

**Azure Integration Runtime** is managed by Microsoft. All the patching, scaling and maintenance of the underlying infrastructure is taken care of. The IR can only access data stores and services in public networks.

**Self-hosted Integration Runtimes** use infrastructure and hardware managed by you. You’ll need to address all the patching, scaling and maintenance. The IR can access resources in both public and private networks.

**Azure-SSIS Integration Runtimes** are VMs running the SSIS engine which allow you to natively execute SSIS packages. They are managed by Microsoft. As a result, all the patching, scaling and maintenance is taken care of. The IR can access resources in both public and private networks.

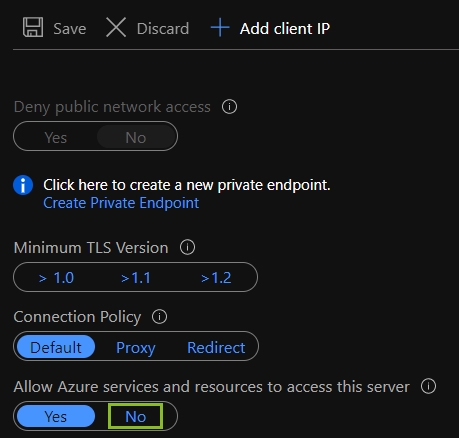
**Virtual Network Service Endpoint**

Using [Virtual Network Service Endpoints](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-service-endpoints-overview) to restrict SQL DB access to only the specified Virtual Network (VNet) adds an extra layer of security. Service Endpoints enables private IP addresses in the VNet to reach the endpoint of an Azure service without needing a public IP address on the VNet.

Once you enable service endpoints in your VNet, you can add a VNet rule to secure the Azure service resources to your VNet. The rule provides improved security by fully removing public internet access to resources and allowing traffic only from your VNet.

In order for the Azure-SSIS IR to access the SQL Database, it needs to be joined to the same VNet and Subnet as illustrated by the above diagram (scenario 4). In this way, only this Subnet can access the SQL Database.

With that in place, turning off “Allow Azure Services to Access Server” is the next step as both the IR and the Azure SQL DB now operate within the context of a VNet and can communicate with private IP addresses which is more secure.



# Configure a self-hosted IR as a proxy for an Azure-SSIS IR

* Article
* 02/16/2022
* 11 minutes to read
* 16 contributors

**APPLIES TO:** https://docs.microsoft.com/en-us/azure/data-factory/media/applies-to/yes.png Azure Data Factory https://docs.microsoft.com/en-us/azure/data-factory/media/applies-to/yes.png Azure Synapse Analytics (Preview)

This article describes how to run SQL Server Integration Services (SSIS) packages on an Azure-SSIS Integration Runtime (Azure-SSIS IR) with a self-hosted integration runtime (self-hosted IR) configured as a proxy.

With this feature, you can access data and run tasks on premises without having to [join your Azure-SSIS IR to a virtual network](https://docs.microsoft.com/en-us/azure/data-factory/join-azure-ssis-integration-runtime-virtual-network). The feature is useful when your corporate network has a configuration too complex or a policy too restrictive for you to inject your Azure-SSIS IR into it.

This feature can only be enabled on SSIS Data Flow Task and Execute SQL/Process Tasks for now.

Enabled on Data Flow Task, this feature will break it down into two staging tasks whenever applicable:

* **On-premises staging task**: This task runs your data flow component that connects to an on-premises data store on your self-hosted IR. It moves data from the on-premises data store into a staging area in your Azure Blob Storage or vice versa.
* **Cloud staging task**: This task runs your data flow component that doesn't connect to an on-premises data store on your Azure-SSIS IR. It moves data from the staging area in your Azure Blob Storage to a cloud data store or vice versa.

If your Data Flow Task moves data from on premises to cloud, then the first and second staging tasks will be on-premises and cloud staging tasks, respectively. If your Data Flow Task moves data from cloud to on premises, then the first and second staging tasks will be cloud and on-premises staging tasks, respectively. If your Data Flow Task moves data from on premises to on premises, then the first and second staging tasks will be both on-premises staging tasks. If your Data Flow Task moves data from cloud to cloud, then this feature isn't applicable.

Enabled on Execute SQL/Process Tasks, this feature will run them on your self-hosted IR.

Other benefits and capabilities of this feature allow you to, for example, set up your self-hosted IR in regions that are not yet supported by an Azure-SSIS IR, and allow the public static IP address of your self-hosted IR on the firewall of your data sources.

## Prepare the self-hosted IR

To use this feature, you first create a data factory and set up an Azure-SSIS IR in it. If you have not already done so, follow the instructions in [Set up an Azure-SSIS IR](https://docs.microsoft.com/en-us/azure/data-factory/tutorial-deploy-ssis-packages-azure).

You then set up your self-hosted IR in the same data factory where your Azure-SSIS IR is set up. To do so, see [Create a self-hosted IR](https://docs.microsoft.com/en-us/azure/data-factory/create-self-hosted-integration-runtime).

Finally, you download and install the latest version of self-hosted IR, as well as the additional drivers and runtime, on your on-premises machine or Azure virtual machine (VM), as follows:

* Download and install the latest version of [self-hosted IR](https://www.microsoft.com/download/details.aspx?id=39717).
* If you use Object Linking and Embedding Database (OLEDB), Open Database Connectivity (ODBC), or ADO.NET connectors in your packages, download and install the relevant drivers on the same machine where your self-hosted IR is installed, if you haven't done so already.

If you use the earlier version of the OLEDB driver for SQL Server (SQL Server Native Client [SQLNCLI]), [download the 64-bit version](https://www.microsoft.com/download/details.aspx?id=50402).

If you use the latest version of OLEDB driver for SQL Server (MSOLEDBSQL), [download the 64-bit version](https://www.microsoft.com/download/details.aspx?id=56730).

If you use OLEDB/ODBC/ADO.NET drivers for other database systems, such as PostgreSQL, MySQL, Oracle, and so on, you can download the 64-bit versions from their websites.

* If you use data flow components from Azure Feature Pack in your packages, [download and install Azure Feature Pack for SQL Server 2017](https://www.microsoft.com/download/details.aspx?id=54798) on the same machine where your self-hosted IR is installed, if you haven't done so already.
* If you haven't done so already, [download and install the 64-bit version of Visual C++ (VC) runtime](https://www.microsoft.com/en-us/download/details.aspx?id=40784) on the same machine where your self-hosted IR is installed.

### Enable Windows authentication for on-premises tasks

If on-premises staging tasks and Execute SQL/Process Tasks on your self-hosted IR require Windows authentication, you must also [configure Windows authentication feature on your Azure-SSIS IR](https://docs.microsoft.com/en-us/sql/integration-services/lift-shift/ssis-azure-connect-with-windows-auth).

Your on-premises staging tasks and Execute SQL/Process Tasks will be invoked with the self-hosted IR service account (NT SERVICE\DIAHostService, by default), and your data stores will be accessed with the Windows authentication account. Both accounts require certain security policies to be assigned to them. On the self-hosted IR machine, go to **Local Security Policy** > **Local Policies** > **User Rights Assignment**, and then do the following:

1. Assign the Adjust memory quotas for a process and Replace a process level token policies to the self-hosted IR service account. This should occur automatically when you install your self-hosted IR with the default service account. If it doesn't, assign those policies manually. If you use a different service account, assign the same policies to it.
2. Assign the Log on as a service policy to the Windows Authentication account.

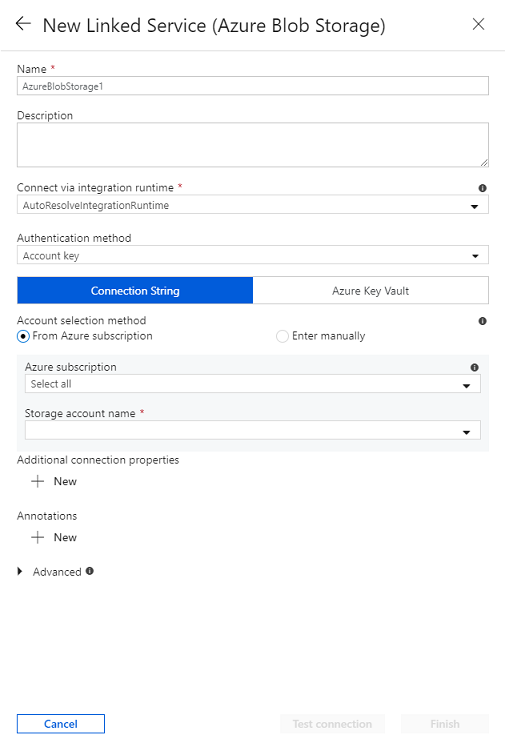
## Prepare the Azure Blob Storage linked service for staging

If you haven't already done so, create an Azure Blob Storage linked service in the same data factory where your Azure-SSIS IR is set up. To do so, see [Create an Azure Data Factory linked service](https://docs.microsoft.com/en-us/azure/data-factory/quickstart-create-data-factory-portal#create-a-linked-service). Be sure to do the following:

* For **Data Store**, select **Azure Blob Storage**.
* For **Connect via integration runtime**, select **AutoResolveIntegrationRuntime** (not your self-hosted IR), so we can ignore it and use your Azure-SSIS IR instead to fetch access credentials for your Azure Blob Storage.
* For **Authentication method**, select **Account key**, **SAS URI**, **Service Principal**, **Managed Identity**, or **User-Assigned Managed Identity**.

**Tip**

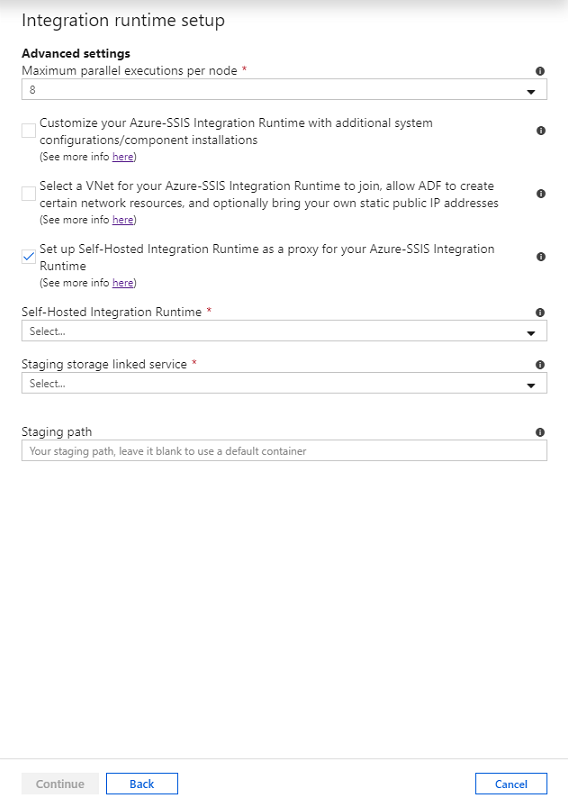
If you select the **Service Principal** method, grant your service principal at least a Storage Blob Data Contributor role. For more information, see [**Azure Blob Storage connector**](https://docs.microsoft.com/en-us/azure/data-factory/connector-azure-blob-storage#linked-service-properties). If you select the **Managed Identity**/**User-Assigned Managed Identity** method, grant the specified system/user-assigned managed identity for your ADF a proper role to access Azure Blob Storage. For more information, see [**Access Azure Blob Storage using Azure Active Directory (Azure AD) authentication with the specified system/user-assigned managed identity for your ADF**](https://docs.microsoft.com/en-us/sql/integration-services/connection-manager/azure-storage-connection-manager#managed-identities-for-azure-resources-authentication).



## Configure an Azure-SSIS IR with your self-hosted IR as a proxy

Having prepared your self-hosted IR and Azure Blob Storage linked service for staging, you can now configure your new or existing Azure-SSIS IR with the self-hosted IR as a proxy in your data factory portal or app. Before you do so, though, if your existing Azure-SSIS IR is already running, you can stop, edit, and then restart it.

1. In the **Integration runtime setup** pane, skip past the **General settings** and **Deployment settings** pages by selecting the **Continue** button.
2. On the **Advanced settings** page, do the following:
   1. Select the **Set up Self-Hosted Integration Runtime as a proxy for your Azure-SSIS Integration Runtime** check box.
   2. In the **Self-Hosted Integration Runtime** drop-down list, select your existing self-hosted IR as a proxy for the Azure-SSIS IR.
   3. In the **Staging storage linked service** drop-down list, select your existing Azure Blob Storage linked service or create a new one for staging.
   4. In the **Staging path** box, specify a blob container in your selected Azure Storage account or leave it empty to use a default one for staging.
   5. Select the **Continue** button.



You can also configure your new or existing Azure-SSIS IR with the self-hosted IR as a proxy by using PowerShell.

PowerShellCopy

$ResourceGroupName = "[your Azure resource group name]"

$DataFactoryName = "[your data factory name]"

$AzureSSISName = "[your Azure-SSIS IR name]"

# Self-hosted integration runtime info - This can be configured as a proxy for on-premises data access

$DataProxyIntegrationRuntimeName = "" # OPTIONAL to configure a proxy for on-premises data access

$DataProxyStagingLinkedServiceName = "" # OPTIONAL to configure a proxy for on-premises data access

$DataProxyStagingPath = "" # OPTIONAL to configure a proxy for on-premises data access

# Add self-hosted integration runtime parameters if you configure a proxy for on-premises data access

if(![string]::IsNullOrEmpty($DataProxyIntegrationRuntimeName) -and ![string]::IsNullOrEmpty($DataProxyStagingLinkedServiceName))

{

Set-AzDataFactoryV2IntegrationRuntime -ResourceGroupName $ResourceGroupName `

-DataFactoryName $DataFactoryName `

-Name $AzureSSISName `

-DataProxyIntegrationRuntimeName $DataProxyIntegrationRuntimeName `

-DataProxyStagingLinkedServiceName $DataProxyStagingLinkedServiceName

if(![string]::IsNullOrEmpty($DataProxyStagingPath))

{

Set-AzDataFactoryV2IntegrationRuntime -ResourceGroupName $ResourceGroupName `

-DataFactoryName $DataFactoryName `

-Name $AzureSSISName `

-DataProxyStagingPath $DataProxyStagingPath

}

}

Start-AzDataFactoryV2IntegrationRuntime -ResourceGroupName $ResourceGroupName `

-DataFactoryName $DataFactoryName `

-Name $AzureSSISName `

-Force

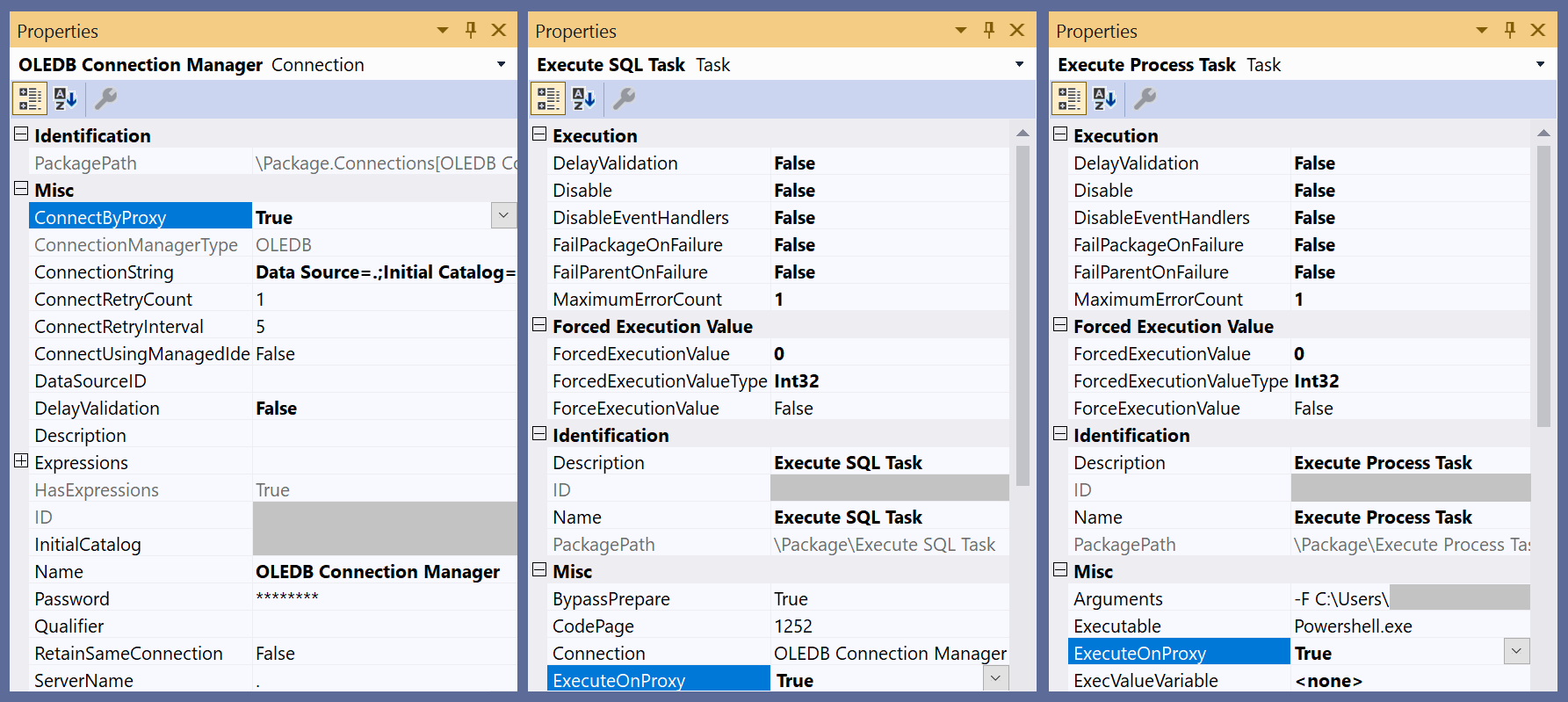
## Enable SSIS packages to use a proxy

By using the latest SSDT as either the SSIS Projects extension for Visual Studio or a standalone installer, you can find a new ConnectByProxy property in the connection managers for supported data flow components and ExecuteOnProxy property in Execute SQL/Process Tasks.

* [Download the SSIS Projects extension for Visual Studio](https://marketplace.visualstudio.com/items?itemName=SSIS.SqlServerIntegrationServicesProjects)
* [Download the standalone installer](https://docs.microsoft.com/en-us/sql/ssdt/download-sql-server-data-tools-ssdt#ssdt-for-vs-2017-standalone-installer)

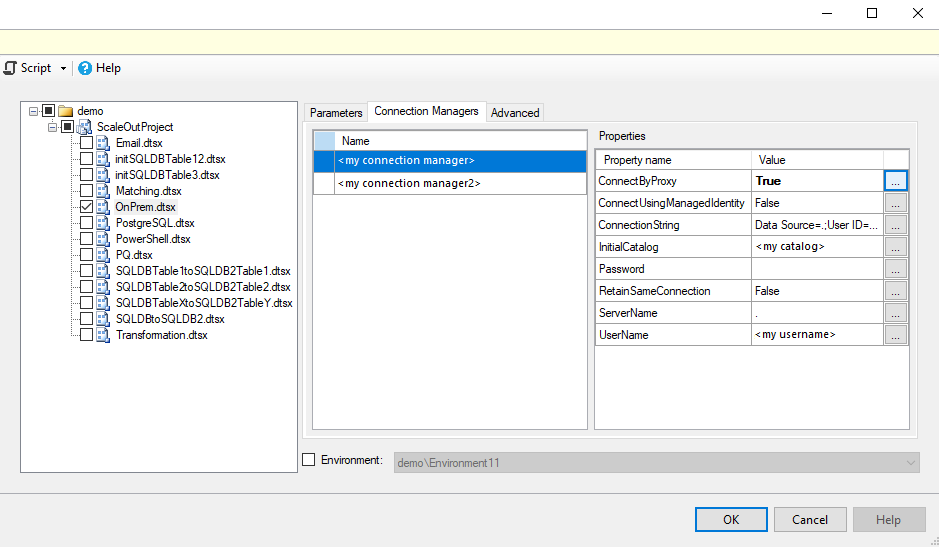
When you design new packages containing Data Flow Tasks with components that access data on premises, you can enable the ConnectByProxy property by setting it to True in the **Properties** pane of relevant connection managers.

When you design new packages containing Execute SQL/Process Tasks that run on premises, you can enable the ExecuteOnProxy property by setting it to True in the **Properties** pane of relevant tasks themselves.

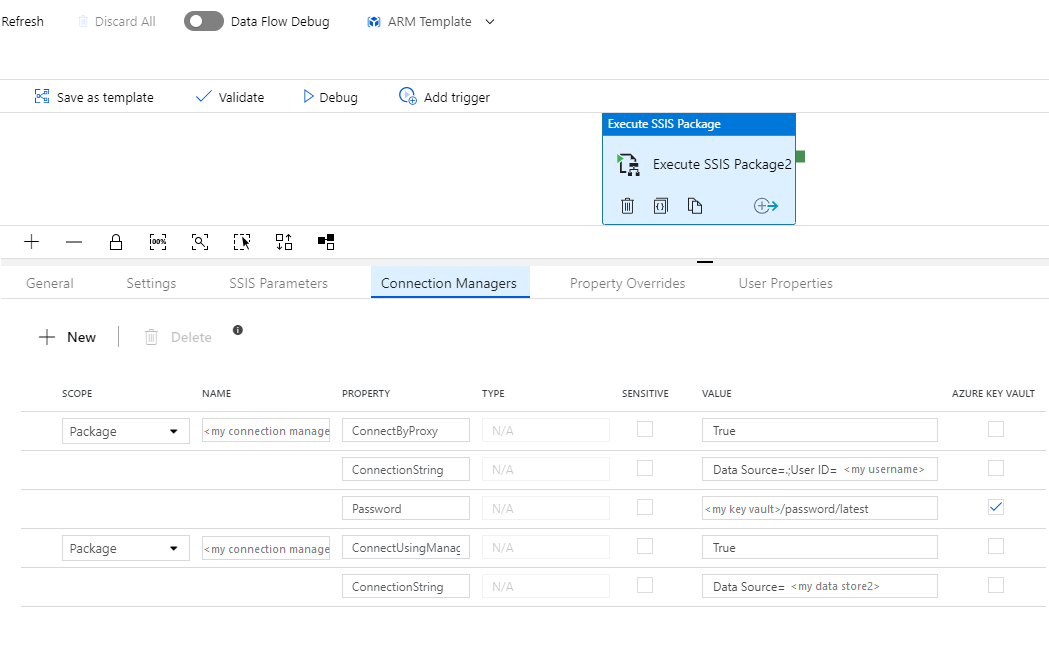


You can also enable the ConnectByProxy/ExecuteOnProxy properties when you run existing packages, without having to manually change them one by one. There are two options:

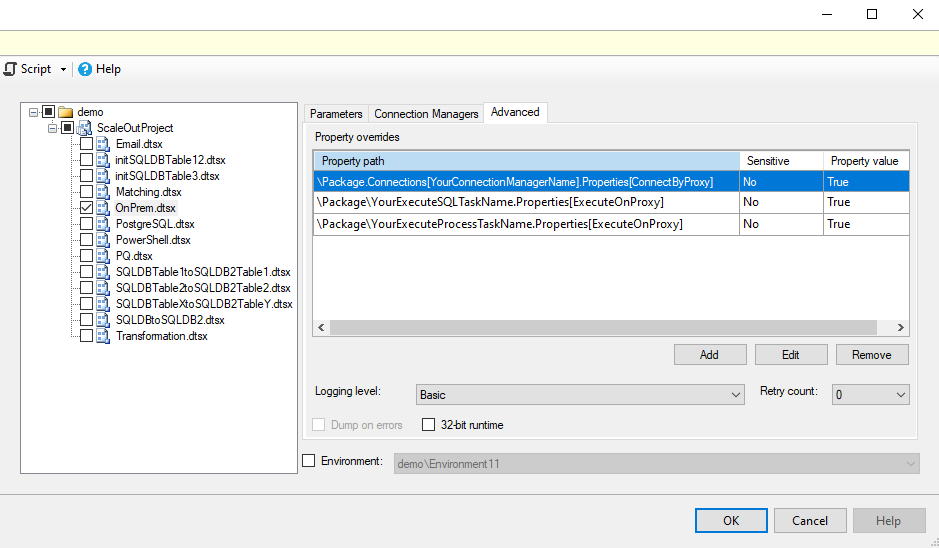
* **Option A**: Open, rebuild, and redeploy the project containing those packages with the latest SSDT to run on your Azure-SSIS IR. You can then enable the ConnectByProxy property by setting it to True for the relevant connection managers that appear on the **Connection Managers** tab of **Execute Package** pop-up window when you're running packages from SSMS.



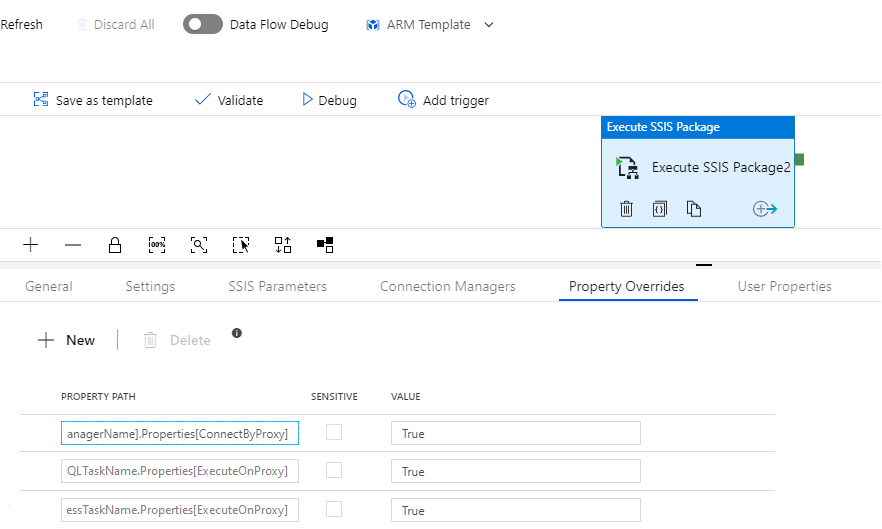
You can also enable the ConnectByProxy property by setting it to True for the relevant connection managers that appear on the **Connection Managers** tab of [Execute SSIS Package activity](https://docs.microsoft.com/en-us/azure/data-factory/how-to-invoke-ssis-package-ssis-activity) when you're running packages in Data Factory pipelines.



* **Option B:** Redeploy the project containing those packages to run on your SSIS IR. You can then enable the ConnectByProxy/ExecuteOnProxy properties by providing their property paths, \Package.Connections[YourConnectionManagerName].Properties[ConnectByProxy]/\Package\YourExecuteSQLTaskName.Properties[ExecuteOnProxy]/\Package\YourExecuteProcessTaskName.Properties[ExecuteOnProxy], and setting them to True as property overrides on the **Advanced** tab of **Execute Package** pop-up window when you're running packages from SSMS.

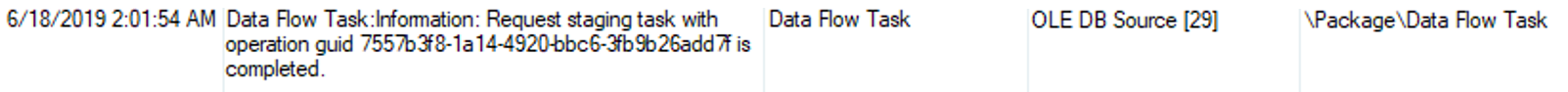


You can also enable the ConnectByProxy/ExecuteOnProxy properties by providing their property paths, \Package.Connections[YourConnectionManagerName].Properties[ConnectByProxy]/\Package\YourExecuteSQLTaskName.Properties[ExecuteOnProxy]/\Package\YourExecuteProcessTaskName.Properties[ExecuteOnProxy], and setting them to True as property overrides on the **Property Overrides** tab of [Execute SSIS Package activity](https://docs.microsoft.com/en-us/azure/data-factory/how-to-invoke-ssis-package-ssis-activity) when you're running packages in Data Factory pipelines.



## Debug the on-premises tasks and cloud staging tasks

On your self-hosted IR, you can find the runtime logs in the C:\ProgramData\SSISTelemetry folder and the execution logs of on-premises staging tasks and Execute SQL/Process Tasks in the C:\ProgramData\SSISTelemetry\ExecutionLog folder. You can find the execution logs of cloud staging tasks in your SSISDB, specified logging file paths, or Azure Monitor depending on whether you store your packages in SSISDB, enable [Azure Monitor integration](https://docs.microsoft.com/en-us/azure/data-factory/monitor-ssis), etc. You can also find the unique IDs of on-premises staging tasks in the execution logs of cloud staging tasks.



If you've raised customer support tickets, you can select the **Send logs** button on **Diagnostics** tab of **Microsoft Integration Runtime Configuration Manager** that's installed on your self-hosted IR to send recent operation/execution logs for us to investigate.

## Billing for the on-premises tasks and cloud staging tasks

The on-premises staging tasks and Execute SQL/Process Tasks that run on your self-hosted IR are billed separately, just as any data movement activities that run on a self-hosted IR are billed. This is specified in the [Azure Data Factory data pipeline pricing](https://azure.microsoft.com/pricing/details/data-factory/data-pipeline/) article.

The cloud staging tasks that run on your Azure-SSIS IR are not be billed separately, but your running Azure-SSIS IR is billed as specified in the [Azure-SSIS IR pricing](https://azure.microsoft.com/pricing/details/data-factory/ssis/) article.

## Enable custom/3rd party data flow components

To enable your custom/3rd party data flow components to access data on premises using self-hosted IR as a proxy for Azure-SSIS IR, follow these instructions:

1. Install your custom/3rd party data flow components targeting SQL Server 2017 on Azure-SSIS IR via [standard/express custom setups](https://docs.microsoft.com/en-us/azure/data-factory/how-to-configure-azure-ssis-ir-custom-setup).
2. Create the following DTSPath registry keys on self-hosted IR if they don’t exist already:
   1. Computer\HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\140\SSIS\Setup\DTSPath set to C:\Program Files\Microsoft SQL Server\140\DTS\
   2. Computer\HKEY\_LOCAL\_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Microsoft SQL Server\140\SSIS\Setup\DTSPath set to C:\Program Files (x86)\Microsoft SQL Server\140\DTS\
3. Install your custom/3rd party data flow components targeting SQL Server 2017 on self-hosted IR under the DTSPath above and ensure that your installation process:
   1. Creates <DTSPath>, <DTSPath>/Connections, <DTSPath>/PipelineComponents, and <DTSPath>/UpgradeMappings folders if they don't exist already.
   2. Creates your own XML file for extension mappings in <DTSPath>/UpgradeMappings folder.
   3. Installs all assemblies referenced by your custom/3rd party data flow component assemblies in the global assembly cache (GAC).

Here are examples from our partners, [Theobald Software](https://kb.theobald-software.com/xtract-is/XIS-for-Azure-SHIR) and [Aecorsoft](https://www.aecorsoft.com/blog/2020/11/8/using-azure-data-factory-to-bring-sap-data-to-azure-via-self-hosted-ir-and-ssis-ir), who have adapted their data flow components to use our express custom setup and self-hosted IR as a proxy for Azure-SSIS IR.

## Enforce TLS 1.2

If you need to access data stores that have been configured to use only the strongest cryptography/most secure network protocol (TLS 1.2), including your Azure Blob Storage for staging, you must enable only TLS 1.2 and disable older SSL/TLS versions at the same time on your self-hosted IR. To do so, you can download and run the main.cmd script that we provide in the CustomSetupScript/UserScenarios/TLS 1.2 folder of our public preview blob container. Using [Azure Storage Explorer](https://storageexplorer.com/), you can connect to our public preview blob container by entering the following SAS URI:

https://ssisazurefileshare.blob.core.windows.net/publicpreview?sp=rl&st=2020-03-25T04:00:00Z&se=2025-03-25T04:00:00Z&sv=2019-02-02&sr=c&sig=WAD3DATezJjhBCO3ezrQ7TUZ8syEUxZZtGIhhP6Pt4I%3D

## Current limitations

* Only data flow components that are built-in/preinstalled on Azure-SSIS IR Standard Edition, except Hadoop/HDFS/DQS components, are currently supported, see [all built-in/preinstalled components on Azure-SSIS IR](https://docs.microsoft.com/en-us/azure/data-factory/built-in-preinstalled-components-ssis-integration-runtime).
* Only custom/3rd party data flow components that are written in managed code (.NET Framework) are currently supported - Those written in native code (C++) are currently unsupported.
* Changing variable values in both on-premises and cloud staging tasks is currently unsupported.
* Changing variable values of type object in on-premises staging tasks won't be reflected in other tasks.
* ParameterMapping in OLEDB Source is currently unsupported. As a workaround, please use SQL Command From Variable as the AccessMode and use Expression to insert your variables/parameters in a SQL command. As an illustration, see the ParameterMappingSample.dtsx package that can be found in the SelfHostedIRProxy/Limitations folder of our public preview blob container. Using Azure Storage Explorer, you can connect to our public preview blob container by entering the above SAS URI.

