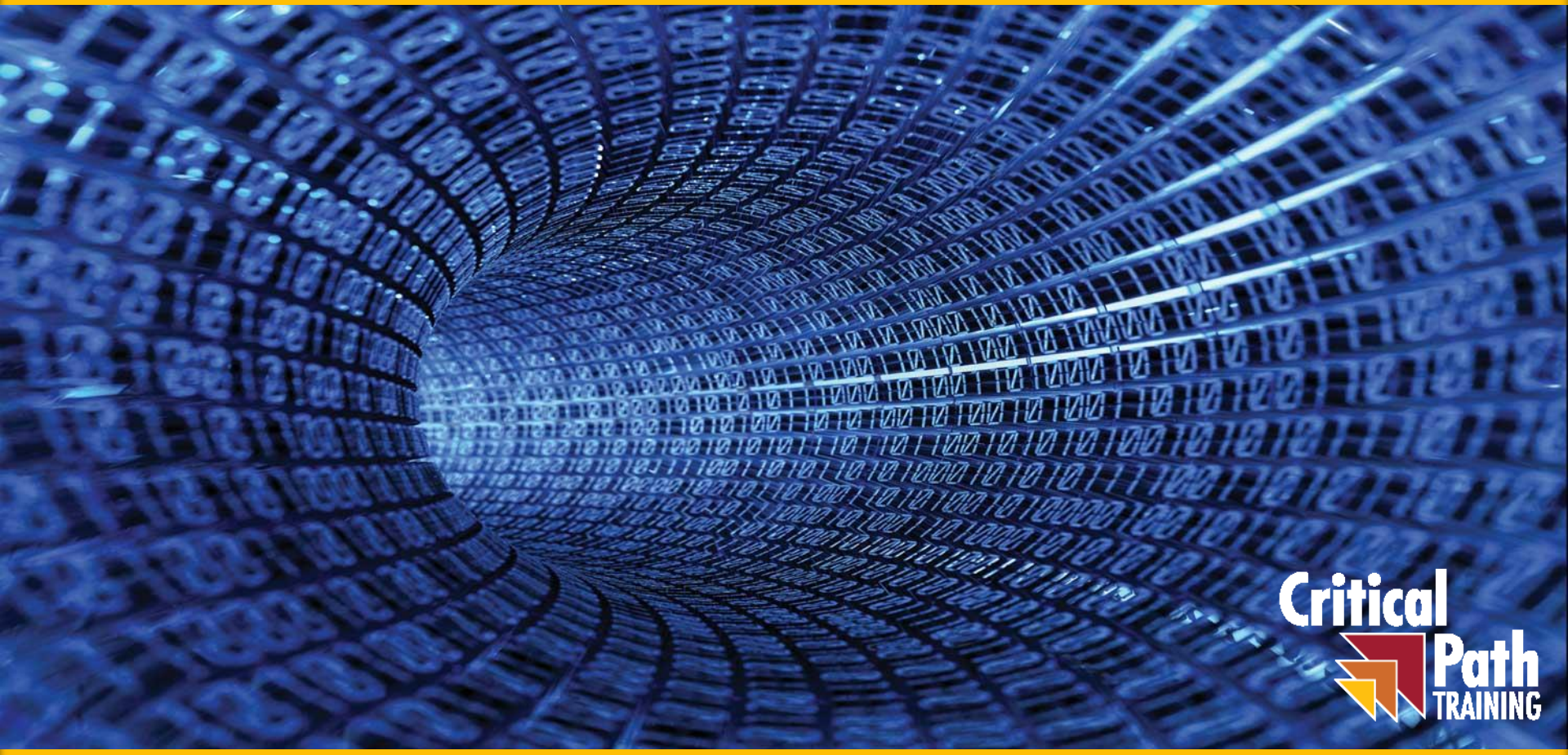
Programming the Power BI Service API



Agenda

- Power BI Service API Overview
- Creating App Workspaces and Workspace Associations
- Retrieving Data about Datasets, Reports and Dashboards
- Publishing PBIX Project Files
- Patching Datasource Credentials & Refreshing Datasets
- Cloning Power BI Content across Workspaces
- Creating Realtime Dashboards

What Is the Power BI Service API?

- What is the Power BI Service API?
  - API built on OAuth2, OpenID Connect, REST and ODATA
  - API secured by Azure Active Directory (AAD)
  - API to program with workspaces, datasets, reports & dashboards
  - API also often called “Power BI REST API”

- What can you do with the Power BI Service API?
  - Publish PBIX project files
  - Update connection details and datasource credentials
  - Create workspaces and clone content across workspaces
  - Embed Power BI reports and dashboards tiles in web pages
  - Create streaming datasets in order to build real-time dashboards

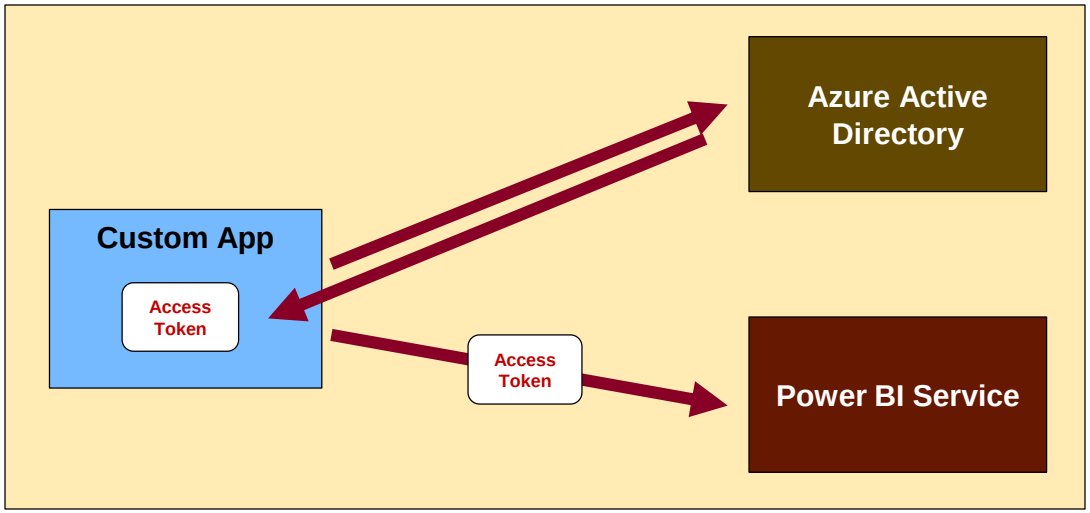
Getting Started

- What you need to get started?
  - Visual Studio 2017 or Visual Studio 2015
  - Organizational account in an Azure AD tenancy
  - License for Power BI Pro
  - Access to Azure portal to create Azure AD applications

- Azure subscription not required!
  - Azure portal used to create Azure AD application
  - Azure subscription helpful to create Azure resources

Authenticating with Azure AD

- User must be authenticated against Azure AD
  - User authentication used to obtain access token
  - Can be accomplished with the Azure AD Authentication Library
  - Access token pass to Power BI Service API in call REST calls



Access Token Acquisition (Native Client)

- With interactive login

```
static string aadAuthorizationEndpoint = "https://login.windows.net/common/oauth2/authorize";
static string resourceUriPowerBi = "https://analysis.windows.net/powerbi/api";
static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

public const string clientId = "315e87eb-a6a0-4886-9b20-9f7ecdaca888";
public const string redirectUrl = "https://localhost/app1234";

static string GetAccessToken() {

  // create new authentication context
  var authenticationContext = new AuthenticationContext(aadAuthorizationEndpoint);

  // use authentication context to trigger user sign-in and return access token
  var userAuthnResult = authenticationContext.AcquireTokenAsync(resourceUriPowerBi,
                                                                 clientId,
                                                                 new Uri(redirectUrl),
                                                                 new PlatformParameters(PromptBehavior.Auto)).Result;


  // return access token to caller
  return userAuthnResult.AccessToken;

}
```

- With User Password Credential flow (non-interactive)

```
string userName = "tedp@sharepointconfessions.onMicrosoft.com";
string userPassword = "Dublin@1234";

UserPasswordCredential creds = new UserPasswordCredential(userName, userPassword);
var userAuthnResult = authenticationContext.AcquireTokenAsync(PowerBiServiceResourceUri,
                                                              ClientID,
                                                              creds).Result;


// cache access token in AccessToken field
AccessToken = userAuthnResult.AccessToken;
```

Access Token Acquisition (web app)

```
private static string aadInstance = "https://login.microsoftonline.com/";
private static string resourceUrlPowerBi = "https://analysis.windows.net/powerbi/api";
private static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

private static string clientId = ConfigurationManager.AppSettings["client-id"];
private static string clientSecret = ConfigurationManager.AppSettings["client-secret"];
private static string redirectUrl = ConfigurationManager.AppSettings["reply-url"];

private static async Task<string> GetAccessTokenAsync() {

  // determine authorization URL for current tenant
  string tenantID = ClaimsPrincipal.Current.FindFirst("http://schemas.microsoft.com/identity/claims/tenantid").Value;
  string tenantAuthority = aadInstance + tenantID;

  // create ADAL cache object
  ApplicationDbContext db = new ApplicationDbContext();
  string signedInUserID = ClaimsPrincipal.Current.FindFirst(ClaimTypes.NameIdentifier).Value;
  ADALTokenCache userTokenCache = new ADALTokenCache(signedInUserID);

  // create authentication context
  AuthenticationContext authenticationContext = new AuthenticationContext(tenantAuthority, userTokenCache);

  // create client credential object using client ID and client Secret"];
  ClientCredential clientCredential = new ClientCredential(clientId, clientSecret);

  // create user identifier object for logged on user
  string objectIdentifierId = "http://schemas.microsoft.com/identity/claims/objectidentifier";
  string userObjectID = ClaimsPrincipal.Current.FindFirst(objectIdentifierId).Value;
  UserIdentifier userIdentifier = new UserIdentifier(userObjectID, UserIdentifierType.UniqueId);

  // get access token for Power BI Service API from AAD
  AuthenticationResult authenticationResult =
    await authenticationContext.AcquireTokenSilentAsync(
        resourceUrlPowerBi,
        clientCredential,
        userIdentifier);

  // return access token back to user
  return authenticationResult.AccessToken;

}
```

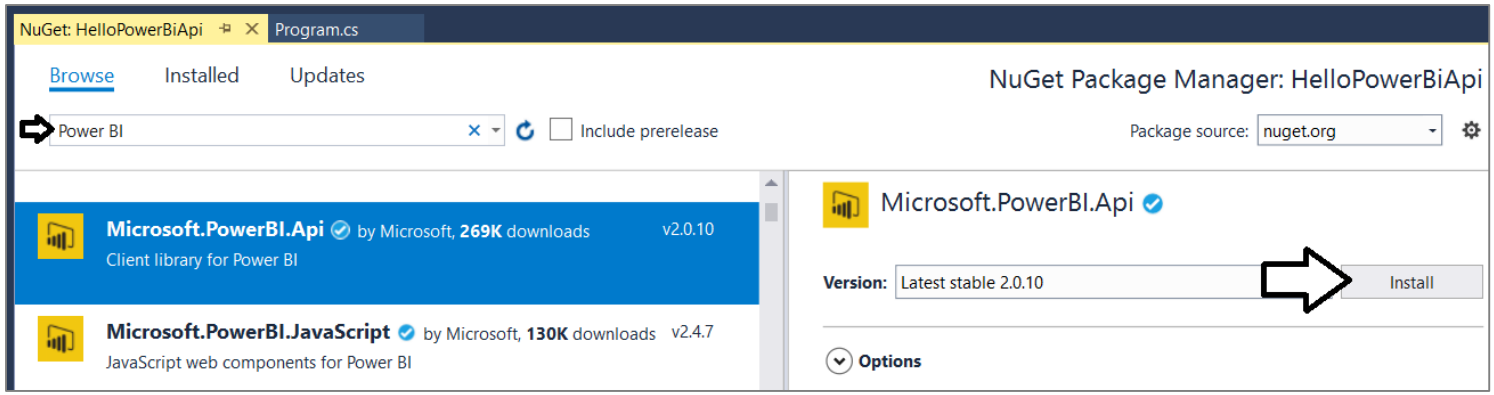
REST Calls to the Power BI Service API

```
static string ExecuteGetRequest(string restUrl) {
  HttpClient client = new HttpClient();
  HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Get, restUrl);
  request.Headers.Add("Authorization", "Bearer " + GetAccessToken());
  request.Headers.Add("Accept", "application/json;odata.metadata=minimal");
  HttpResponseMessage response = client.SendAsync(request).Result;
  if (response.StatusCode != HttpStatusCode.OK) {
    throw new ApplicationException("Error occured calling the Power BI Servide API");
  }
  return response.Content.ReadAsStringAsync().Result;
}

static void Main() {
  // get report data from app workspace
  string restUrl = "https://api.powerbi.com/v1.0/myorg/groups/" + appWorkspaceId + "/reports/";
  var json = ExecuteGetRequest(restUrl);
  ReportCollection reports = JsonConvert.DeserializeObject<ReportCollection>(json);
  foreach (Report report in reports.value) {
    Console.WriteLine("Report Name: " + report.name);
    Console.WriteLine();
  }
}
```

```
public class Report {
  public string id { get; set; }
  public string name { get; set; }
  public string webUrl { get; set; }
  public string embedUrl { get; set; }
  public bool isOwnedByMe { get; set; }
  public string datasetId { get; set; }
}

public class ReportCollection {
  public List<Report> value { get; set; }
}
```

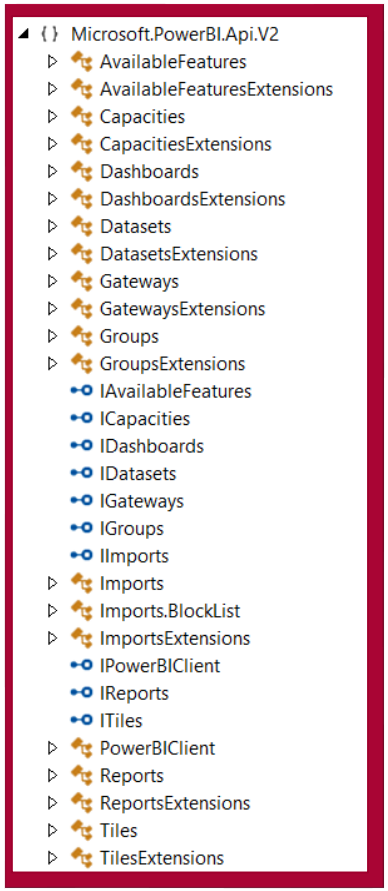
Power BI Service SDK

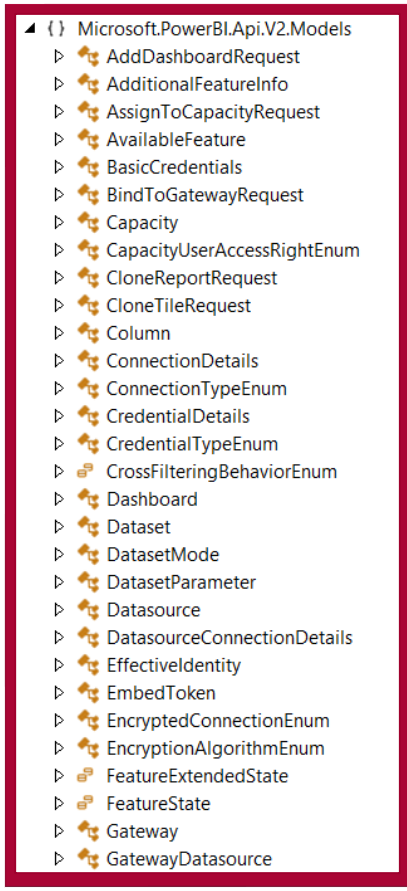
- Added as a NuGet package

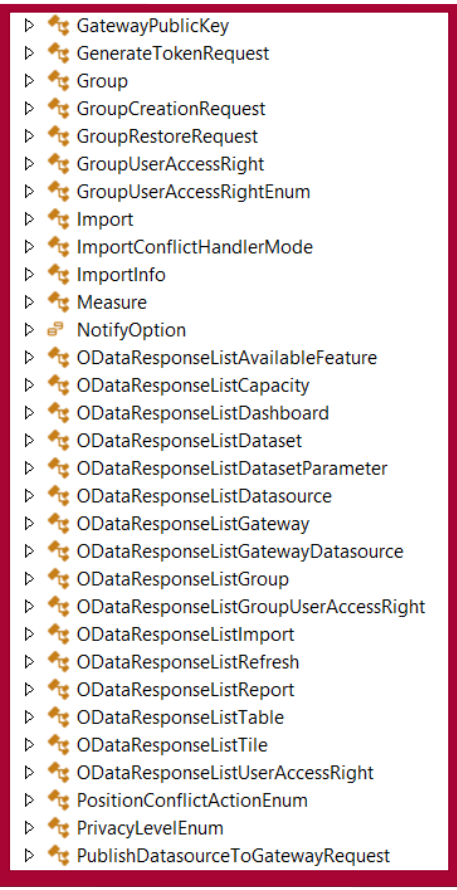


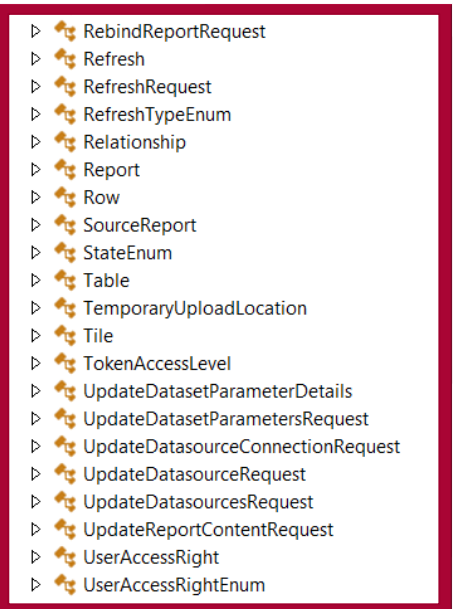
The Power BI SDK Classes

- SDK provides object model of classes









Initializing an Instance of PowerBIClient

- PowerBIClient object serves as top-level object
  - Used to execute calls against Power BI Service
  - Initialized with function to retrieve AAD access token

```
static string GetAccessToken() ...

static PowerBIClient GetPowerBiClient() {
  var tokenCredentials = new TokenCredentials(GetAccessToken(), "Bearer");
  return new PowerBIClient(new Uri(urlPowerBiRestApiRoot), tokenCredentials);
}

static void Main() {
  PowerBIClient pbiClient = GetPowerBiClient();
  var reports = pbiClient.Reports.GetReports().Value;
  foreach (var report in reports) {
    Console.WriteLine(report.Name);
  }
}
```

Enumerating Collections with PowerBiClient

```
static void DisplayAppWorkspaceAssets() {

  PowerBIClient pbiClient = GetPowerBiClient();

  Console.WriteLine("Listing assets in app workspace: " + appWorkspaceId);

  Console.WriteLine("Datasets:");
  var datasets = pbiClient.Datasets.GetDatasetsInGroup(appWorkspaceId).Value;
  foreach (var dataset in datasets) {
    Console.WriteLine(" - " + dataset.Name + " [" + dataset.Id + "]");
  }

  Console.WriteLine();
  Console.WriteLine("Reports:");
  var reports = pbiClient.Reports.GetReportsInGroup(appWorkspaceId).Value;
  foreach (var report in reports) {
    Console.WriteLine(" - " + report.Name + " [" + report.Id + "]");
  }

  Console.WriteLine();
  Console.WriteLine("Dashboards:");
  var dashboards = pbiClient.Dashboards.GetDashboardsInGroup(appWorkspaceId).Value;
  foreach (var dashboard in dashboards) {
    Console.WriteLine(" - " + dashboard.DisplayName + " [" + dashboard.Id + "]");
  }
}
```

Creating App Workspaces

```
public static async Task<Group> CreateWorkspacesAsync(string WorkspaceName) {

  PowerBIClient pbiClient = GetPowerBiClient();
  GroupCreationRequest createRequest = new GroupCreationRequest(WorkspaceName);
  var workspace = await pbiClient.Groups.CreateGroupAsync(createRequest);

  var secondaryAdmin = "pbiemasteruser@sharepointconfessions.onmicrosoft.com";
  var userRights = new GroupUserAccessRight("Admin", secondaryAdmin);
  await pbiClient.Groups.AddGroupUserAsync(workspace.Id, userRights);

  return workspace;

}
```

Importing a PBIX File

```
public static async Task UploadPBIX(string WorkspaceId, string pbixName, string importName, bool updateSqlCredentials = false) {

  string PbixFilePath = HttpContext.Current.Server.MapPath("/PBIX/" + pbixName);
  PowerBIClient pbiClient = GetPowerBiClient();
  FileStream stream = new FileStream(PbixFilePath, FileMode.Open, FileAccess.Read);
  var import = await pbiClient.Imports.PostImportWithFileAsyncInGroup(WorkspaceId, stream, importName);

  if (updateSqlCredentials) {
    await PatchSqlDatasourceCredentials(WorkspaceId, importName);
  }

  return;
}
```

Patching Datasource Credentials

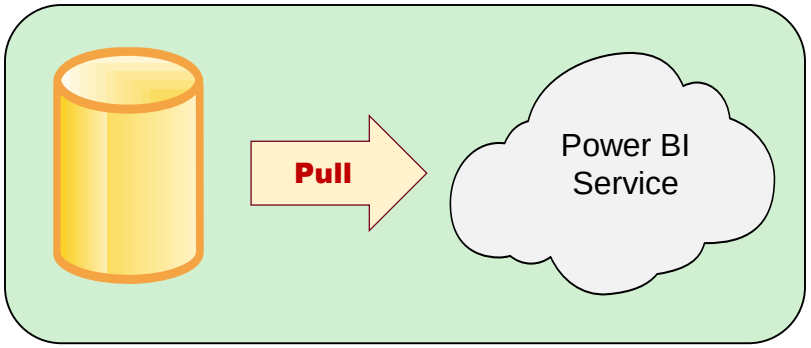
```
public static async Task PatchSqlDatasourceCredentials(string WorkspaceId, string importName) {
  PowerBIClient pbiClient = GetPowerBiClient();
  var datasets = (await pbiClient.Datasets.GetDatasetsInGroupAsync(WorkspaceId)).Value;
  foreach (var dataset in datasets) {
    if (importName.Equals(dataset.Name)) {
      string datasetId = dataset.Id;
      var datasources = (await pbiClient.Datasets.GetDatasourcesInGroupAsync(WorkspaceId, datasetId)).Value;
      foreach (var datasource in datasources) {
        if (datasource.DatasourceType == "SQL") {
          var datasourceId = datasource.DatasourceId;
          var gatewayId = datasource.GatewayId;
          // create credentials for Azure SQL database log in
          Creds.BasicCredentials creds = new Creds.BasicCredentials("CptStudent", "pass@word1");
          CredentialDetails details = new CredentialDetails(creds);
          UpdateDatasourceRequest req = new UpdateDatasourceRequest(details);
          // Update credentials through gateway
          await pbiClient.Gateways.UpdateDatasourceAsync(gatewayId, datasourceId, details);
        }
      }
    }
  }
  return;
}
```

Exporting/Importing PBIX Files

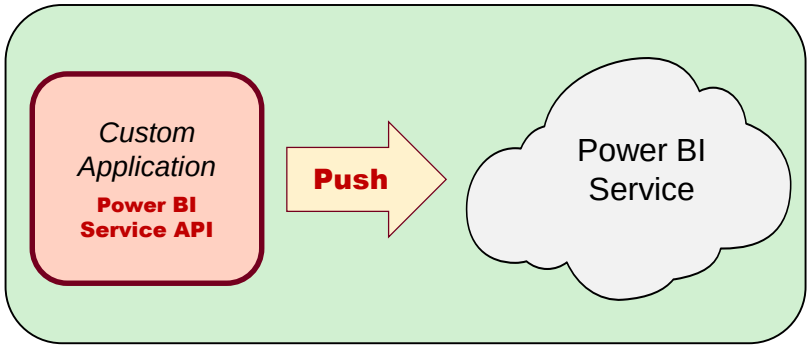
```
var reports = pbiClient.Reports.GetReportsInGroup(sourceAppWorkspaceId).Value;

string downloadPath = @"C:\Student\downloads\";
// create download folder if it doesn't exist
if (!Directory.Exists(downloadPath)) {
  Directory.CreateDirectory(downloadPath);
}

foreach (var report in reports) {

  var reportStream = pbiClient.Reports.ExportReportInGroup(sourceAppWorkspaceId, report.Id);
  string filePath = downloadPath + report.Name + ".pbix";
  Console.WriteLine("Downloading PBIX file for " + report.Name + "to " + filePath);
  FileStream stream1 = new FileStream(filePath, FileMode.Create, FileAccess.ReadWrite);
  reportStream.CopyToAsync(stream1).Wait();
  reportStream.Close();
  stream1.Close();
  stream1.Dispose();

  FileStream stream = new FileStream(filePath, FileMode.Open, FileAccess.Read);
  Console.WriteLine("Publishing " + filePath + " to " + targetAppWorkpaceName);
  var import = pbiClient.Imports.PostImportWithFileInGroup(targetAppWorkspaceId, stream, report.Name);

  Console.WriteLine("Deleing file " + filePath);
  stream.Close();
  stream.Dispose();
  File.Delete(filePath);

  Console.WriteLine();
}

Console.WriteLine("Export/Import process completed");
```

Pull Datasets versus Real-time Datasets

- Pull Datasets
  - Imported Datasets
  - DirectQuery Datasets
  - Live Connect Datasets

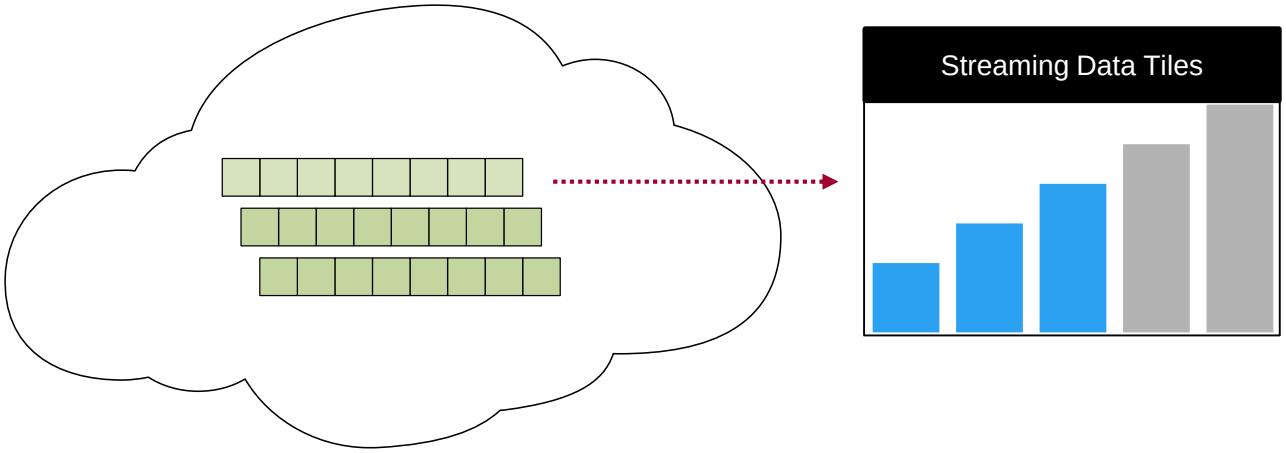


- Real-time Datasets
  - Streaming Datasets
  - Push Datasets
  - Hybrid Datasets



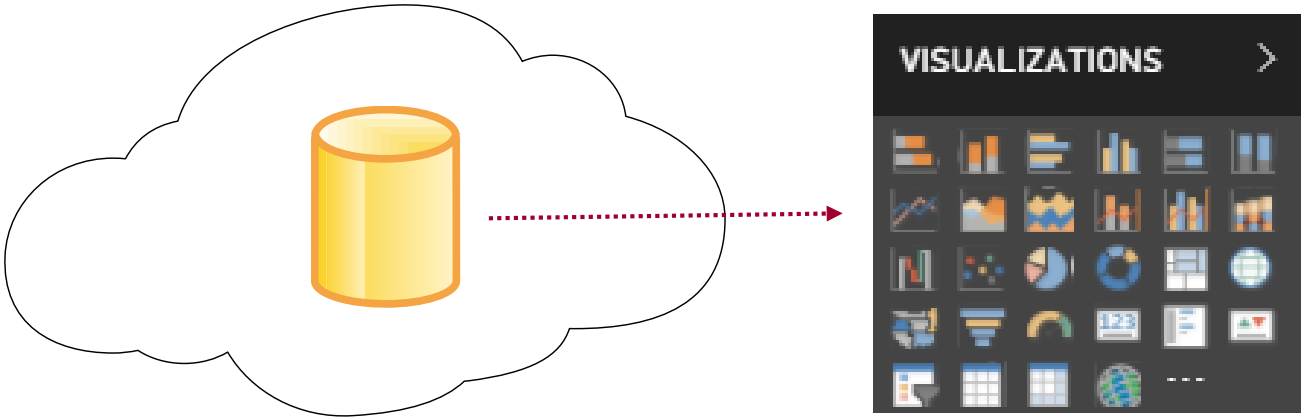
Streaming Datasets

- Data stored in cloud-based cache – not persisted in DB
- Restricted to single table - no rich data modeling
- Not supported by standard Power BI report designer
- Dashboard created using specialized streaming data tiles
- No support for DAX, aggregation or filtering



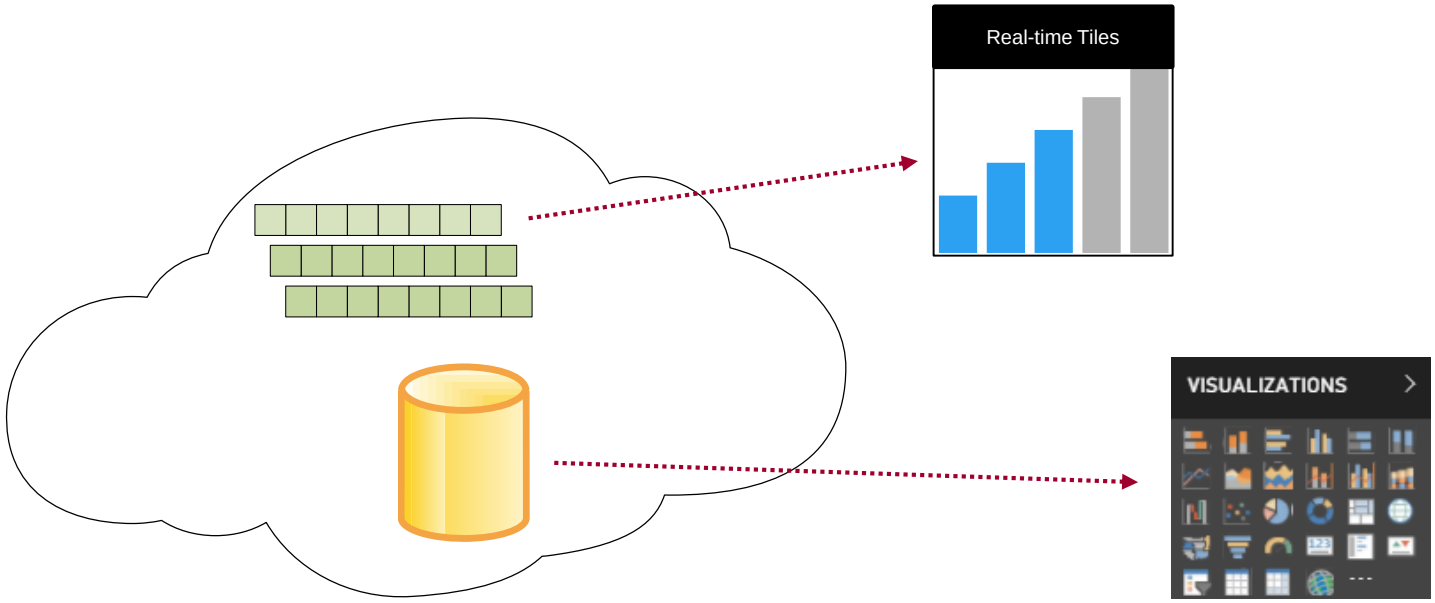
Push Datasets

- Data stored in Azure SQL DB – not in cache
- Supports multiple tables and table relationships
- Supported by standard Power BI report designer
- Supports DAX, measures, aggregation & filtering



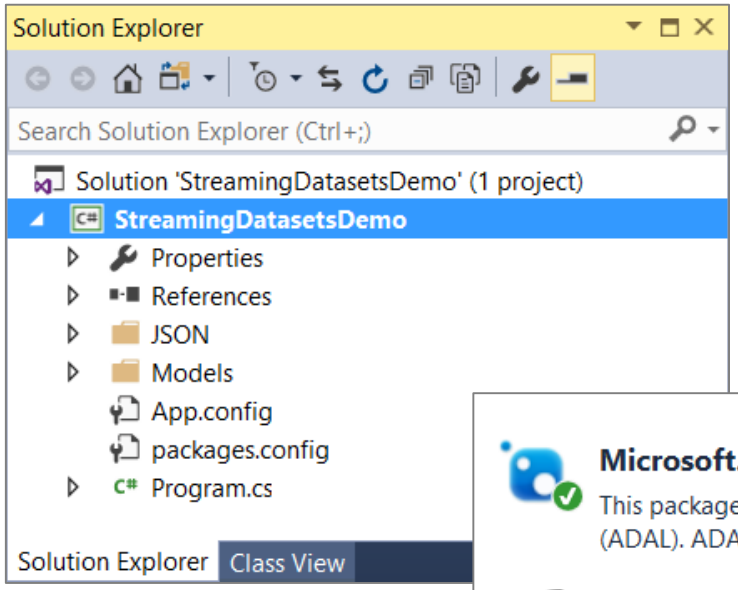
Hybrid Datasets

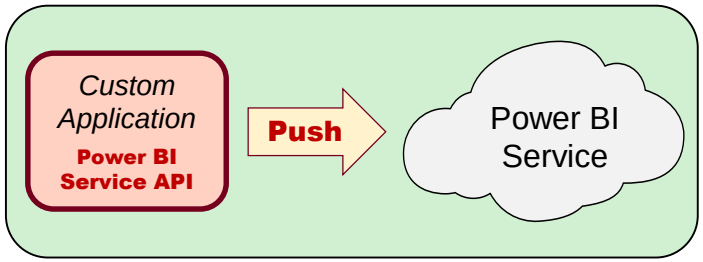
- Data stored in cloud-based cache AND in Azure SQL DB
- Restricted to single table and no rich data modeling
- Supported by streaming data tiles
- Supported by Power BI report designer

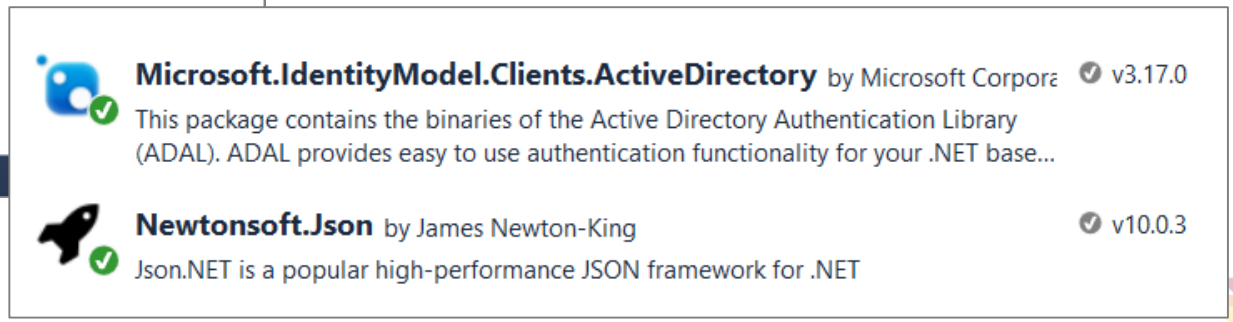


The StreamingDatasetsDemo Project

- Console application project in Visual Studio 2017
  - Installed package for Azure AD Authentication library
  - Installed package to serialize .NET objects to JSON







Creating a Streaming Dataset

- Streaming dataset created using JSON schema definition
  - Streaming dataset limited to a single table
  - Columns defined using name and datatype
  - No support for any other column properties (e.g. formatting)

DemoStreamingDataset.json

```
{
  "name": "TemperatureReadings",
  "defaultMode": "Streaming",
  "tables": [
    { "name": "TemperatureReadings",
      "columns": [
        { "name": "Run", "dataType": "string" },
        { "name": "Time", "dataType": "Datetime" },
        { "name": "TimeWindow", "dataType": "string" },
        { "name": "TargetTemperature", "dataType": "Double" },
        { "name": "MinTemperature", "dataType": "Double" },
        { "name": "MaxTemperature", "dataType": "Double" },
        { "name": "BatchA", "dataType": "Double" },
        { "name": "BatchB", "dataType": "Double" },
        { "name": "BatchC", "dataType": "Double" }
      ]
    }
  ]
}
```

Creating a Custom Dataset

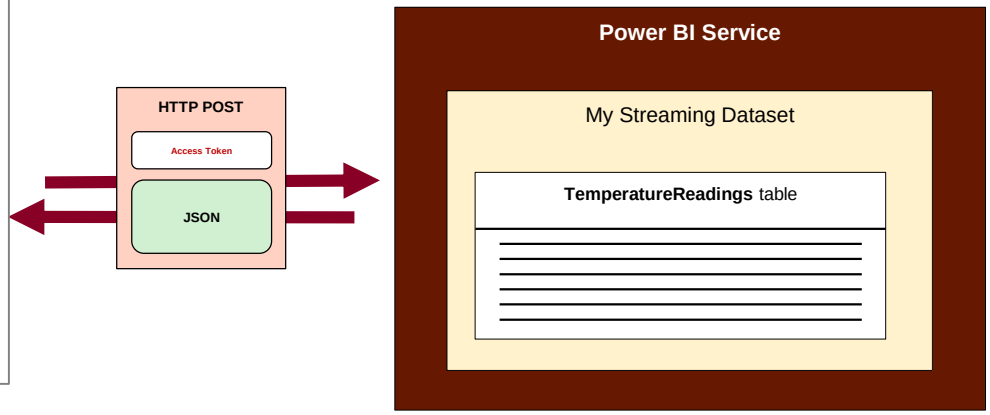
- Dataset created by executing HTTP POST operation
  - One-time operation done as application begins running

```
// prepare call to create new dataset
string restUrlDatasets = ProgramGlobalConstants.PowerBiServiceRootUrl + "datasets";
string jsonNewDataset = Properties.Resources.NewDataset_json;
// execute REST call to create new dataset
string json = ExecutePostRequest(restUrlDatasets, jsonNewDataset);
// retrieve Guid to track dataset ID
Dataset dataset = JsonConvert.DeserializeObject<Dataset>(json);
CustomDatasetId = dataset.id;
```

1
2
3

DemoStreamingDataset.json

```
{
  "name": "TemperatureReadings",
  "defaultMode": "Streaming",
  "tables": [
    { "name": "TemperatureReadings",
      "columns": [
        { "name": "Run", "dataType": "string" },
        { "name": "Time", "dataType": "Datetime" },
        { "name": "TimeWindow", "dataType": "string" },
        { "name": "TargetTemperature", "dataType": "Double" },
        { "name": "MinTemperature", "dataType": "Double" },
        { "name": "MaxTemperature", "dataType": "Double" },
        { "name": "BatchA", "dataType": "Double" },
        { "name": "BatchB", "dataType": "Double" },
        { "name": "BatchC", "dataType": "Double" }
      ]
    }
  ]
}
```



Adding Rows by Converting C# to JSON

```
TemperatureReadingsRow row = new TemperatureReadingsRow {
  Run = RunName,
  Time = DateTime.Now,
  TimeWindow = currentTimeWindow,
  TargetTemperature = 212,
  MinTemperature = 100,
  MaxTemperature = 250,
  BatchA = temperatureBatchA,
  BatchB = temperatureBatchB,
  BatchC = temperatureBatchC,
};

TemperatureReadingsRow[] rows = { row };
TemperatureReadingsRows temperatureReadingsRows = new TemperatureReadingsRows { rows = rows };
string jsonNewRows = JsonConvert.SerializeObject(temperatureReadingsRows);
string restUrlTargetTableRows = string.Format("{0}/{1}/tables/TemperatureReadings/rows", restUrlDatasets, DatasetId);
string jsonResultAddExpenseRows = ExecutePostRequest(restUrlTargetTableRows, jsonNewRows);
```

```
public class TemperatureReadingsRow {
  public string Run { get; set; }
  public DateTime Time { get; set; }
  public string TimeWindow { get; set; }
  public double TargetTemperature { get; set; }
  public double MinTemperature { get; set; }
  public double MaxTemperature { get; set; }
  public double BatchA { get; set; }
  public double BatchB { get; set; }
  public double BatchC { get; set; }
}

class TemperatureReadingsRows {
  public TemperatureReadingsRow[] rows { get; set; }
}
```

```
{
  "rows": [
    {
      "Run": "Run 06",
      "Time": "2017-10-05T22:43:40.364569-04:00",
      "TimeWindow": "22:43:30",
      "TargetTemperature": 212.0,
      "MinTemperature": 100.0,
      "MaxTemperature": 250.0,
      "BatchA": 152.73999999999995,
      "BatchB": 152.78,
      "BatchC": 152.25
    }
  ]
}
```

Real-time Dataset Matrix

Feature	Streaming	Hybrid	Push
Updates in real-time	Yes	Yes	Yes
Smooth animations	Yes	Yes	No
Backed by Azure SQL DB	No	Yes	Yes
Report Designer Support	No	Yes	Yes
Allow Rich Data Modeling	No	No	Yes
Ingestion Rate	5 request/sec 15KB/request		1 request/second 16MB/request

Summary

- ✓ Power BI Service API Overview
- ✓ Creating App Workspaces and Workspace Associations
- ✓ Retrieving Data about Datasets, Reports and Dashboards
- ✓ Publishing PBIX Project Files
- ✓ Patching Datasource Credentials & Refreshing Datasets
- ✓ Cloning Power BI Content across Workspaces
- ✓ Creating Realtime Dashboards