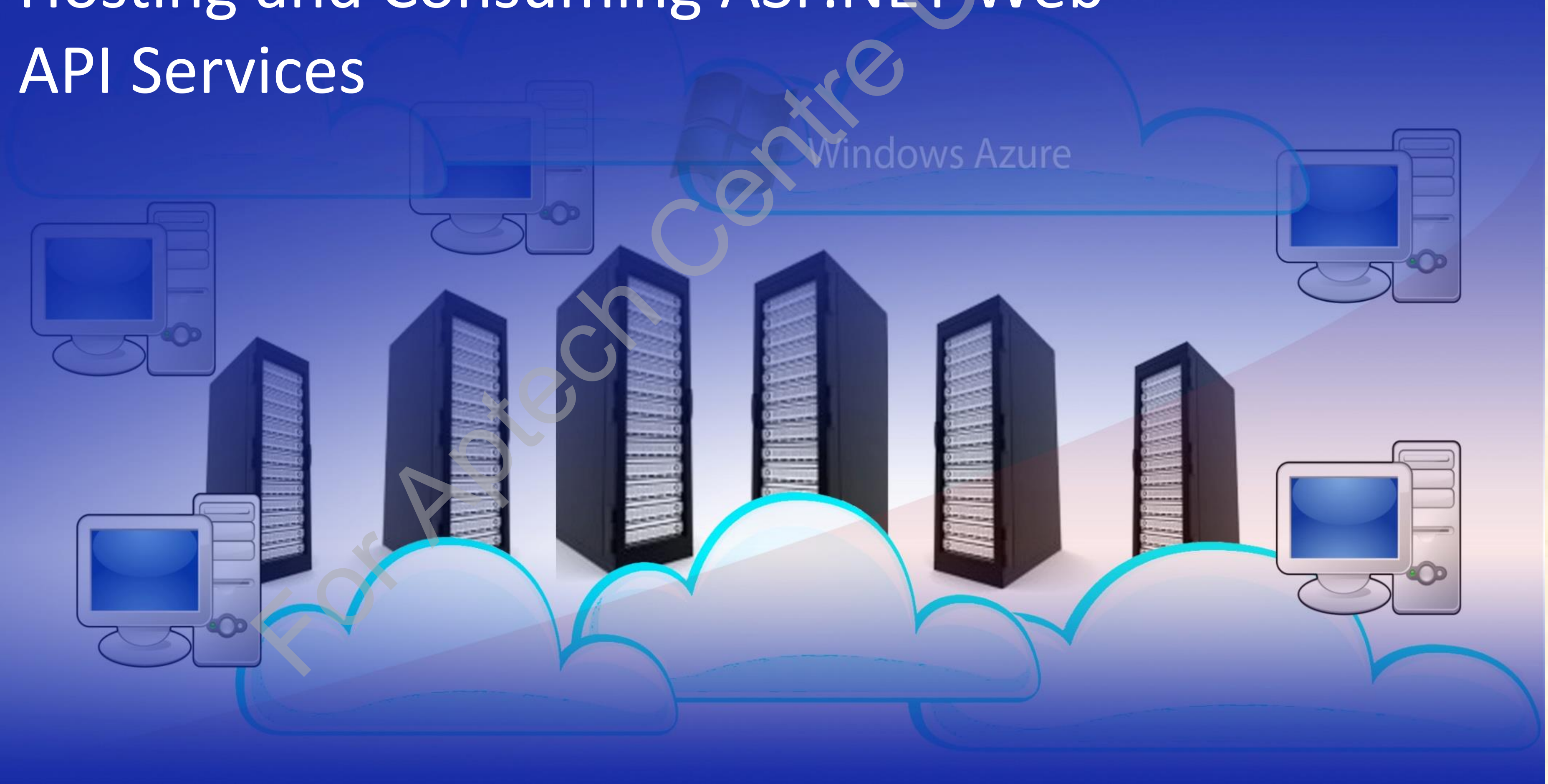
# Enterprise Application Development Using Windows Azure and Web Services

## Session 4

## Hosting and Consuming ASP.NET Web API Services



## Learning Objectives

- Define and describe how to host and manage a Web API service
- Explain how to consume a Web API service
- Explain how to host a Web API service in a Windows Azure worker role
- Define and describe the best practices to employ while developing Web API services

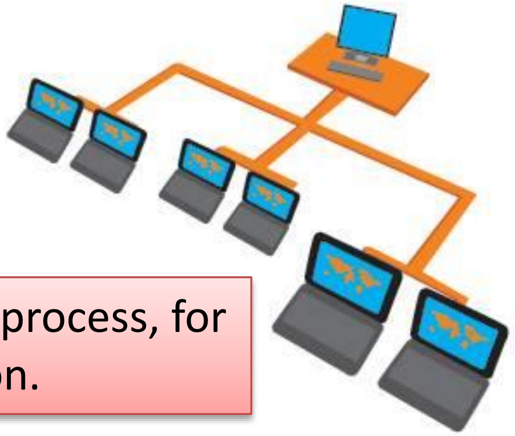
## Hosting and Managing a Web API Service

- If an ASP.NET Web API service is part of a Web application, it needs to be hosted on a Web server, such as IIS.

- You can also:

Create a service and host the service in a self-hosted process, for example, a process that executes a console application.

Host ASP.NET Web services in a Windows Azure worker role.



## Hosting a Web API Service on IIS 1-7

- When you host an ASP.NET Web API service on a server, such as IIS:
  - The Web server is responsible for providing the runtime environment for the application to service client information.
  - The Web server provides several functionalities, such as:

Listeners that listen for incoming client requests at a specific port

Forwarding requests to the hosted application

Returning back the application response to the client

## Hosting a Web API Service on IIS 2-7

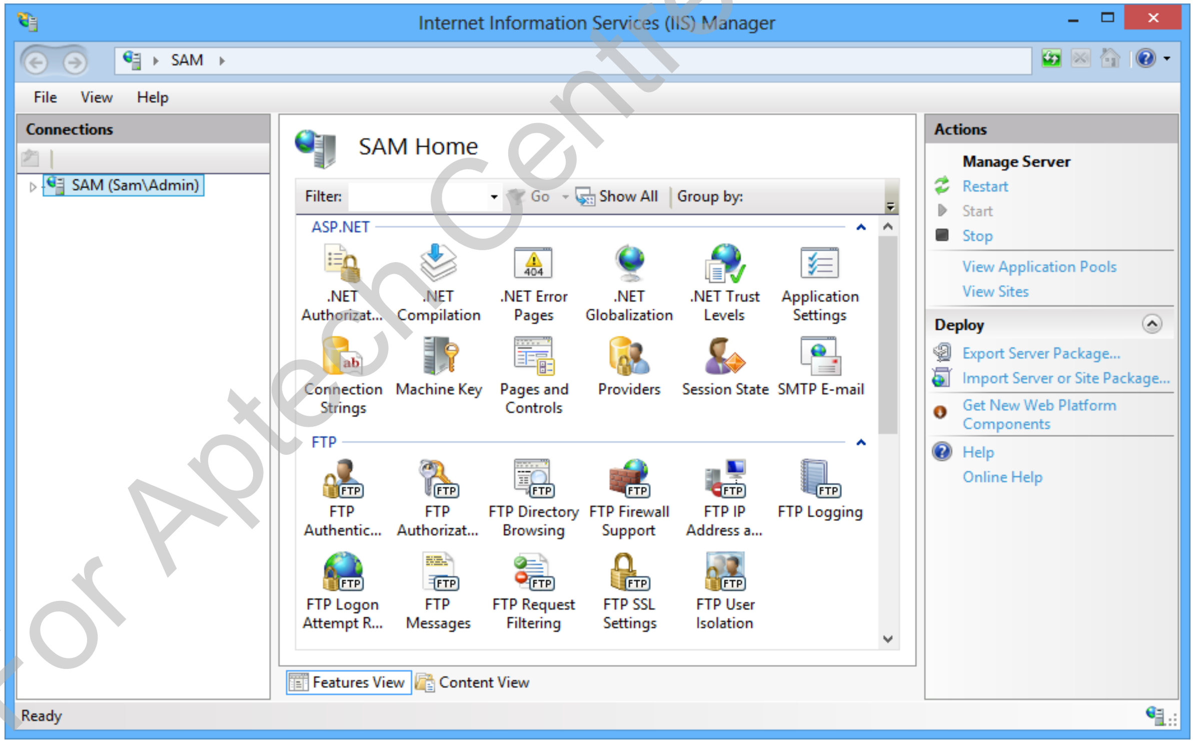
- Following are the steps to host a Web API service in Visual Studio 2013:

| | |
|---|---|
| Step 1 | • Start Visual Studio 2013 in **Administrator** mode and create a Web API project named **WebAPIIISHostingDemo** in Visual Studio 2013. |
| Step 2 | • Create a folder named **WebAPIIISHostingDemo** in the **C:\inetpub\wwwroot** folder. |
| Step 3 | • Press the **Windows+R** key. The **Run** dialog box is displayed. |

# Hosting a Web API Service on IIS 3-7

Step 4

- Type **inetmgr** in the **Open** text field and click **OK**.
- The **Internet Information Services (IIS) Manager** window is displayed.
- Following figure shows the **Internet Information Services (IIS) Manager** window:



# Hosting a Web API Service on IIS 4-7

Step 5

- Expand the connection node under the **Connections** pane.

Step 6

- Expand the **Sites** node.

Step 7

- Right-click the **Default Web Site** node under the **Sites** node and then, select **Add Application**. The **Add Application** dialog box is displayed.

Step 8

- In the **Add Application** dialog box, type **WebAPIIISHostingDemo** in the **Alias** text field and type **C:\inetpub\wwwroot\WebAPIIISHostingDemo** in the **Physical path** text field.

Step 9

- Click **OK** in the **Add Application** dialog box. The **WebAPIIISHostingDemo** node is displayed under the **Connection** node of the **Internet Information Services (IIS) Manager** window.

# Hosting a Web API Service on IIS 5-7

Step 10
- Close the **Internet Information Services (IIS) Manager** window.

Step 11
- Right-click the **WebAPIIISHostingDemo** application in the **Solution Explorer** window of **Visual Studio 2013** and then select **Publish.** The **Publish Web** dialog box is displayed.

Step 12
- From the **Select or import a publish profile** drop-down list, select the **<New…>** option. The **New Profile** dialog box is displayed.

Step 13
- Type **WebAPIIISHostingDemo** in the **Profile name** text field.

Step 14
- Click **OK.** The **Publish Web** dialog box is displayed. Ensure that **Web Deploy** is selected in the **Publish method** drop-down list.

# Hosting a Web API Service on IIS 6-7

Step 15

- In the **Server** text field, type **localhost** and in the **Site name** text field, type **DefaultWebSite/WebAPIIISHostingDemo**.

Step 16

- Click **Validate Connection**. When the connection is valid, a correct mark is displayed by the side of the **Validate Connection** button.

Step 17

- Click **Next**. The **Publish Web** dialog box is displayed.

Step 18

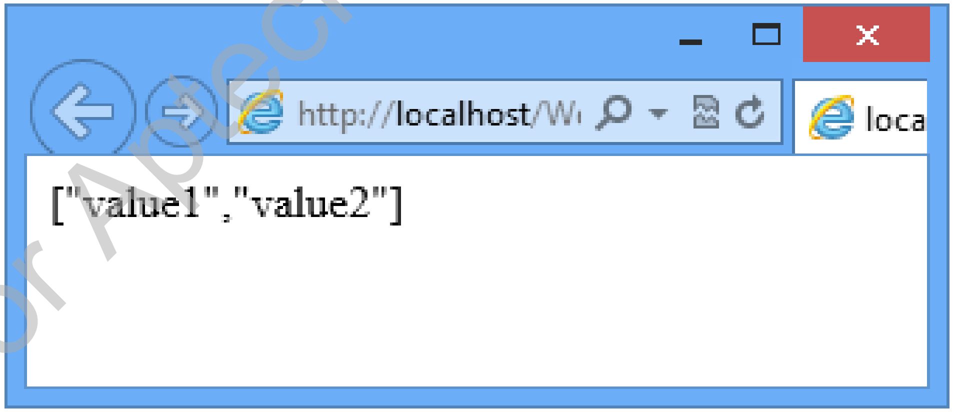
- Click **Publish**. The **Output** window of Visual Studio 2013 displays a message to indicate that the project has been successfully published.

## Hosting a Web API Service on IIS 7-7

- Test the published application by typing following URL in the browser:

  http://localhost/WebAPIIISHostingDemo/api/values

- Following figure shows the response of the Web API service hosted on IIS:



```
["value1","value2"]
```

## Self-Hosting a Web API Service 1-3

- For a standalone Web API service, you do not require a Web server, such as IIS to host the service.

- You can create a host process and self-host the Web API service on it.
- For example:
  - You have the flexibility to self-host a Web API service on a host process that runs a console application.
  - To self-host a Web API service, you can use Open Web Interface for .NET (OWIN) that acts as an interface between Web servers and .NET applications.

## Self-Hosting a Web API Service 2-3

- **OWIN:**

  - Abstracts the functionalities for receiving requests from client.
  - Routes it to the appropriate Web API controller.
  - Returns the response to the client.

  - Defines an open API that enables you to build Web applications over a hosting platform by providing an architecture that is structured well and supports pluggability.

## Self-Hosting a Web API Service 3-3

- In order to self-host a Web API service using OWIN, you need to:
  - Create a Web API service as a console application and configure it for self-hosting.
  - Start the OWIN host in order to consume the service.

OWIN

# Creating and Configuring a Standalone Web API Service 1-4

- Steps to create a standalone Web API service using Visual Studio 2013 are:

| | |
|---|---|
| Step 1 | Open Visual Studio 2013 in **Administrator** mode. |
| Step 2 | Click **File → New → Project**. The **New Project** dialog box is displayed. |
| Step 3 | Expand the **Installed → Templates → Visual C#** node and select **Windows** on the left pane of the **New Project** dialog box. On the right pane, select **Console Application**. |
| Step 4 | Type **WebAPIHostingDemo** in the **Name** text field. |

# Creating and Configuring a Standalone Web API Service 2-4

| | |
|---|---|
| Step 5 | Click **OK**. The **Solution Explorer** window displays the newly created **WebAPIHostingDemo** project. |
| Step 6 | From the **Tools** menu, click **Library Package Manager** and then, click **Package Manager Console**. |
| Step 7 | Enter the following command in the **Package Manager Console** window: `Install-Package Microsoft.AspNet.WebApi.OwinSelfHost` This command starts the OWIN installation. |
| Step 8 | In **Solution Explorer**, right-click the project and select **Add → Class** to add a new class. |
| Step 9 | Type `StartupService.cs` in the **Name** text field. In the `StartupService` class, add a `Configuration()` method that accepts an `IAppBuilder` object. This object is responsible for providing a host to an application. |

# Creating and Configuring a Standalone Web API Service 3-4

- Following code shows the `StartupService` class:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web.Http;
using Owin;
namespace WebAPIHostingDemo {
   class StartupService
  {
   public void Configuration(IAppBuilderappBuilder) {
         HttpConfigurationconfig = new HttpConfiguration();
         config.Routes.MapHttpRoute(name: "OwinApi",
         routeTemplate: "owin/{controller}/{id}",
         defaults: new { id = RouteParameter.Optional }
         );

         appBuilder.UseWebApi(config);
        }
    }
}
```

## Creating and Configuring a Standalone Web API Service 4-4

- In this code:
  - The `Configuration()` method creates an `HttpConfiguration` object that represents a configuration of an HTTP server.
  - The `MapHttpRoute()` method is used to map a request URL to a route template defined by the `routeTemplate` parameter.
  - The `UseWebApi()` method of the `IAppBuilder` object is called passing the initialized `HttpConfiguration` object as parameter.

# Adding the Web API Controller 1-2

- You should add a Web API controller that will implement the service and provide the response, after successful installation of OWIN and configuration of the host for a Web API service.

- Steps to add a Web API controller in Visual Studio 2013 are:

Step 1
- Right-click the **WebAPIHostingDemo** project in the **Projects Solution** and select **Add → Class**. The **Add New Item – WebAPIHostingDemo** dialog box appears.

Step 2
- Type **WAController** in the **Name** field and click the **Add** button.

Step 3
- In the **WAController** class, add the HTTP methods to handle requests.

# Adding the Web API Controller 2-2

- Following code snippet shows the `WAController` class:

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web.Http;
namespace WebAPIHostingDemo {
   public class WAController : ApiController {
       public IEnumerable<string> Get() {
            return new string[] { "Henry", "Mark", "Mary" };
       }
   }
}
```

- In this code:
  - The `WAController` class extends the `ApiController` class and provides a `Get()` method.
  - This method returns a string array as an `IEnumerable` object.

# Consuming the Web API Service 1-4

- To consume the Web API service, you need to:

Start the OWIN host and send an asynchronous request to the service.

→

Update the `Main()` method of the `Program` class in the **WebAPIHostingDemo** project that Visual Studio 2013 creates for you by default.

→

Implement the functionality in the `Main()` method to start the OWIN host, access the Web API service, and print out the response of the service.

# Consuming the Web API Service 2-4

- Following code snippet shows the `Program` class:

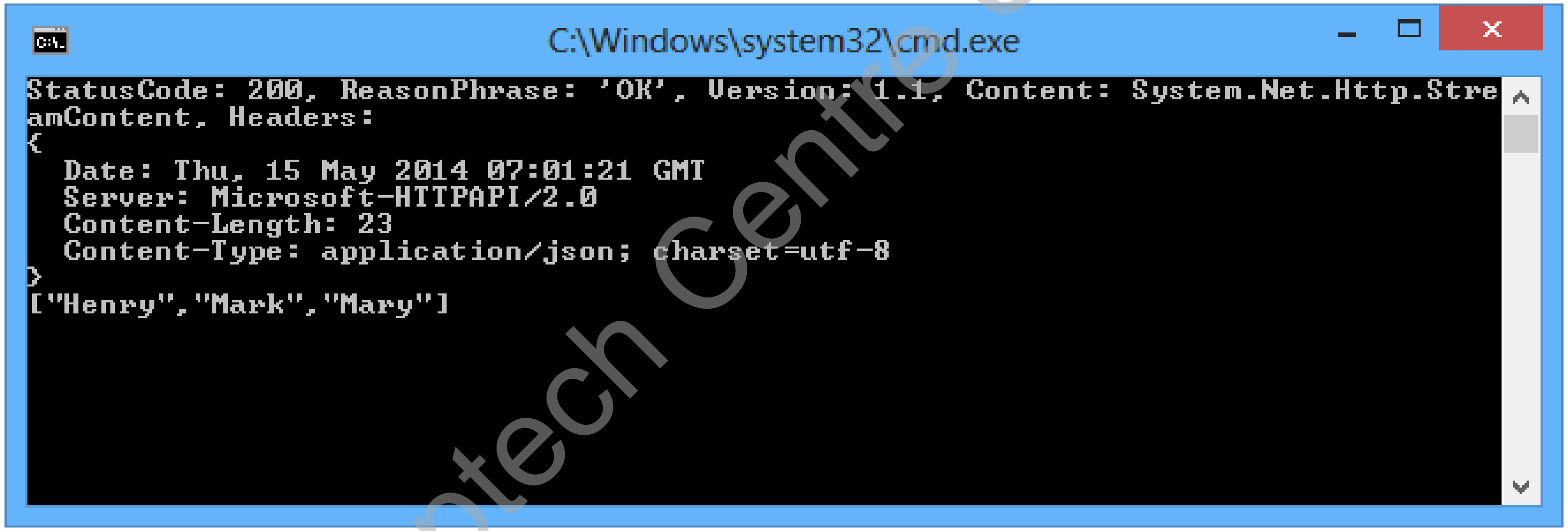
```
using Microsoft.Owin.Hosting;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

namespace WebAPIHostingDemo {
   class Program {
      static void Main(string[] args) {
         WebApp.Start<StartupService>(url: "http://localhost:9095/");
         HttpClient client = new HttpClient();
         Task<HttpResponseMessage>
         resp=client.GetAsync("http://localhost:9095/owin/WA");
         var response = resp.Result;
         Task<String > respMessage=response.Content.ReadAsStringAsync();
         Console.WriteLine(response);
         Console.WriteLine(respMessage.Result);
         Console.ReadLine(); }
   }
}
```

## Consuming the Web API Service 3-4

- In this code:
  - The `Main()` method specifies a base address to access the Web API service.
  - Then, it calls the `WebApp.Start<StartupService>()` method to use a `StartupService` object for starting the service at the base URL passed as parameter.
  - Next, the `GetAsync()` method of the `HttpClient` class is called passing the Web API service URL as a string.
  - This method returns a `Task<HttpResponseMessage>` object that represents an asynchronous operation returning an HTTP response message as an `HttpResponseMessage` object.
  - Finally, the response is retrieved by a call to the `ReadAsStringAsync()` method and the response object along with the response result is displayed as output.

## Consuming the Web API Service 4-4

- Following figure shows the response of the Web API service:

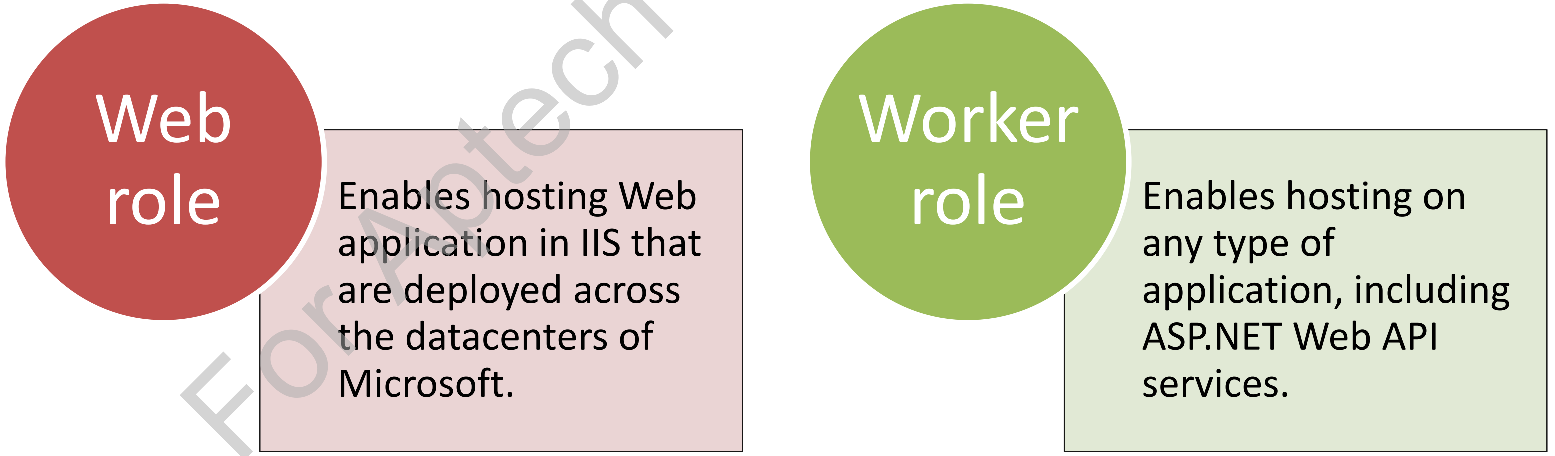


```
StatusCode: 200, ReasonPhrase: 'OK', Version: 1.1, Content: System.Net.Http.Stre
amContent, Headers:
{
   Date: Thu, 15 May 2014 07:01:21 GMT
   Server: Microsoft-HTTPAPI/2.0
   Content-Length: 23
   Content-Type: application/json; charset=utf-8
}
["Henry","Mark","Mary"]
```

# Hosting Services in a Windows Azure Worker Role 1-2

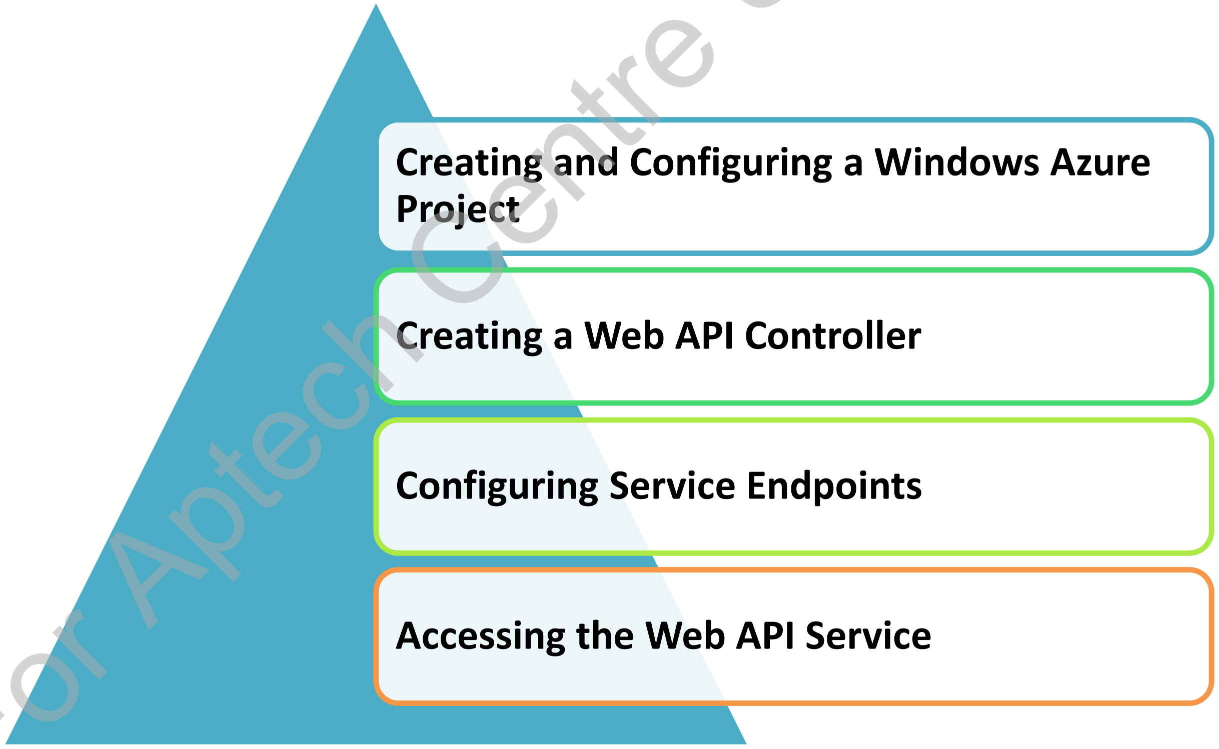
- Windows Azure provides Compute capabilities, which enables you to host applications in the cloud.

- A Compute acts as a container for Web and worker roles.


Web role
Enables hosting Web application in IIS that are deployed across the datacenters of Microsoft.
Worker role
Enables hosting on any type of application, including ASP.NET Web API services.

# Hosting Services in a Windows Azure Worker Role 2-2

- Perform these tasks to host an ASP.NET Web API service in a Windows Azure worker role:


Creating and Configuring a Windows Azure Project
Creating a Web API Controller
Configuring Service Endpoints
Accessing the Web API Service

# Creating and Configuring a Windows Azure Project 1-5

- Following are the steps to create and configure a Windows Azure project:

| | |
|---|---|
| Step 1 | • Open Visual Studio 2013 in **Administrator** mode. |
| Step 2 | • Click **File → New → Project**. The **New Project** dialog box is displayed. |
| Step 3 | • Expand the **Installed → Templates → Visual C#** node and select **Cloud** on the left pane of the **New Project** dialog box. On the right pane, select **Windows Azure Cloud Service**. |
| Step 4 | • Type **AzureServiceDemo** in the **Name** text field in the **New Project** dialog box. |
| Step 5 | • Click **OK**. The **New Windows Azure Cloud Service** window is displayed. |

# Creating and Configuring a Windows Azure Project 2-5

Step 6

- Select **Worker Role** under the **.NET Framework 4.5 roles** section.

Step 7

- Click the (**>**) button to add the selected role to the **New Windows Azure Cloud Service solution** section.

Step 8

- Click **OK**. The **Solution Explorer** window displays the following two projects:
  - **AzureServiceDemo**: Defines the roles and configuration for the Azure application.
  - **WorkerRole1**: Contains the code for the worker role.

Step 9

- Click **Tools → Library Package Manager → Package Manager Console**. The **Package Manager Console** window is displayed.

Step 10

- Type the following command in the **Package Manager Console** window:

```
Install-Package Microsoft.AspNet.WebApi.OwinSelfHost
```

# Creating and Configuring a Windows Azure Project 3-5

Step 11

- In **Solution Explorer**, right-click the **WorkerRole1** project and select **Add → Class**. The **Add New Item – WorkerRole1** dialog box is displayed.

Step 12

- Type **Startup.cs** in the **Name** text field.

Step 13

- Click **Add**. The **Code Editor** displays the code of the `Startup` class.

Step 14

- In the `Startup` class, add a `Configuration()` method to configure the application.

# Creating and Configuring a Windows Azure Project 4-5

- Following code snippet shows the `Startup` class:

```
using Owin;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web.Http;

namespace WorkerRole1 {
   class Startup {
      public void Configuration(IAppBuilderappBuilder) {
          HttpConfigurationconfig = new HttpConfiguration();
          config.Routes.MapHttpRoute(
          name: "Default",
          routeTemplate: "{controller}/{id}",
          defaults: new { id = RouteParameter.Optional }
          );
          appBuilder.UseWebApi(config);
      }
   }
}
```

## Creating and Configuring a Windows Azure Project 5-5

- In this code:

  - A `HttpConfiguration` object is created to define a route named `Default` with the `{controller}/{id}` routing pattern where the `id` part is optional.

## Creating a Web API Controller 1-4

- After creating the `Startup` class, you need to add a Web API controller class.
- Following are the steps to add a Web API controller class:

| | |
|---|---|
| Step 1 | • Right-click the **WorkerRole1** project and select **Add → Class**. The **Add New Item – WorkerRole1** dialog box is displayed. |
| Step 2 | • Type **TestController.cs** in the **Name** field and click the **Add** button. |
| Step 3 | • In the **`TestController`** class, add the HTTP methods to handle requests. |

## Creating a Web API Controller 2-4

- Following code snippet shows the `TestController` class:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web.Http;

namespace WorkerRole1 {
    class TestController : ApiController {
        public IEnumerable<string> Get() {
           return new string[] { "Laptop", "Desktop", "Mobile", "Printer",
                  "PS2" };
        }
    }
}
```

- In this code:
  - The `Get()` method of the controller returns a string array.

# Creating a Web API Controller 3-4

Step 4

- In the **Solution Explorer**, double-click the **WorkerRole.cs** file.
- In the `WorkerRole` class, override the `OnStart()` method of the `RoleEntryPoint` class. The `OnStart()` method is called to initialize a worker role object.

- Following code snippet shows the `WorkerRole` class:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net;
using System.Threading;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.Diagnostics;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.Storage;
using Microsoft.Owin.Hosting;
namespace WorkerRole1 {
  public class WorkerRole : RoleEntryPoint {
       private IDisposable _app = null;
```

## Creating a Web API Controller 4-4

```
    public override void Run() {
       Trace.TraceInformation("WorkerRole1 entry point called");
       while (true) {
       Thread.Sleep(10000);
       Trace.TraceInformation("Working");
       }
    }
    public override boolOnStart(){
      ServicePointManager.DefaultConnectionLimit = 12;
      var testEndpoint =
      RoleEnvironment.CurrentRoleInstance.InstanceEndpoints["TestEndpoint"];
      string baseUri = String.Format("{0}://{1}/test",
      testEndpoint.Protocol, testEndpoint.IPEndpoint);
      _app = WebApp.Start<Startup>(new StartOptions(url: baseUri));
      returnbase.OnStart();
         }
     }
}
```

- This code overrides the `OnStart()` method of the `RoleEntryPoint` class.

# Configuring Service Endpoints 1-2

- After creating the Web API controller, you configure the service endpoint using these steps:

Step 1

- In the **Solution Explorer** window, expand the **AzureServiceDemo** node. The **Roles** node is displayed.
- Right-click **WorkerRole1** under **Roles** and select **Properties** from the context menu that appears. Visual Studio 2013 displays the properties of the **AzureServiceDemo** project.

Step 2

- Click **Endpoints**.

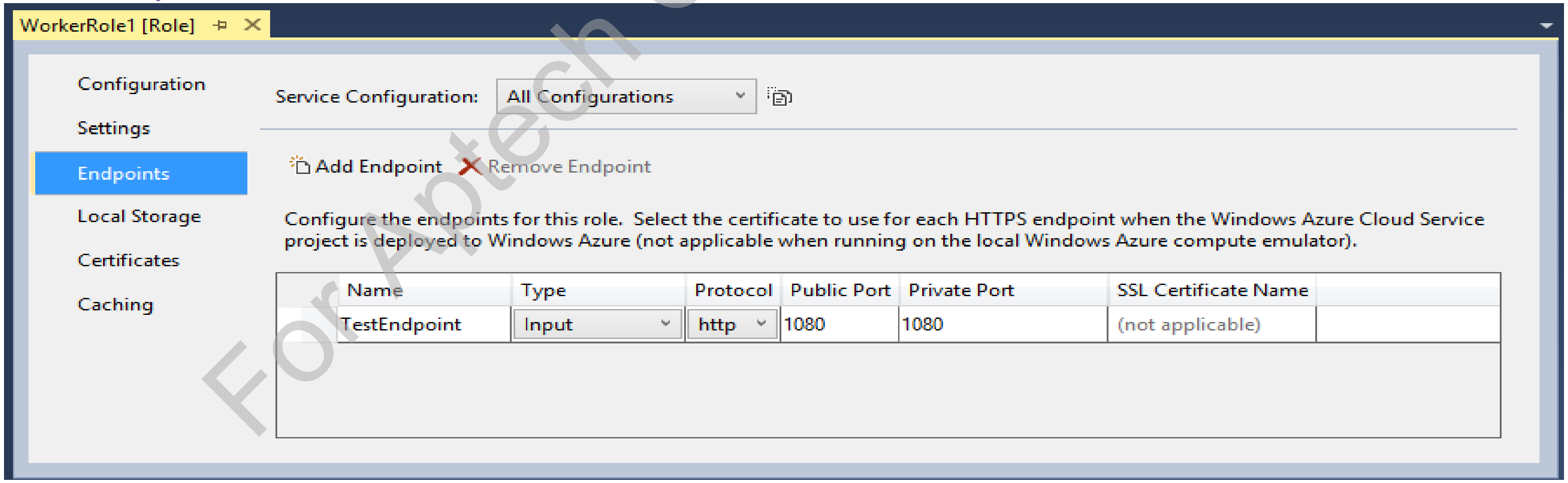
# Configuring Service Endpoints 2-2

Step 3

- Click **Add Endpoint**.

Step 4

- Type **TestEndpoint** in the **Name** field. Select **http** from the **Protocol** drop-down list and type **1080** in the **Public Port** and **Private Port** fields.
- These port numbers can be different.
- The public port is what clients use when they send a request to the role.
- Following figure shows adding endpoint:



Service Configuration: All Configurations

Configure the endpoints for this role. Select the certificate to use for each HTTPS endpoint when the Windows Azure Cloud Service project is deployed to Windows Azure (not applicable when running on the local Windows Azure compute emulator).

| Name | Type | Protocol | Public Port | Private Port | SSL Certificate Name |
|---|---|---|---|---|---|
| TestEndpoint | Input | http | 1080 | 1080 | (not applicable) |

# Accessing the Web API Service

- After configuring the service endpoint, you can access the Web API service.

- Following are the steps to access the Web API service:

1. Click **Build → Build Solution**.

2. Click **Debug → Start Debugging**. The **Azure Compute Emulator** assigns **127.0.0.1** as the **IP address** of the endpoint and **1080** as the **port number** that you specified while configuring the service endpoint.

# Best Practices for HTTP Services Using ASP.NET Web API

- Some industry-proven and tested best practices that should be applied when creating HTTP services using ASP.NET Web API are:

✓ Create a Simple Controller

✓ Use Models

✓ Use Attribute Routing

## Create a Simple Controller 1-2

- It is recommended to keep the controller as simple as possible.
- For example:
  - You should try to separate data access code using data access technology, such as Entity Framework from the controller to a data access service.
  - As a result, you will not require updating data access code of each controller when the data source of the application changes.
  - You will only need to update the data access service implementation and the application can use the new data source, thereby significantly decreasing maintenance and code changes in the application.

## Create a Simple Controller 2-2

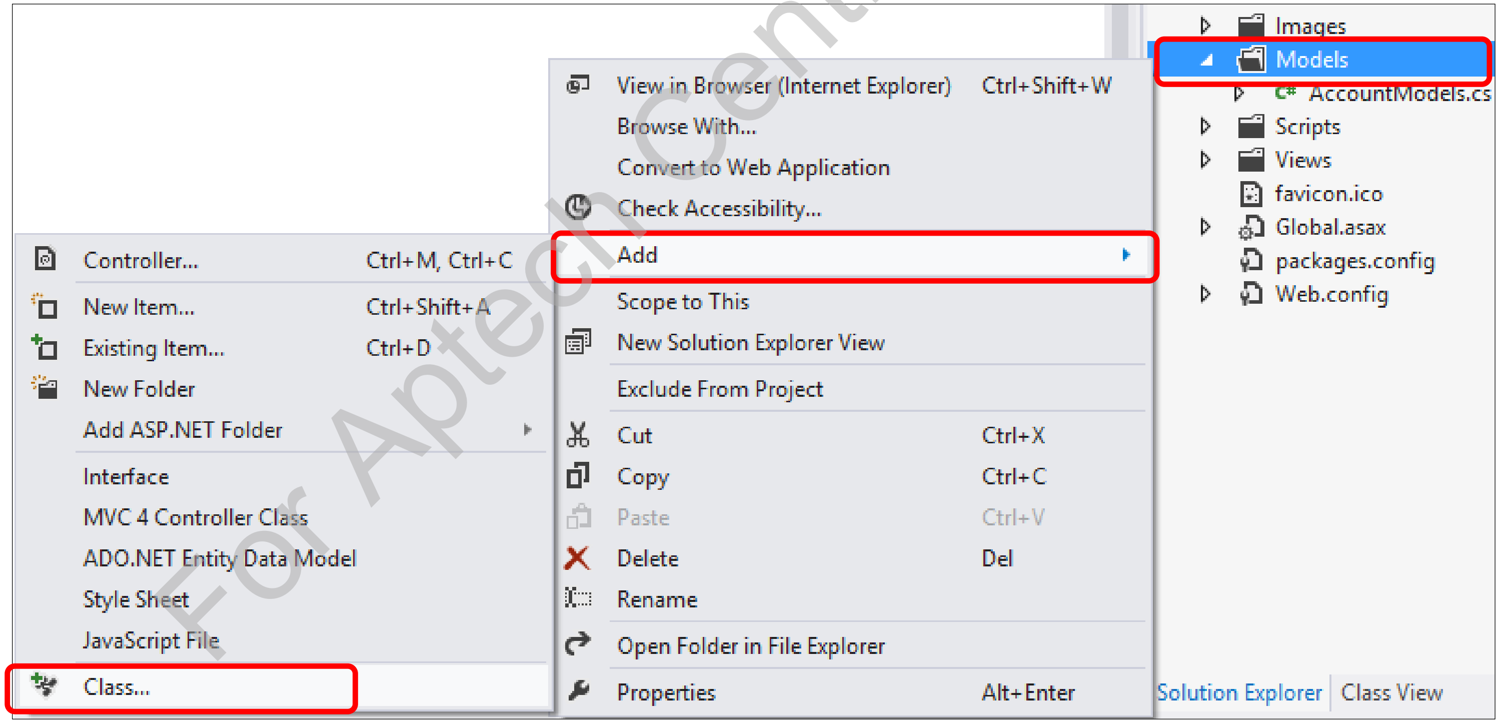
- Following code snippet shows a controller that uses an implementation of an application-specific `IDataService` interface to access data stored in a data store:

```
public class ProductsController : ApiController
{
    private readonly IDataService dataService;
    public ProductController(IDataService dataservice)
    {
           dataService = dataservice;
    }
    public IEnumerable<ProductModel> Get()
    {
          return dataService.AllProducts().AsModel();
    }
}
```

- In this code:
  - The controller is initialized with an `IDataService` implementation. In the `Get()` method, the controller uses the `IDataService` implementation to retrieve and return the records of all products that the `ProductModel` object represents.

## Use Models 1-2

- You should always create and use models to expose data of a Web API service.
- A client request for a service does not always require all information of an entity.



## Use Models 2-2

- For example:
  - A client might request only the product category and price for a particular product.
  - You should ensure that your service provides only the specific information instead of returning all the details of the product.
  - This can be achieved by creating a `Product` model with public properties that should be made accessible to clients.
  - Similarly, you should ensure that your service does not expose potentially sensitive information.
  - In such situations, you should create a model to represent the information that the service returns instead of returning all data that includes the sensitive data to the client.

## Use Attribute Routing 1-5

- In ASP.NET Web API applications:
  - It is common to use convention-based routing where you register routes with the ASP.NET Routing Framework using the `WebApiConfig.cs` file.
- ASP.NET Web API 2 introduces:
  - Attribute routing that allows you to explicitly define routes for a controller and its actions.

# Use Attribute Routing 2-5

- When you work on a large scale project being developed by multiple developers:
  - You should consider using attribute routing.
  - As attribute routing is directly applied to controllers and actions, it becomes simpler and intuitive for developers to understand how to individually call and test actions and controller.
  - As the number of routes in the application increases, the task to manage the routes in the route table becomes cumbersome and error prone.
  - You can use attribute routing by applying the `RoutePrefix` attribute to the controller. This attribute specifies a common prefix for an entire controller.
- Following code snippet shows applying the `RoutePrefix` attribute to a controller:

```
[RoutePrefix("api/customers")]
public class CustomersController : ApiController
```

## Use Attribute Routing 3-5

- When you use the `RoutePrefix` attribute on a controller, the action methods of the controller inherits the route prefix as the beginning of their route.
- You can apply attribute routing in an action method by applying the `Route` attribute.
- The value specified for the `Route` attribute will follow the value specified for the `RoutePrefix` attribute on the controller.

## Use Attribute Routing 4-5

- Following code snippet shows two actions that use the `Route` attribute:

```
[RoutePrefix("api/customers")]
public class CustomersController : ApiController
{
    private readonly IDataService dataService;
    public ProductController(IDataService dataservice)
    {
        dataService = dataservice;
    }
    [Route("")]
    public IEnumerable<ProductModel> Get(){
         return dataService.AllProducts().AsModel();
    }
    [Route("{id}")]
    public ProductModel Get(int id)
    {
        var product = dataService.GetProduct(id);
        return Ok(product.AsModel());
    }
}
```

# Use Attribute Routing 5-5

- In this code:
  - The first `GET` method uses the `Route` attribute with an empty string. Therefore, the route defined in the controller will apply to it.
  - The second `GET` method specifies a placeholder for the product ID that will get appended to the route defined in the controller.
  - For example, the URL

    **`http://www.webapiexample.com/api/customers`**

    will invoke the first `GET` method, while the URL

    **`http://www.webapiexample.com/api/customers/5`**

    will invoke the second `GET` method.

## Summary 1-2

- An ASP.NET Web API service must be accessible over the Web to make its services available to different types of clients.
- When you provide a Web API service as a Web application, the application is hosted on a server, such as IIS.
- Using OWIN, you can self-host a Web API service.
- Windows Azure Compute acts as a container for Web and worker roles that enables you to host applications in the cloud.

## Summary 2-2

- Key best practices when creating HTTP services using ASP.NET Web API include creating a simple controller, using models, and attribute routing.
- To expose data of a Web API service, you should create and use models.
- Routing in ASP.NET Web API 2 allows you to explicitly define routes for a controller and its actions.