# Get started with the C# SDK using Azure Functions

## Overview

It's easy to get started programming against the Common Data Service (CDS) using Azure Functions. This topic walks you through getting a CDS Azure Function up and running.

There are four key steps:

1. **CDS database acquisition**. The Common Data Service is currently only available through **PowerApps**. You need to get access to a PowerApps environment and ensure it contains a CDS database. This allows you to configure the SDK to access that database.
2. **Application registration**. To give your Azure function access to the Common Data Service, you need to register a few applications in **Azure Active Directory**. This allows you to establish an identity for your applications and specify the permission levels they needs in order to access the CDS APIs.
3. **Azure Function creation, configuration and programming**. You can skip most of this step if you choose to start from the Azure Function project we provide you, and publish it to Azure. If you choose to start from scratch however, you can create and configure your [Azure Functions](https://azure.microsoft.com/en-us/services/functions/) from the web portal. There you will be able to use the appropriate Functions template, and configure the CDS C# SDK's NuGet references, authentication, and target environment.
4. **Console client application creation and configuration**. You can then run and debug your Azure Function by running the client console app and making HTTP calls to the Function.

In addition, you can use this function from a PowerApps application. There are two key steps:

1. **Custom API creation and configuration**
2. **Programming and running your PowerApps**

# CDS database acquisition

The Common Data Service is currently only available through PowerApps. You need to get access to a PowerApps environment and ensure it contains a CDS database. This allows you to configure the SDK to access that database.

## Prerequisites

1. If you have already signed up for PowerApps, you can go to [PowerApps](https://powerapps.microsoft.com) and sign in. If you have not signed up yet, you can follow instructions to [sign up](https://powerapps.microsoft.com/en-us/tutorials/signup-for-powerapps/).
2. Ensure you have admin access to an environment containing a CDS database, by going to the [PowerApps](https://powerapps.microsoft.com) portal, clicking on the **gear icon** located on top right of the screen, then clicking on **Admin center**. If you do not have admin access to any environments containing a database, follow these instructions to [create a database](https://powerapps.microsoft.com/en-us/tutorials/create-database/).

## Getting the environment ID

After acquiring an environment that contains a CDS database, you can use that environment's identifier to configure your CDS SDK application. The **environment ID** can be found as part of the URI you are using to access the environment. Record this value as it will be used in the upcoming configuration step. An example of such a URI and environment Id are as follows:

URI: https://web.powerapps.com/environments/d1ec10fa-74d5-44e5-b0f7-e448e3ca7509/home  
Environment ID: d1ec10fa-74d5-44e5-b0f7-e448e3ca7509

# Application registration

To give your Azure Function access to the Common Data Service, you need to register a **Web app / API** applications in **Azure Active Directory**. This allows you to establish an identity for your applications and specify the permission level it needs to access the CDS APIs. You will also need to register the applications calling the Azure function. In this guide, we will use a simple console application to call into the Azure function, for this step we will require a **Native application** registration. Later, as advanced steps, we will configure a PowerApps Custom API to call the Function, which will require registering another **Web app / API**. All these apps must be configured in Azure AD with the correct **Required permissions** and **known client applications**, for the end to end flow to work correctly.

## Prerequisites

If you have already signed up for an Azure subscription, go to [Azure portal](https://portal.azure.com) and ensure you can create an application registration under Azure Active Directory. If you cannot, go to the [Azure](https://azure.microsoft.com) site and sign up for a free trial.

1. In [Azure portal](https://portal.azure.com) go to **Azure Active Directory**
2. Click on **Properties** and copy the value of **Directory ID** and record it as config variable **AAD tenant** for upcoming steps. Alternatively, you can use the domain name from your AAD login email.

## Azure function application registration

Follow these steps to register and configure your Azure Function in Azure AD:

1. Go back to **Azure Active Directory** then click on **App registrations**.
2. Create the Azure function application resource that we will use to call into CDS directly:
   1. Click on **Add** to see a Create pane.
   2. Enter a **Name** for your Azure function application.
   3. Select **Web app / API** as application type.
   4. Add a **Sign-on URL**, it could be any valid URI string. For example: <http://localhost>.
   5. Click on **Create**.
3. Open the registered app:
   1. Search for your newly registered app by name.
   2. Click on it after finding it in the list of applications.
   3. Record configuration value **Application ID** for upcoming steps.
4. Get the application secret:
   1. Click on **Keys** to open a new pane.
   2. Add a new **Key description** like "key", set the **Duration** to "Never expires" and click **Save**.
   3. Record configuration value for the **Application secret** by copying and pasting contents of the "Value" cell.
5. Setup **Required permissions** for connecting to CDS:
   1. Click on **Required permissions** to open a new pane.
   2. Click on **Add**.
   3. Navigate to **Select an API**.
   4. Search for and choose **PowerApps Runtime Service**, then click **Select**. If you cannot find this service refer to the **Troubleshooting** section under **Required permissions service not found**.
   5. Check all entries under **Delegated permissions**, then click **Select**. Note: you may have to check and uncheck the box multiple times before the Select button is enabled.
   6. Click on **Done** to finalize setting up permissions for this service.
   7. Repeat the 3 steps above for **Windows Azure Service Management API**.
6. Get the Function application's resource ID:
   1. Open the application's JSON manifest directly, by clicking on **Manifest** on top of the registered app pane.
   2. Record the config value **AAD function resource ID** from any entry under identifierUris. Following is an example: https://[tenant\_name].onmicrosoft.com/4598e084-55a7-4d57-88ae-59902d8e3a52.
7. **Note** - Configuring **known client applications** must be performed after registering the other application(s). The step is needed for seamless propagation of required permissions to clients. After completing all registrations go to section **Add known applications to Azure Function app**.

## Client application registration

Follow these steps to register and configure your client application in Azure AD:

1. Go back to **Azure Acitve Directory** then click on **App registrations**.
2. Create the application resource that we will be used when prompting you to login:
   1. Click on **Add** to see a Create pane.
   2. Enter a **Name** for your client application.
   3. Select **Native** as application type.
   4. Add a **Redirect URI**, it could be any valid URI string. For example: <http://localhost>.
   5. Record configuration value **Redirect URI** for upcoming steps.
   6. Click on **Create**.
3. Open the Registered app:
   1. Search for your newly registered app by name.
   2. Click on it after finding it in the list of applications.
   3. Record configuration value **Application ID** for upcoming steps.
4. Setup **Required permissions** for connecting to the Azure Function:
   1. Click on **Required permissions** to open a new pane.
   2. Click on **Add**.
   3. Navigate to **Select an API**.
   4. Search for and select **Name** of the Azure Function web app created in previous step, then click **Select**.
   5. Check all boxes under **Delegated permissions** section, then click **Select**. Note: you may have to check and uncheck the box multiple times before the Select button is enabled.
   6. Click on **Done** to finalize setting up permissions for this service.

## Advanced - PowerApps Custom API application registration

You can skip this steps if you are not planning to use the Azure Function from PowerApps. If you are interested in using PowerApps, follow these steps to register and configure your PowerApps Custom API application in Azure AD:

1. Go back to **Azure Active Directory** then click on **App registrations**.
2. Create the application resource that we will be used during the PowerApps user login process:
   1. Click on **Add** to see a Create pane.
   2. Enter a **Name** for your registered client application.
   3. Select **Web app / API** as application type.
   4. Set **Sign on URL** to https://msmanaged-na.consent.azure-apim.net/redirect, and record it for upcoming steps.
   5. Click on **Create**.
3. Open the **Registered app**:
   1. Search for your newly registered app by name
   2. Click on it after finding it in the list of applications.
   3. Record configuration value **Application ID** for upcoming steps.
4. Setup **Required permissions** for connecting to the Azure Function:
   1. Click on **Required permissions** to open a new pane.
   2. Click on **Add**.
   3. Navigate to **Select an API**.  
      Search for and choose **PowerApps Runtime Service**, then click **Select**.
   4. Search for and select **Name** of the Azure Function web app created in previous step, then click **Select**.
   5. Check all boxes under **Delegated permissions** section, then click **Select**.
   6. Click on **Done** to finalize setting up permissions for this service.

## Add known applications to Azure Function app

For seamless propagation of required permissions to clients, setup **known client applications**. This means that the first time a user logs into a client application, they will be asked to provide all required permissions needed to run this flow. If you skip this step, you may have to manually authorize use of CDS for each user.

1. Go back to **Azure Active Directory** then click on **App registrations**.
2. Open the registered app:
   1. Search for your Azure Function web app created in previous step.
   2. Click on it after finding it in the list of applications.
   3. Modify the application's JSON manifest directly, by clicking on **Manifest** on top of the registered app pane.
   4. Click on **Download** and save a backup of the manifest, then click **Edit** to go back.
   5. Get the application IDs for both your **Client application** and **PowerApps custom API application**.
   6. Add client application IDs as JSON string entries under the JSON array named knownClientApplications while maintaining validity of the manifest, then click **Save**. The manifest will look similar to the following:

"knownClientApplications": [  
 "f60bee48-2341-4528-a91c-ec97f3ded...",  
 "b3b3d65c-5cbf-4323-b32d-68fd40778..."  
 ],

# Azure Function creation, configuration and programming

You can skip most of this step if you choose to start from the Azure Function project we provide you, and publish it to Azure. If you choose to start from scratch however, you can create and configure your [Azure Functions](https://azure.microsoft.com/en-us/services/functions/) from the web portal. There you will be able to use the appropriate Functions template, and configure the CDS C# SDK's NuGet references, authentication, and target environment.

## Prerequisites

As with the previous step you need an Azure subscription, and access to [Azure portal](https://portal.azure.com). If you don't already have a subscription, go to the [Azure](https://azure.microsoft.com) site and sign up for a free trial.

Ensure you have the following configuration values from previous steps:

1. **AAD tenant**. This value identifies the tenant your database resides in.
2. **AAD application ID**. This value identifies the AAD web app you registered earlier.
3. **AAD application secret**. This value identifies secret of the AAD web app you registered earlier.
4. **PowerApps environment ID**. This value identifies the PowerApps environment that contains your target CDS database.

## Sample Azure Function project

For the private preview release, a project called SampleFunctionApplication will be included in the CDS SDK material. If you choose to go with this sample project, you will need the latest [Azure Functions tools for Visual Studio](https://aka.ms/azfunctiontools) installed on your computer.

1. Replace the brackets in **appsettings.json** with configuration values mentioned in the **prerequisites** section.
2. Right click on the Function project and click **Publish ...**
3. Select the publish target **Microsoft Azure App Service**.
4. Enter your credentials.
5. Click **New** and create a new app service in your existing subscription.
6. Create new **Resource Group**, **App Service Plan** or **Storage Account** as appropriate.
7. Skip to the **Console client app creation and configuration** section.
8. **Note** that after publishing, you may have to delete the published **project.lock.json** file as detailed in the troubleshooting section **Assembly load issue after publish to Azure**.

You can also follow the similar instructions as below to create and configure the Azure function in Visual Studio, then publish it as shown above.

## Azure Function creation and configuration

Assuming you already have an Azure subscription set up, create and open a new function app from the [Azure Portal](https://portal.azure.com/#create/Microsoft.FunctionApp):

1. Enter a unique **App name**.
2. Select your current **Subscription**.
3. Select an existing **Resource group** or provide a unique name for a new one.
4. Select the existing **Consumption Plan** as your hosting plan.
5. Select the **Location** of the function.
6. Select an existing **Storage account** or follow steps to create one.
7. Click **Create**.
8. Under **All resources** search for and select the app you created.

Create a new Azure function using templates:

1. Click on **New Function** under Functions from the left pane.
2. Select **GenericWebHook-CSharp** from the list of templates.
3. Name the new Function **UpdateProjectCategory**, and click **Create**.
4. **Note** the sample code window opened under the **Develop** section.
5. Record the configuration value **Function URL** for upcoming steps by clicking on **Get function URL** on the top right, and clicking copy.

Copy the following JSON snippet to the **project.json** file of the Function project. On the Azure Portal Functions experience, you can update this file through the following steps:

1. Go to the **View files** tab on the top right of the pane.
2. Create the file by clicking on **Add** and typeing in **project.json**.
3. Paste the JSON contents below into the edit window, then click **Save**.

{  
 "frameworks": {  
 "net46":{  
 "dependencies": {  
 "Castle.Core": "3.3.3",  
 "Google.Protobuf": "3.0.0",  
 "Microsoft.AspNet.WebApi.Client": "5.2.3",  
 "Microsoft.CommonDataService": "1.0.188-preview",  
 "Microsoft.IdentityModel.Clients.ActiveDirectory": "3.13.8",  
 "Microsoft.IdentityModel.Logging": "1.1.3",  
 "Microsoft.IdentityModel.Tokens": "5.1.3",  
 "Newtonsoft.Json": "9.0.1",  
 "System.IdentityModel.Tokens.Jwt": "5.1.3"  
 }  
 }  
 }  
}

From **View files** pane on the right side, open **run.csx**, and replace the using and #r statements with below:

using Microsoft.CommonDataService;  
using Microsoft.CommonDataService.CommonEntitySets;  
using Microsoft.CommonDataService.Configuration;  
using Microsoft.CommonDataService.ServiceClient.Security;  
using Microsoft.CommonDataService.ServiceClient.Security.Credentials;  
using System;  
using System.Collections.Generic;  
using System.Net;

Copy the following code snippet inside the run() method body of **run.csx**, replacing existing code.

log.Info($"C# HTTP trigger function processed a request. RequestUri={req.RequestUri}");  
  
// parse query parameter  
string name = req.GetQueryNameValuePairs()  
 .FirstOrDefault(q => string.Compare(q.Key, "name", true) == 0)  
 .Value;  
  
if(name == null)  
{  
 log.Info($"Name was not passed correctly.");  
 return req.CreateResponse(HttpStatusCode.BadRequest);  
}  
  
var connection = new ConnectionSettings  
{  
 Tenant = "[[Replace with AAD tenant value]]",  
 EnvironmentId = "[[Replace with PowerApps environment ID value]]",  
 Credentials = new UserImpersonationCredentialsSettings  
 {  
 ApplicationId = "[[Replace with Function application ID value]]",  
 ApplicationSecret = "[[Replace with Function application secret value]]"  
 }  
};  
  
using (var client = await connection.CreateClient(req))  
{  
 // Query product categories for Surfaces and Phones  
 var query = client.GetRelationalEntitySet<ProductCategory>()  
 .CreateQueryBuilder()  
 .Where(pc => pc.Name == "Surface" || pc.Name == "Phone")  
 .Project(pc => pc.SelectField(f => f.CategoryId).SelectField(f => f.Name));  
  
 OperationResult<IReadOnlyList<ProductCategory>> queryResult = null;  
 client.CreateRelationalBatchExecuter(RelationalBatchExecutionMode.Transactional)  
 .Query(query, out queryResult)  
 .ExecuteAsync().Wait();  
  
 // Delete any Surfaces and Phones  
 var deleteExecutor = client.CreateRelationalBatchExecuter(RelationalBatchExecutionMode.Transactional);  
 foreach (var entry in queryResult.Result)  
 {  
 deleteExecutor.DeleteWithoutConcurrencyCheck(entry);  
 }  
 deleteExecutor.ExecuteAsync().Wait();  
  
 // Insert Surface and Phone product lines  
 var surfaceCategory = new ProductCategory() { Name = "Surface", Description = "Surface product line" };  
 var phoneCategory = new ProductCategory() { Name = "Phone", Description = "Phone product line" };  
 await client.CreateRelationalBatchExecuter(RelationalBatchExecutionMode.Transactional)  
 .Insert(surfaceCategory)  
 .Insert(phoneCategory)  
 .ExecuteAsync();  
  
 // Query for Surface and Phone Product lines  
 query = client.GetRelationalEntitySet<ProductCategory>()  
 .CreateQueryBuilder()  
 .Where(pc => pc.Name == name)  
 .OrderByAscending(pc => new object[] { pc.CategoryId })  
 .Project(pc => pc.SelectField(f => f.CategoryId).SelectField(f => f.Name).SelectField(f => f.Description));  
  
 await client.CreateRelationalBatchExecuter(RelationalBatchExecutionMode.Transactional)  
 .Query(query, out queryResult)  
 .ExecuteAsync();  
  
 // Update all selected Product Lines with description  
 var updateExecutor = client.CreateRelationalBatchExecuter(RelationalBatchExecutionMode.Transactional);  
 foreach (var entry in queryResult.Result)  
 {  
 log.Info($"Updateing '{entry.Name}'.");  
 var updateProductCategory = client.CreateRelationalFieldUpdates<ProductCategory>();  
 string updatedDescription = $"{DateTime.Now.ToString()} - Updated '{entry.Name}'";  
 updateProductCategory.Update(pc => pc.Description, updatedDescription);  
  
 updateExecutor.Update(entry, updateProductCategory);  
 }  
 await updateExecutor.ExecuteAsync();  
  
 log.Info($"C# HTTP trigger function completed.");  
 return req.CreateResponse(HttpStatusCode.OK);  
}

Configure the target environment, and security setting of the app by replacing the corresponding bracket text in code with configuration values:

1. **AAD tenant** should replace [[Replace with AAD tenant value]].
2. **PowerApps environment ID** should replace [[Replace with PowerApps environment ID value]].
3. **AAD application** should replace [[Replace with function application ID value]].
4. **AAD application secret** should replace [[Replace with Function application secret value]].
5. Click **Save** at the top of the pane.

# Console client app creation and configuration

## Prerequisites

To create a new console project you'll need [Visual Studio 2015](https://www.visualstudio.com/) or above installed on your computer.

Ensure you have the following configuration values from previous steps:

1. **Function URL**. This value identifies location of the Azure Function.
2. **AAD tenant**. This value identifies the tenant your database resides in.
3. **AAD client console application ID**. This value identifies the AAD app you registered earlier for the client console application.
4. **AAD client console application redirect URI**. This value specifies the redirect URI used when you are prompted to login.
5. **AAD function application resource URI**. This value uniquely identifies the function application your client application will be calling. Think of this as an alternative ID for you function's AAD application registration.

## Project creation and configuration

1. Start **Visual Studio**.
2. From the **File** menu select **New project**.
3. In the **New Project** dialog box, click **Installed > Templates > Visual C#**.
   1. Choose **Console Application**.
   2. Make sure that .NET Framework 4.5.2 is selected as the target framework.
   3. Specify a name for your project and create the new Visual Studio solution.
4. Find your project on the Solution Explorer, right click on it and select **Manage NuGet packages**.
   1. Search for **Microsoft.AspNet.WebApi.Client**.
   2. Select the **Microsoft.AspNet.WebApi.Client** NuGet package and click on **Install**, to get the latest package.
   3. Proceed through the **License acceptance** dialog. **Note** that by clicking accept you are agreeing with all package license terms.
   4. Repeat the last 3 steps for **Microsoft.IdentityModel.Clients.ActiveDirectory** and **Newtonsoft.Json**.

From Solution Explorer, open the **Program.cs** file, and replace using statements on top with following code:

using Microsoft.IdentityModel.Clients.ActiveDirectory;  
using System;  
using System.Net.Http;  
using System.Net.Http.Headers;  
using System.Threading.Tasks;

In **Program.cs**, copy the following code to replace the contents of **Program** class.

public const string TenantNameOrId = "[[Replace with AAD tenant value]]";  
public const string ClientAppId = "[[Replace with AAD client application ID value]]";  
public const string RedirectUri = "[[Replace with AAD client redirect URI value]]";  
public const string AzureHostedResetUriString = "[[Replace with Function URL value]]"; // Azure hosted Function URI  
public const string FunctionResourceId = "[[Replace with AAD function resource ID value]]";  
  
public static string Authority { get { return string.Format(AuthorityTemplate, TenantNameOrId); } }  
private const string AuthorityTemplate = "https://login.windows.net/{0}";  
public const string LocalHostedUpdateUriString = "http://localhost:7071/api/UpdateProductCategory"; // Locally hosted Function URI  
public const string AuthorizationHeaderScheme = "Bearer";  
  
static void Main(string[] args)  
{  
 // Run operation as async function  
 UpdateEntityAsync().Wait();  
 Console.WriteLine("Press any key to continue ...");  
 Console.ReadLine();  
}  
  
private static async Task UpdateEntityAsync()  
{  
 // Get HTTP client and send request  
 var client = await GetHttpClientAsync();  
 bool isAzureHosted = false;  
 var updateUriString = isAzureHosted ? $"{AzureHostedResetUriString}&name=Surface" : $"{LocalHostedUpdateUriString}?name=Surface";  
 var response = await client.PostAsJsonAsync<object>(updateUriString, null);  
 Console.WriteLine($"Status: '{response.StatusCode}'");  
 Console.WriteLine($"Contents: {await response.Content.ReadAsStringAsync()}");  
}  
  
private static async Task<HttpClient> GetHttpClientAsync()  
{  
 // Prompt for login and create security token  
 var authenticationContext = new AuthenticationContext(Authority);  
 AuthenticationResult authenticationResult = await authenticationContext.AcquireTokenAsync(FunctionResourceId  
 , ClientAppId  
 , new Uri(RedirectUri)  
 , new PlatformParameters(PromptBehavior.Always));  
 var securityTokenString = authenticationResult.CreateAuthorizationHeader();  
  
 // Create and configure the HTTP client  
 var client = new HttpClient();  
 var authorizationHeaderParameter = securityTokenString.Replace(AuthorizationHeaderScheme + " ", string.Empty);  
 client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(AuthorizationHeaderScheme, authorizationHeaderParameter);  
 return client;  
}

Configure the target environment, and security setting of the app by replacing the corresponding bracket text in code with configuration values:

1. **AAD tenant** should replace [[Replace with AAD tenant value]].
2. **AAD client application ID** should replace [[Replace with AAD client application ID value]].
3. **AAD client application redirect URI** should replace [[Replace with AAD client redirect URI value]].
4. **Function URL** should replace [[Replace with Function URL value]].
5. **AAD function resource ID** should replace [[Replace with AAD function resource ID value]].

The client application can call either the hosted version of the Function in Azure, or the locally hosted version from a Visual Studio Function Application. If you are testing against Azure, set the isAzureHosted variable to true, to use the correct URI value.

public const string AzureHostedResetUriString = "https://[unique\_id].azurewebsites.net/api/UpdateProductCategory?code=[unique\_code]"; // Azure hosted Function URI  
public const string LocalHostedUpdateUriString = "http://localhost:7071/api/UpdateProductCategory"; // Locally hosted Function URI  
bool isAzureHosted = true; // false  
var updateUriString = isAzureHosted ? $"{AzureHostedResetUriString}&name=Surface" : $"{LocalHostedUpdateUriString}?name=Surface";

### Compile and run the project

1. Ensure the project compiles by right clicking on the project and clicking **Build**.
2. Run the client code by clicking on **Start** or pressing **F5**.
3. Login using **your credentials** when the Azure AD prompt appears. The first time you run the application, you will be prompted to allow the AAD application you registered earlier to access the services CDS uses.
4. Verify that the program runs and calls the Function.
5. Verify that the function updates Product Categories as expected.
6. **Note** that if you see log errors related to assembly loading please refer to **Assembly load issue after publish to Azure** in the troubleshooting section.

# Troubleshooting

This section will contain the most commonly issue encountered and reported by consumers of this topic.

## Required permissions service not found

In some AAD configurations, like with nested tenants, you may be unable to find the **PowerApps Runtime Service** and **Windows Azure Service Management API** when setting up required permissions in the previous step. In such a case you need to modify the application's JSON manifest directly, by clicking on **Manifest** on top of the registered app pane. Add the following entries under the JSON array named requiredResourceAccess while maintaining validity of the manifest, then click **Save**.

{  
 "resourceAppId": "82f77645-8a66-4745-bcdf-9706824f9ad0",  
 "resourceAccess": [  
 {  
 "id": "4ae1b148-ab4d-496d-8183-9292090fcca4",  
 "type": "Scope"  
 }  
 ]  
},  
{  
 "resourceAppId": "797f4846-ba00-4fd7-ba43-dac1f8f63013",  
 "resourceAccess": [  
 {  
 "id": "41094075-9dad-400e-a0bd-54e686782033",  
 "type": "Scope"  
 }  
 ]  
}

## Assembly load issue after publish to Azure

If you started from a Visual Studio Function Application project and published to the Azure Functions portal, you may see an issue with assembly loading when the function gets called. Error may look like the following:

Function completed (Failure, Id=00000000-0000-0000-0000-000000000007)  
Exception while executing function: Functions.UpdateProductCategory. Microsoft.CommonDataService.ServiceClient.Security: Could not load file or assembly 'Microsoft.CommonDataService.Common, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35' or one of its dependencies. The system cannot find the file specified.

This issue can be resolve by deleting the **project.lock.json** file:

1. Go to the **View files** tab on the top right of the pane.
2. Select the filw named **project.lock.json**.
3. Click **Delete** on top of the pane.
4. Reissue the request from client.