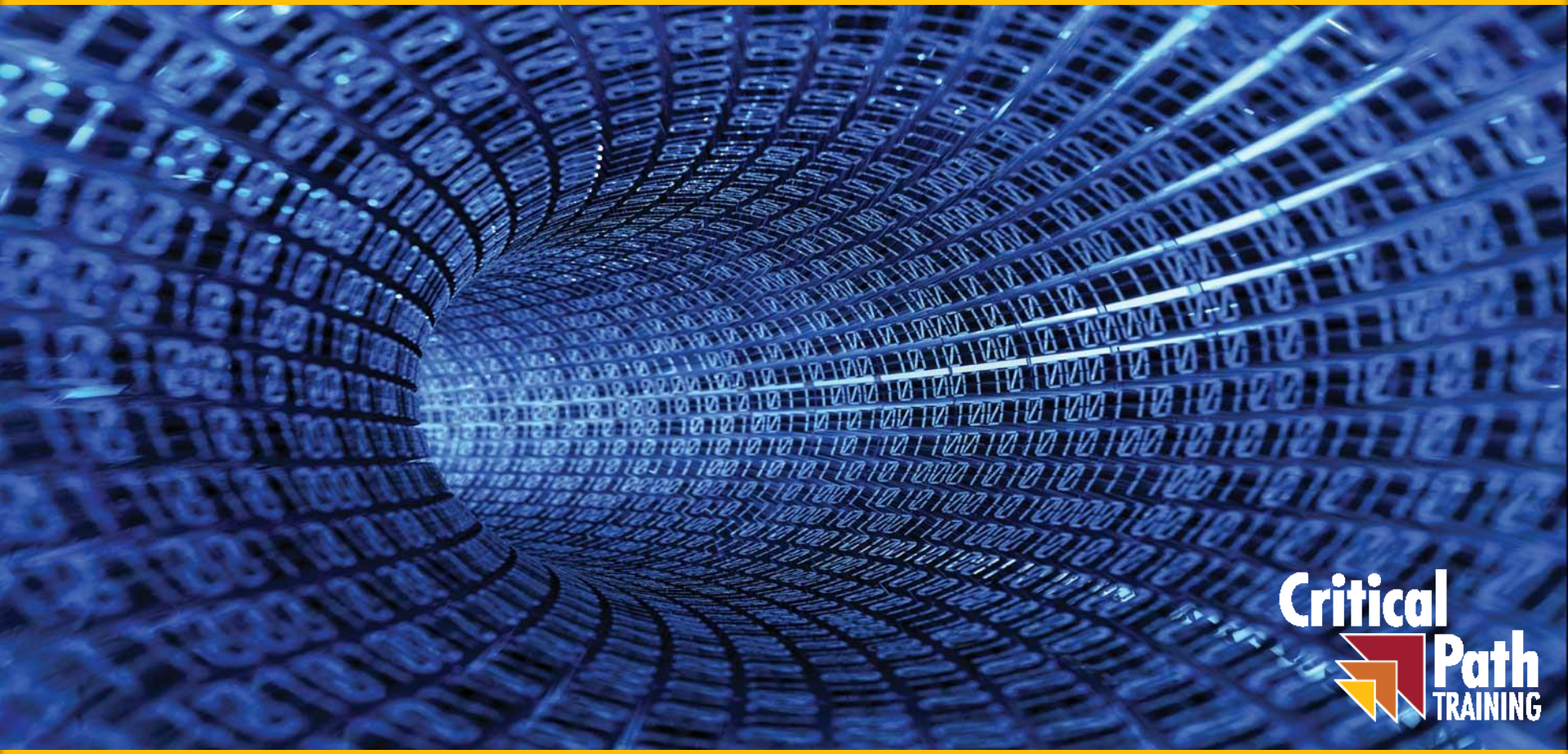
Developing Custom Web Services using Web API



Agenda

- Introduction to Web API
- Developing RESTful Web Services
- Configuring Attribute-based Routing
- Developing Web Services that support ODATA
- Cross-Origin Resource Sharing (CORS)

RESTful Web Services

- RESTful Web Service
  - implemented using the principles of REST
  - REST URI = [base URI] + [resource path] + [query options]
  - Calls based on standard HTTP verbs (**GET**, **POST**, **PUT**, **DELETE**)
  - Passes data to and from client using representations
  - Can be designed to implement custom APIs and/or standard APIs

- Data passed across network using representations
  - Representations model resources – but they're different
  - Based on common formats: HTML, XML, ATOM and JSON
  - Based on specific Internet media types

Internet Media Types

- Internet media type defines format of representation
  - Commonly referred to as Content Types
    previous known as MIME types
  - Examples of common Internet media types

```
text/html
text/xml
application/xml
application/atom+xml
application/json
```

- HTTP headers used to indicate Internet Media Type
  - **Accept** request header indicates what client wants in response
  - **Content-Type** header indicates type of request/response body

Setting Header values with jQuery $.ajax()

- Set **Accept** header when retreiving data from REST-based service

```
$.ajax({
  url: requestUri,
  type: "GET",
  headers: { "Accept": "application/json" }
});
```

- Set **Content-Type** header when passing data to REST-based service

```
var customerData = {
  Title: LastName,
  FirstName: FirstName,
  Company: Company,
  WorkPhone: WorkPhone,
  HomePhone: HomePhone,
  Email: Email
};

var requestBody = JSON.stringify(customerData);

return $.ajax({
  url: requestUri,
  type: "POST",
  data: requestBody,
  headers: {
    "Content-Type": "application/json",
    "Accept": "application/json"
  },
});
```

Introducing WebAPI

- Framework and tooling for building RESTful services
- Part of ASP.NET MVC
  - Uses Controller and Routing paradigm
- Tooling, wizards, scaffolding
  - Simplified creation of REST and OData services
  - Simplified use of Entity Framework to wrap database operations
- Can be a stand-alone service or part of an app
  - When added to an app, you perform additional manual modifications to Global.asax

Controllers

- Controllers inherit from `ApiController`

```
public class ValuesController : ApiController
```

- By default methods are mapped to HTTP verbs

```
public IEnumerable<string> Get() {}

public string Get(int id) {}

public void Post([FromBody]string value){}

public void Put(int id, [FromBody]string value){}

public void Delete(int id){}
```

Routing

- Routes are controlled through maps

```
config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

- Router makes decisions if information is missing
  - Similar to MVC
- By default methods are mapped to HTTP verbs

Responding

- Content Negotiation is automatic
  - `accept: "application/json"`
  - `accept: "application/xml"`
- Return `IQueryable` to support query syntax

```
public IQueryable<string> Get()
{
    var d = new List<string>() {"a", "b" };
    return d.AsQueryable();
}
```

- Return `HttpResponseMessage` for headers and status

```
public HttpResponseMessage Get(int id)
{
  return Request.CreateResponse<string>(HttpStatusCode.OK, data[id - 1]);
}
```

Agenda

- ✓ Introducing WebAPI
- Calling API Controllers from MVC Apps
- Creating a RESTful Service
- Creating an OData Service
- Using Cross-Origin Resource Sharing

Scenarios to Consider

- Additional API Controllers NOT Required
  - You don’t require JavaScript access to the data
  - App has direct EF access to database anyway
- Additional API Controllers MAY be Required
  - You require JavaScript access to the data
  - Data source cannot be accessed directly using EF
  - Want to support clients outside of the App

Adding an API Controller

- Create a new Provider-Hosted App
  - Based on ASP.NET MVC Web Application template
- Add a new Web API 2 Controller
  - Use any template
- Modify the Global.asax file
  - Additional using statements
  - Additional configuration commands

Calling an API Controller

- Add a new MVC5 Controller
  - Use existing Index method
  - Or add a new method to an existing controller
- Add a View for the Method
- Call from Managed Code
  - HttpWebRequest
- Call from JavaScript
  - jQuery ajax

Calling with Managed Code

```
public ActionResult Index()
{
    StringBuilder url = new StringBuilder();
    url.Append(Request.Url.Scheme)
        .Append("://")
        .Append(Request.Url.Host)
        .Append(":")
        .Append(Request.Url.Port)
        .Append("/api/values");

    HttpWebRequest apiRequest = (HttpWebRequest)HttpWebRequest.Create(url.ToString());
    apiRequest.Credentials = CredentialCache.DefaultCredentials;
    apiRequest.Method = "GET";
    apiRequest.Accept = "application/xml";

    HttpWebResponse apiResponse = (HttpWebResponse)apiRequest.GetResponse();
    XDocument responseDoc = XDocument.Load(apiResponse.GetResponseStream());
    XNamespace ns = "http://schemas.microsoft.com/2003/10/Serialization/Arrays";
    List<string> values = (from v in responseDoc.Descendants(ns + "string")
                           select  v.Value ).ToList();

    ViewBag.Values = values;
    return View();
}
```

Calling with JavaScript

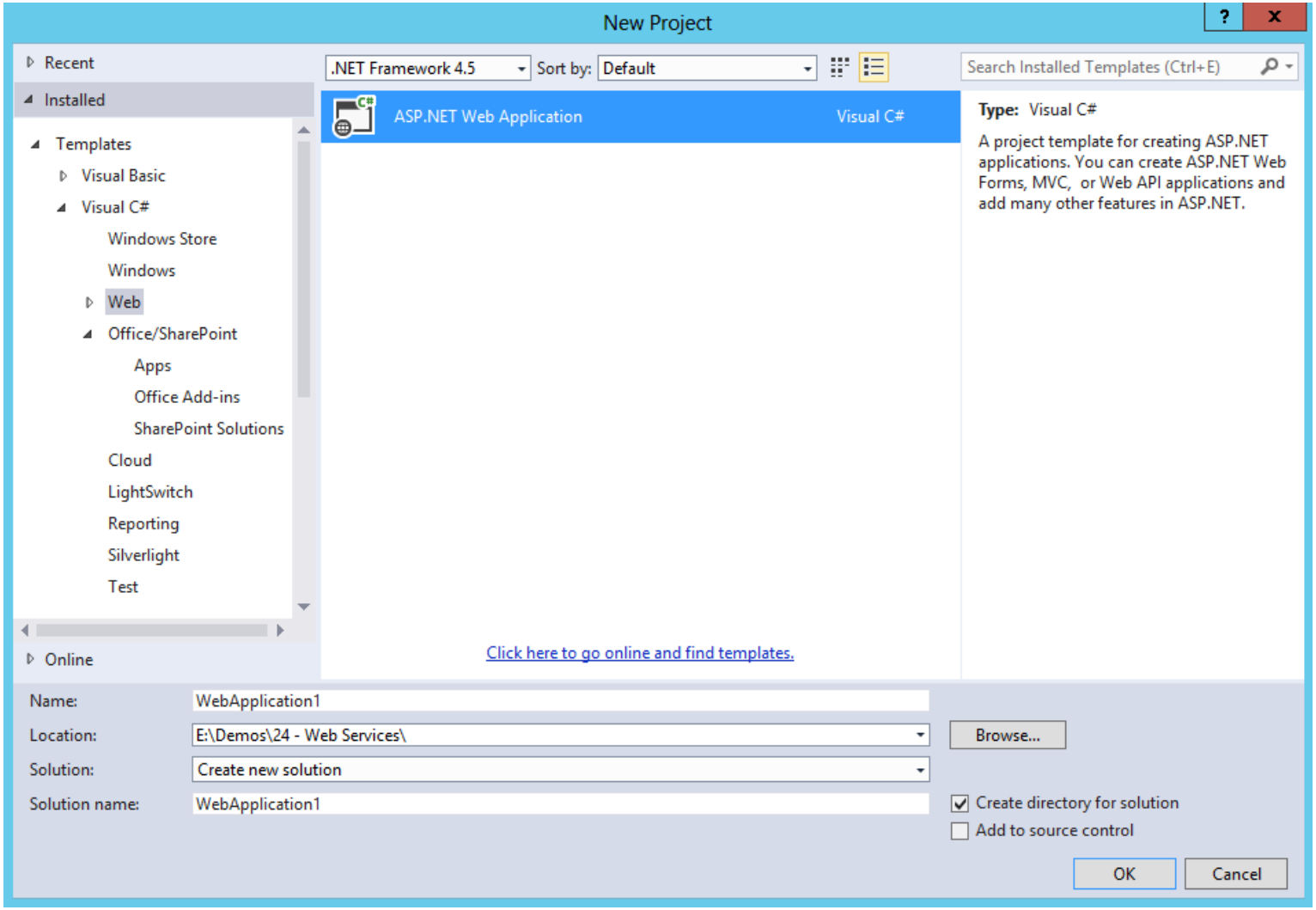
```
(function () {
    "use strict";

    jQuery(function () {

        jQuery.ajax({
            url: "../api/values",
            type: "GET",
            headers: {
                "accept": "application/json",
            },
            success: function (data, status, jqXHR) {
                alert(data[0]);
            },
            error: function (jqXHR, status, message) {
                alert(JSON.stringify(jqXHR));
            }
        });

    });

}());
```


DEMO
Calling API Controllers from MVC Apps

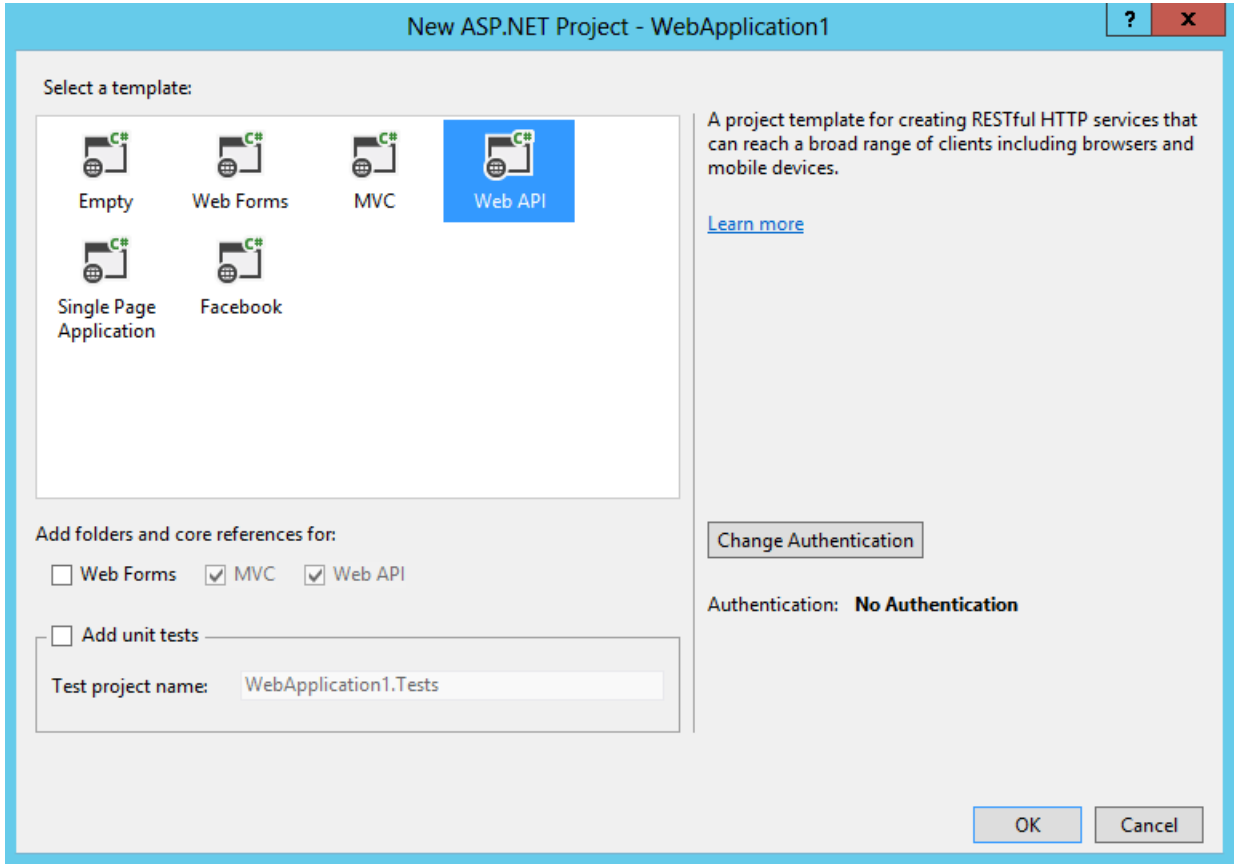
Agenda

- ✓ Introducing WebAPI
- ✓ Calling API Controllers from MVC Apps
- Creating a RESTful Service
- Creating an OData Service
- Using Cross-Origin Resource Sharing

Creating a Stand-Alone RESTful Service



Creating a Stand-Alone RESTful Service




DEMO
Creating and Testing a RESTful Service

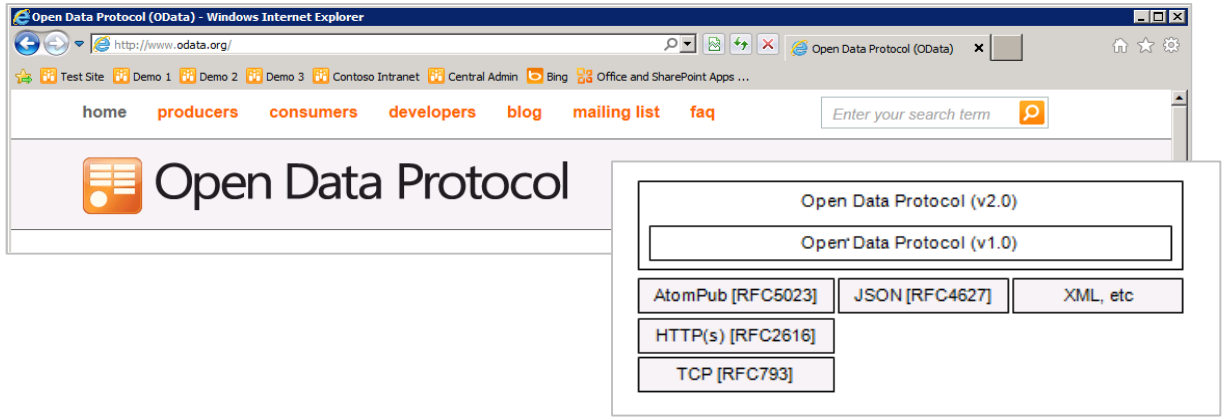
Agenda

- ✓ Introducing WebAPI
- ✓ Calling API Controllers from MVC Apps
- ✓ Creating a RESTful Service
- Creating an OData Service
- Using Cross-Origin Resource Sharing

OData Primer

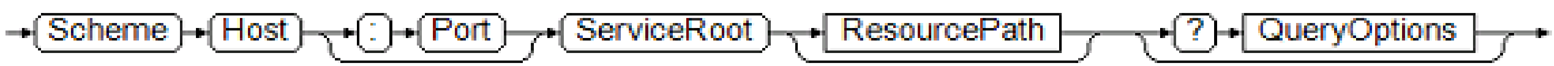
- What is OData?
  - A standardized REST API interface for common CRUD operations
  - Defined by Open Data Protocol specification
  - OData services becoming more popular on Internet (e.g. NetFlix)
  - SharePoint 2010 introduced a REST API for dealing with list items
  - SharePoint 2013 introduces new and expanded REST API

for an excellent resource go to
http://www.odata.org



OData URIs

- URI has three significant parts
  - Service root URI
  - Resource path
  - Query string options




```
http://services.odata.org/OData/OData.svc/Category(1)/Products?$top=2&$orderby=name
\______________________________________/ \__________________/  \_________________/
                    |                              |                    |
             service root URI               resource path         query options
```

Returning OData Results as ATOM vs. JSON

- ATOM-PUB (XML)
  - Good when you want to read and write XML
  - Set **Accept** header to **application/atom+xml**
- JavaScript Object Notation (JSON)
  - Smaller payload than XML for same data
  - ODATA v4 - Set **Accept** header to **application/json**
  - ODATA v3 - Set **Accept** header to **application/json;odata=verbose**

OData Query Option Parameters

- **$select**
  - http://services.odata.org/OData/OData.svc/Products?$select=Price,Name
- **$filter**
  - http://services.odata.org/OData/OData.svc/Products?$filter=startswith(CompanyName, 'Alfr')
- **$orderby**
  - http://services.odata.org/OData/OData.svc/Products?$orderby=Rating
- **$top**
  - http://services.odata.org/OData/OData.svc/Products?$top=5
- **$skip**
  - http://services.odata.org/OData/OData.svc/Products?$skip=5
  - http://services.odata.org/OData/OData.svc/Products?$skip=5&$top=5
- **$expand**
  - http://services.odata.org/OData/OData.svc/Categories?$expand=Products

OData Query Options

- $select
- $filter
- $orderby
- $top
- $skip
- $expand

Controllers

- Controllers inherit from `ODataController`

```
public class ContactsController : ODataController
```

- Methods are mapped to HTTP verbs just like `ApiController`
- Content Negotiation is automatic
- `IQueryable` generated by default

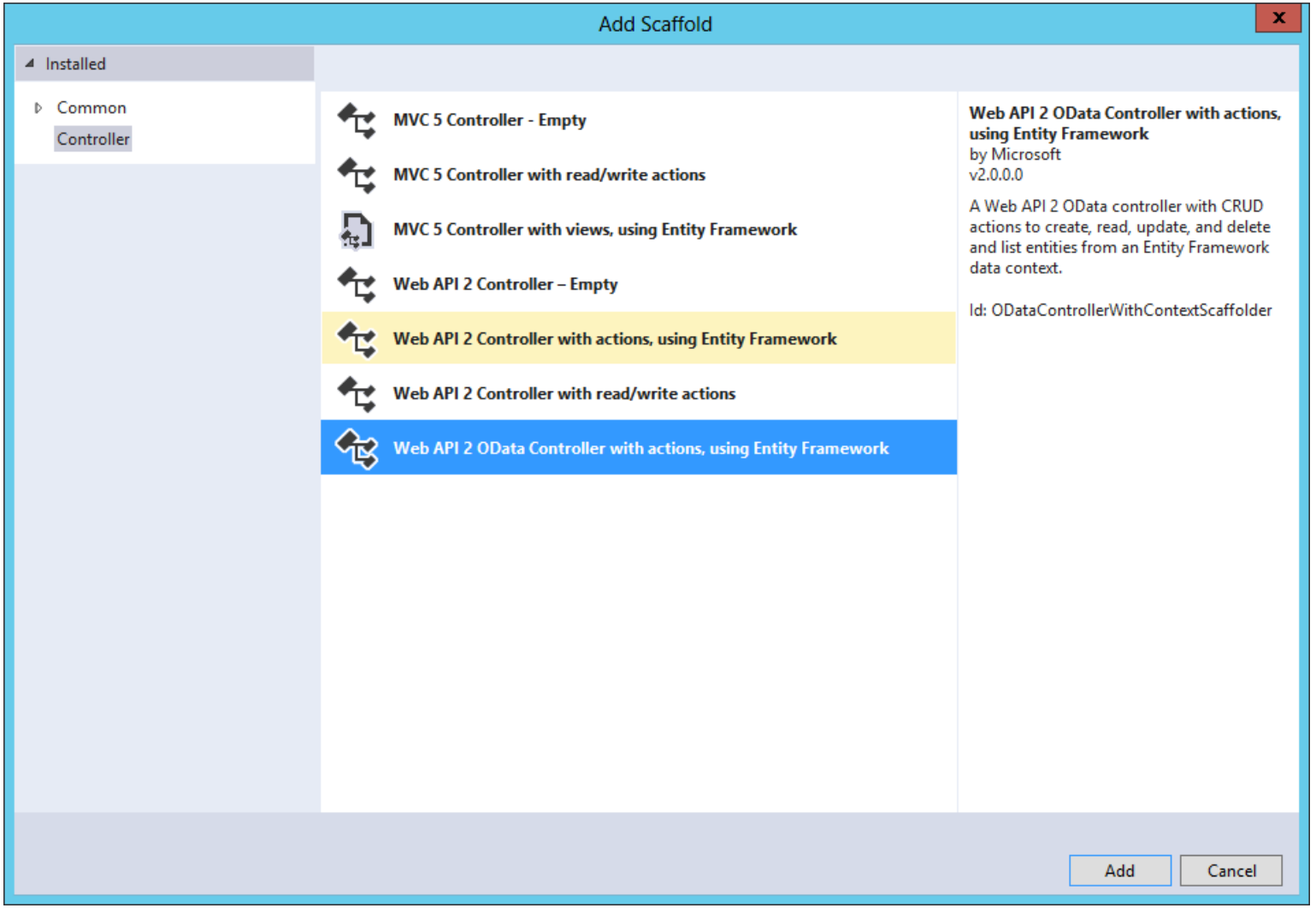
Routing

- Routes are controlled through maps

```
ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
builder.EntitySet<Contact>("Contacts");
builder.EntitySet<Company>("Companies");
config.Routes.MapODataRoute("odata", "odata", builder.GetEdmModel());
```

- Router makes decisions if information is missing
- By default methods are mapped to HTTP verbs

Adding an OData Controller




DEMO
Creating and Testing an OData Service

Agenda

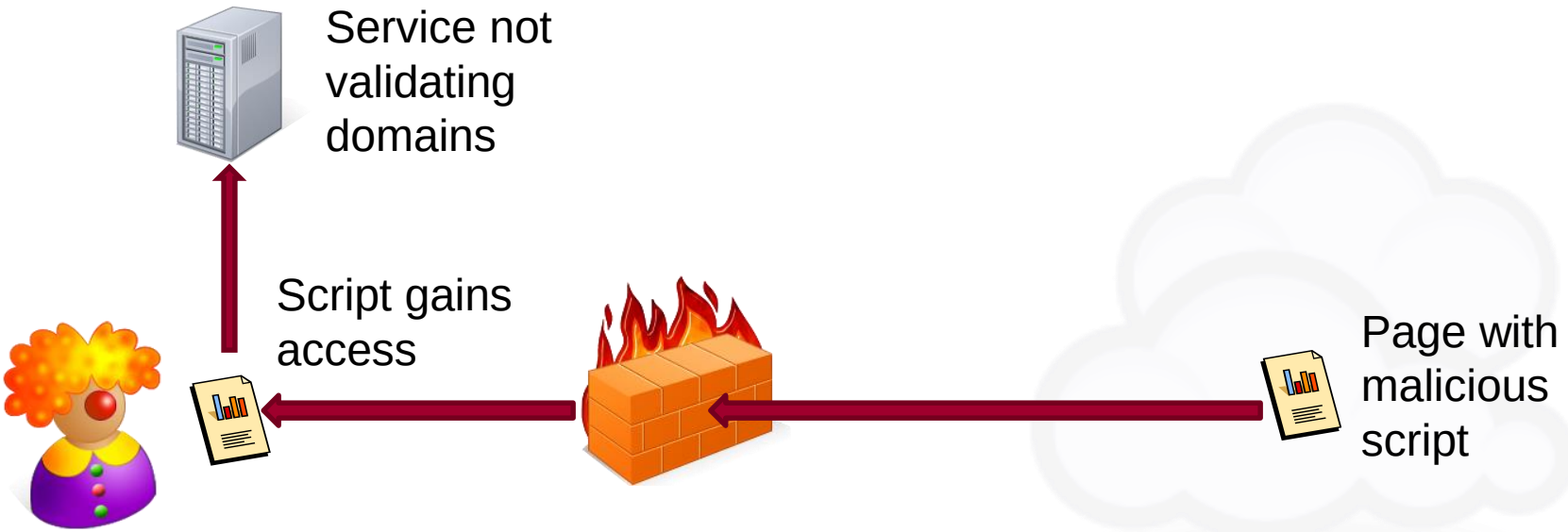
- ✓ Introducing WebAPI
- ✓ Calling API Controllers from MVC Apps
- ✓ Creating a RESTful Service
- ✓ Creating an OData Service
- ▪ Using Cross-Origin Resource Sharing

Cross-Origin Resource Sharing

- Allows JavaScript to make a call across domains
- Superior to JSONP, which only supports GET
- Supported in current versions of all major browsers
- Browser and resource exchange headers
  - Origin header from browser contains origin requesting
  - Access-Control-Allow-Origin header returned from resource if call is allowed
- Enabling in WebAPI2
  - Install Microsoft ASP.NET WebAPI2 CORS NuGet Package
  - Enable CORS in WebApiConfig
  - Use `[EnableCors]` attribute in controllers

Security Considerations

- Secure Sockets Layer – always!
- Always validate calling domain
  - Allowing all domains can open network to attack




DEMO
Cross-Origin Resource Sharing

Summary

- ✓ Introducing Web API
- ✓ Calling API Controllers from MVC Apps
- ✓ Creating a RESTful Service
- ✓ Creating an OData Service
- ✓ Using Cross-Origin Resource Sharing