to browse to the placeholder page for your new Web site.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/open-websiteplaceholder.png)

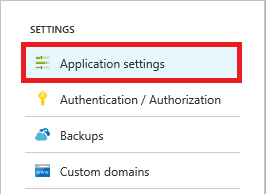
*The Web site URL*

1. Confirm that the placeholder page appears. Then close the browser window or tab in which it opened and return to the Azure Portal.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/placeholder-page.png)

*The Web site's temporary home page*

1. Click **Application settings** in the blade for the Web app.

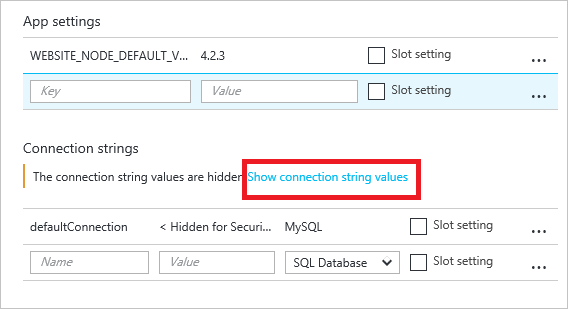
[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/application-settings.png)

*Viewing application settings*

1. Scroll down to the "App settings" and "Connection strings" sections. These sections allow you to define key-value pairs that the app can access at runtime. Specifying values such as these in the portal rather than embedding them in your code makes it easier to run the app in different environments and in different contexts, and also helps mitigate the risk of inadvertently uploading code containing database connection strings and other sensitive items to public source-code repositories.

If you're curious about the **Slot setting** boxes, here's a quick explanation. In the Azure App Service, the *Basic*,*Standard*, and *Premium* pricing tiers tiers allow you to provision multiple deployment slots for a given Web App. You can use these slots in several ways, including setting up pre-production staging environments for testing code changes before putting them into production. The **Slot setting** boxes allow you to specify whether a setting or connection string applies to all deployment slots or only to a particular deployment slot.

1. By default, connection strings are hidden for security reasons. Click **Show connection string values** to reveal them.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/provision-showconnectionstrings.png)

*Showing connection string values*

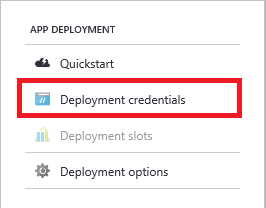
1. Examine the **defaultConnection** value. It was automatically created when your Web App and MySQL database were provisioned. This is the value your application uses to connect to the MySQL database. It includes the server address (*Data Source*) for the database, the database name, and the user credentials for connecting to the database.

Most platforms can access connection strings and app settings as environment variables. .NET applications can access them as if they were part of the application's web.config file. The PHP code that you added in [Exercise 1](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Azure%20Web%20Apps%20HOL.md#Exercise1) retrieves the default connection string by calling getenv("MYSQLCONNSTR\_defaultConnection"); You can find additional information about the conventions used to access these variables [here](https://azure.microsoft.com/en-us/blog/windows-azure-web-sites-how-application-strings-and-connection-strings-work/).

**Exercise 3: Deploy the Web site**

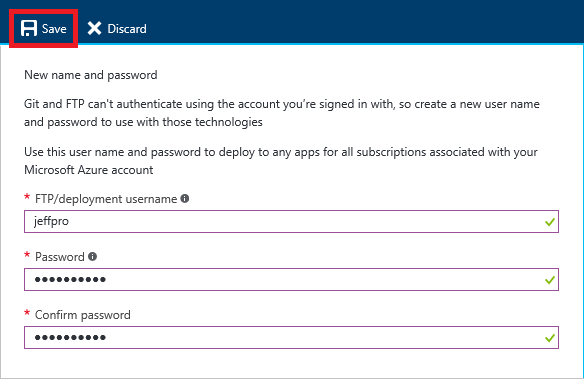
Now it is time to copy the files that comprise your Web site to the Azure Web App. In this exercise, you will publish your Web site using FTP.

1. Click **Deployment credentials** in the blade for the Web App.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/deploy-deploymentcredentials.png)

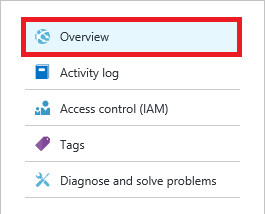
*Viewing deployment credentials*

1. Enter a user name and password for connecting to your site via FTP. **Be sure to remember the password**. Click the **Save**button at the top of the blade to save these credentials.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/deploy-enter-credentials.png)

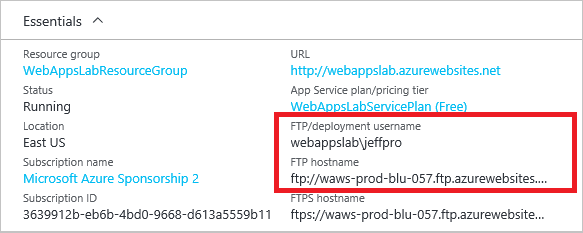
*Specifying FTP credentials*

1. Click **Overview** in the blade for the Web app.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/open-overview.png)

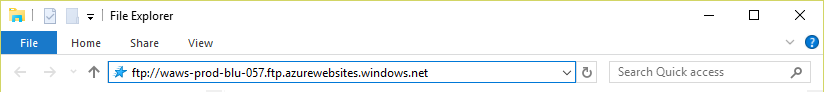
*Specifying FTP credentials*

1. Locate the **FTP/deployment username** and **FTP hostname** values. Hover the mouse cursor over the FTP hostname and click the **Copy** button that appears on the right to copy the hostname to the clipboard.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/deploy-getftpstrings.png)

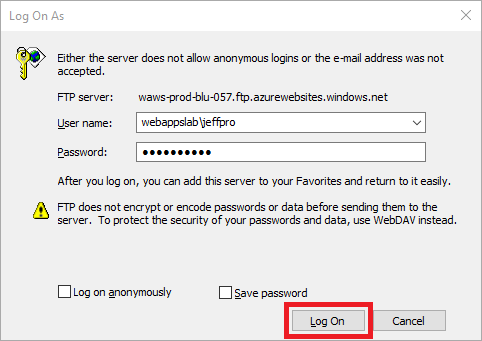
*The FTP username and hostname*

1. Open a File Explorer window and paste the FTP hostname value into the address box at the top of the window. Then press **Enter**.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/deploy-windowsenterftphostname.png)

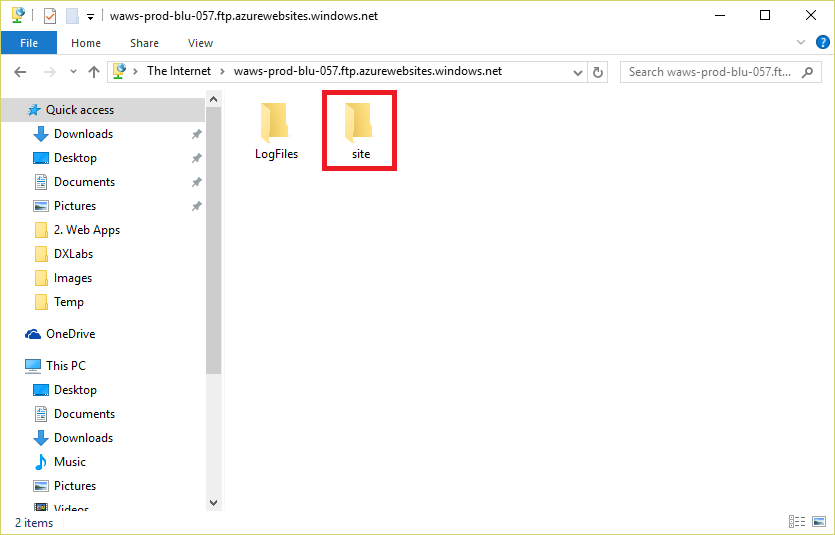
*Entering the FTP hostname in File Explorer*

1. When prompted to enter FTP credentials, enter the FTP username (the **FTP/deployment username** in Step 4) and the password you specified in Step 2. Then click **Log On**.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/deploy-windowscredentials.png)

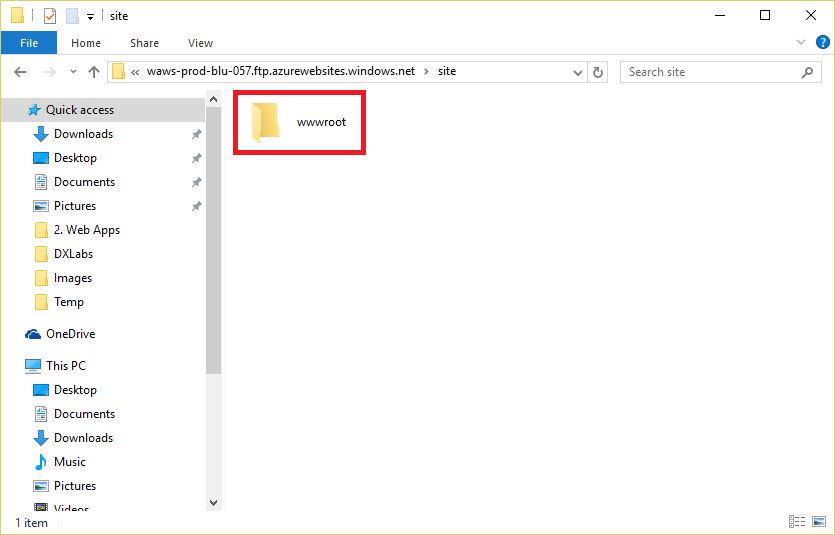
*Entering your FTP credentials*

1. Double-click the **site** folder in the File Explorer window.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/deploy-open-site-folder.png)

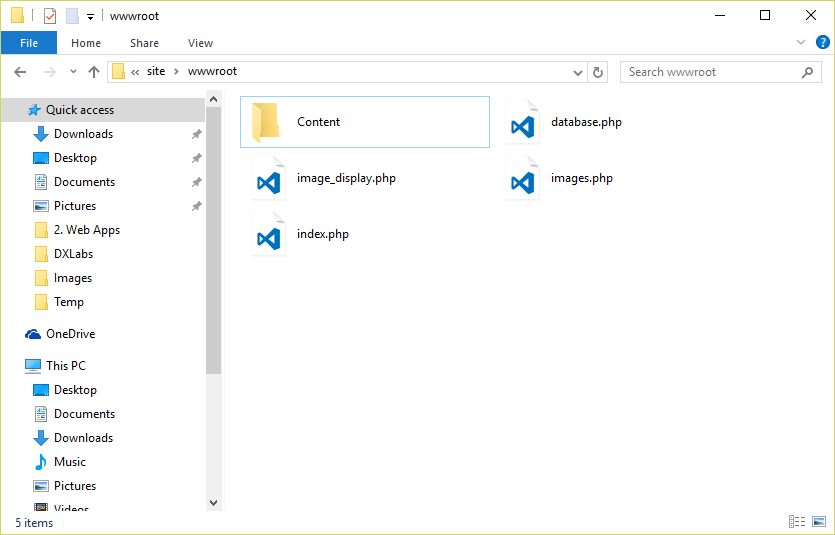
*Opening the site folder*

1. Double-click the **wwwroot** folder in the File Explorer window. This is the root folder for your Web site.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/deploy-open-wwwroot-folder.png)

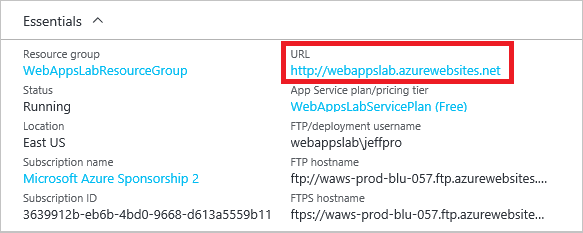
*Opening the wwwroot folder*

1. Delete the **hostingstart.html** file. This is the placeholder page you saw earlier when you connected to the site in your browser. When prompted to confirm the deletion, select **Yes**.
2. Open another File Explorer window and navigate to the folder containing the Web site files you created in [Exercise 1](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Azure%20Web%20Apps%20HOL.md#Exercise1).
3. Copy all of the files and folders that comprise the Web site to your site's **wwwroot** folder.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/deploy-windowsftpcopy.png)

*Copying the Web site files*

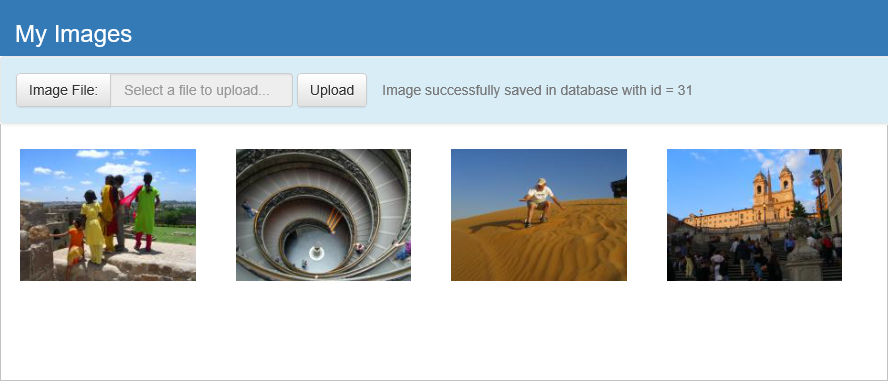
1. Congratulations! You now have a working Web site. Click the site URL in the Azure Portal to open the site in your browser.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/deploy-go-to-web-site.png)

*Navigating to the finished Web site*

1. Upload a few images to the Web site. Click any of the image thumbnails to see an enlarged view.

By default, the PHP installation in your Azure Web App limits file uploads to 8 MB. You can increase the limit by adding a .user.ini file to your site as [described here](https://azure.microsoft.com/en-us/documentation/articles/web-sites-php-configure/). For now, just upload images that are smaller than 8 MB because you would have to adjust other PHP settings such as MEMORY\_LIMIT to handle very large images.

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/2.%20Web%20Apps/Images/final-workingwebsite.png)

*The working Web site*

**Summary**

In this hands-on lab, you learned how to:

* Use Visual Studio Code to build a Web site that uses PHP and MySQL
* Provision an Azure Web App
* Upload a Web site to Azure

Azure Web Apps allows you to use the tools and languages you already know to build great Web sites and publish them to the Web. You can focus on what you know and what you want to accomplish rather than spend time maintaining physical infrastructure, installing security updates, and learning new languages. Moreover, Azure Web Apps can easily leverage other features of the Azure ecosystem such as Azure Storage, Azure Search, and Azure Cognitive Services to achieve unprecedented scale and richness.