# Lifting & Shifting/Modernizing your ETL Workflows with SSIS in ADFv2

Sandy Winarko
Principal Program Manager, IM China

Microsoft

# Sections

- Microsoft ETL/ELT Services in Azure
- ADF Concepts
- Enterprise Edition
- Custom Setup Interface
- Provisioning Methods
- Deployment Methods
- Execution Methods
- Scheduling Methods
- Monitoring Methods
- On-Demand/Just-In-Time Provisioning
- Resources

# Microsoft ETL/ELT Services in Azure

# Customer Insights

- SSIS is a traditional ETL tool that comes bundled with SQL Server on premises
  - Has been around for more than 10 years
  - Some customers have started to lift & shift their ETL workloads to the cloud to reduce their on-prem infra, but found managing Infrastructure as a Service (IaaS)/VMs challenging

# Customer Insights

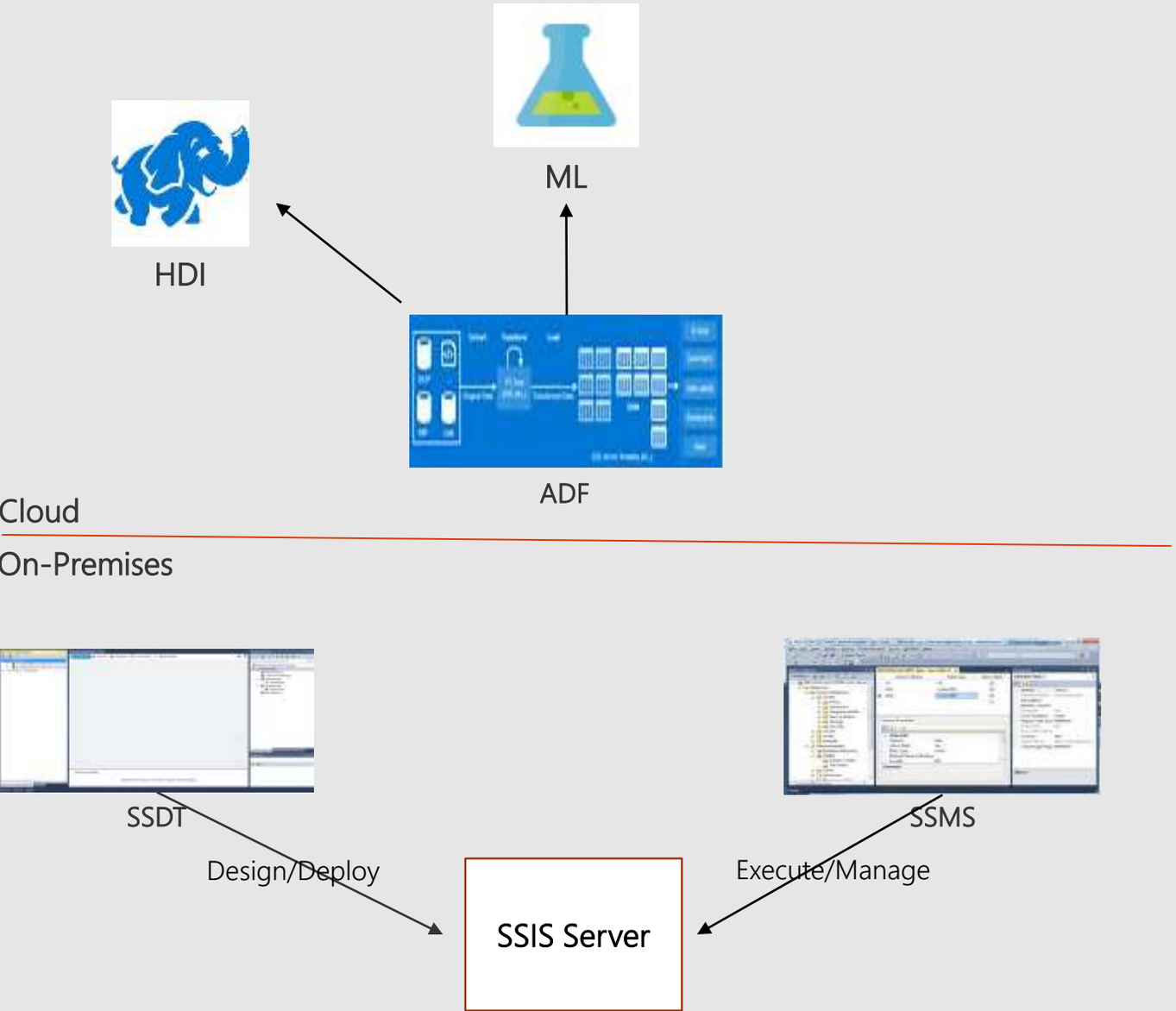
- SSIS is a traditional ETL tool that comes bundled with SQL Server on premises
  - Has been around for more than 10 years
  - Some customers have started to lift & shift their ETL workloads to the cloud to reduce their on-prem infra, but found managing Infrastructure as a Service (IaaS)/VMs challenging

- Azure Data Factory (ADF) is a modern ELT tool that moves/copies data and dispatches transformations for Big Data Analytics in the cloud
  - Some gaps in ELT workflows can be filled w/ code-free authoring of transformations/built-in tasks from SSIS
  - Some customers have started to combine ADF with SSIS on IaaS/VMs, but found managing IaaS/VMs challenging

# Customer Insights

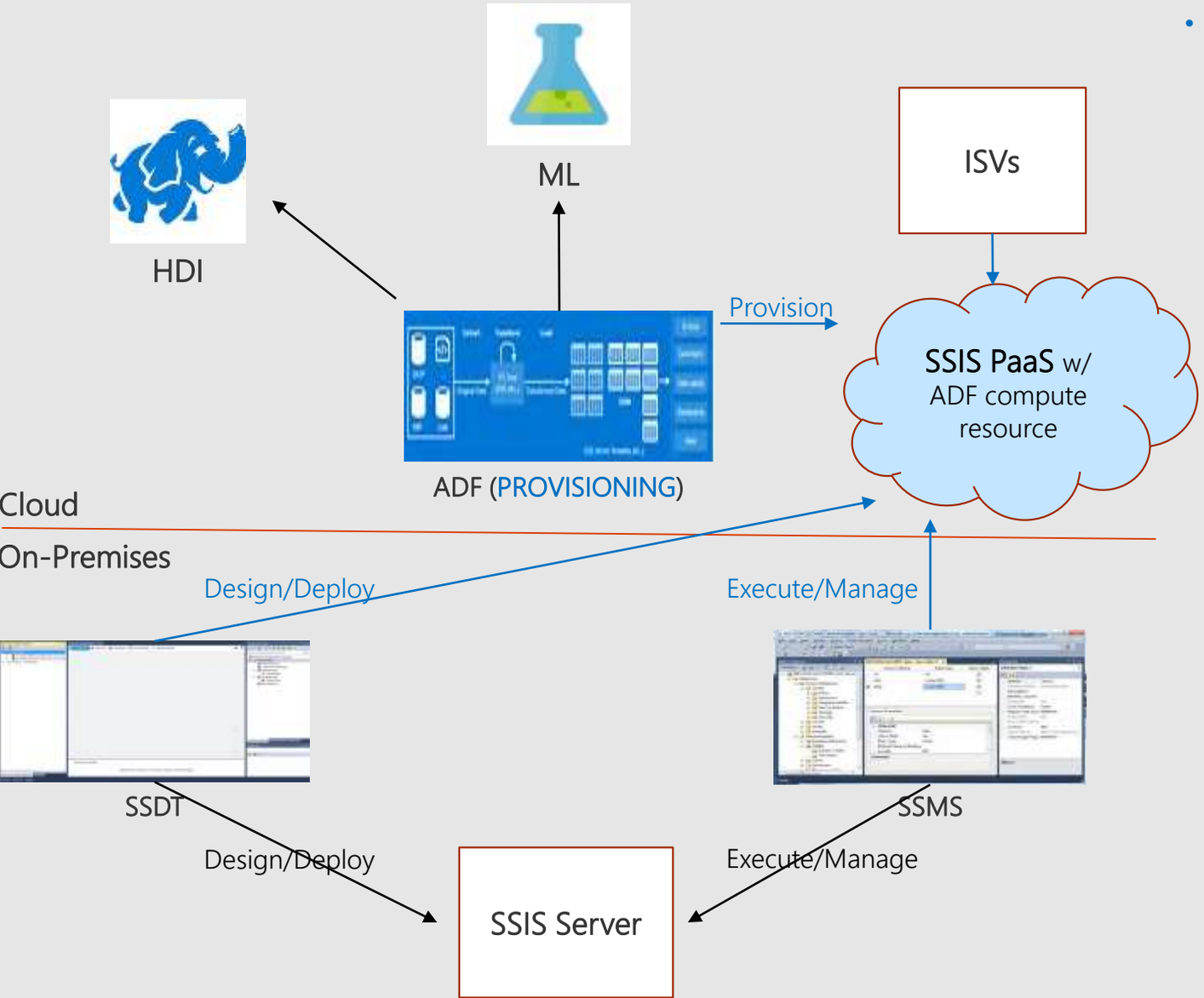
- SSIS is a traditional ETL tool that comes bundled with SQL Server on premises
  - Has been around for more than 10 years
  - Some customers have started to lift & shift their ETL workloads to the cloud to reduce their on-prem infra, but found managing Infrastructure as a Service (IaaS)/VMs challenging
- Azure Data Factory (ADF) is a modern ELT tool that moves/copies data and dispatches transformations for Big Data Analytics in the cloud
  - Some gaps in ELT workflows can be filled w/ code-free authoring of transformations/built-in tasks from SSIS
  - Some customers have started to combine ADF with SSIS on IaaS/VMs, but found managing IaaS/VMs challenging
- Evolution of a cloud-first product: SSIS on premises -> IaaS -> PaaS
  - The stage is set for SSIS PaaS…

# Microsoft ETL/ELT Services in Azure



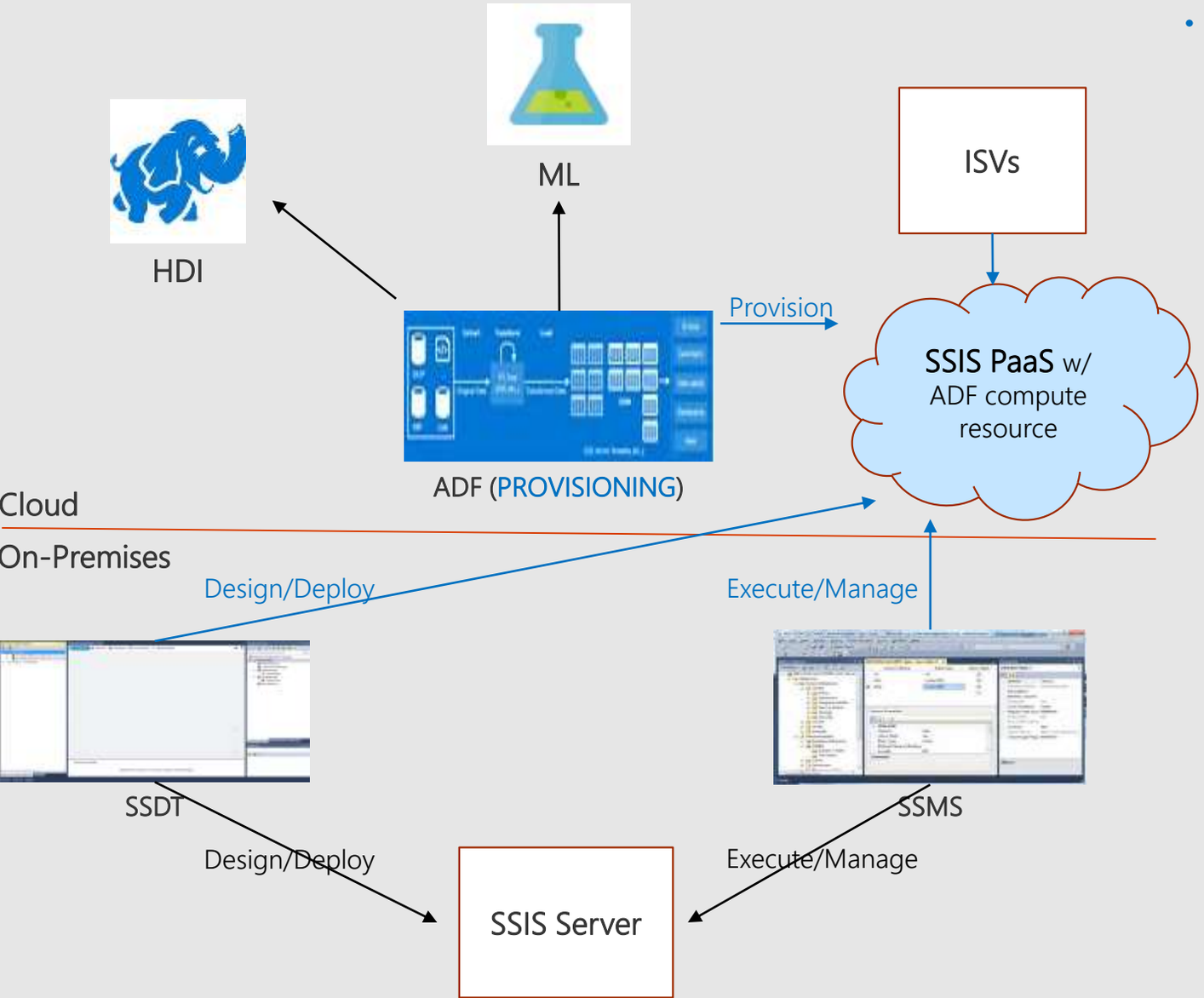

- We are building SSIS PaaS in ADFv2 phase by phase to offer a unified platform for Microsoft ETL/ELT services in the cloud

# Microsoft ETL/ELT Services in Azure



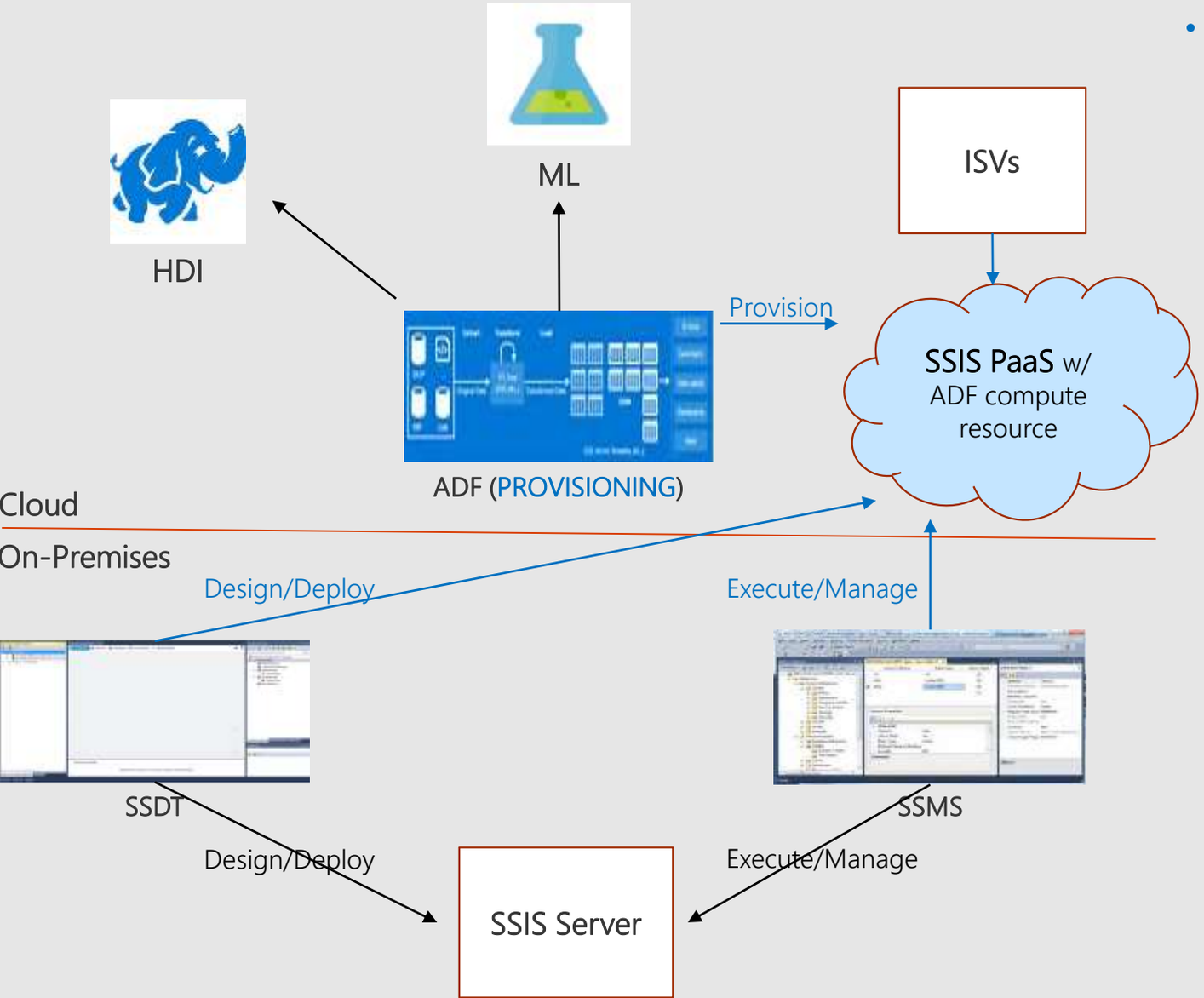

- **Phase 1**: Use ADFv2 to provision SSIS PaaS – Launched at MS Ignite (Sep'17)
  - Create ADFv2, if you have not done so already
  - Use ADFv2 App/SDK/API/PSH to provision SSIS PaaS w/ ADF compute resource called Azure-SSIS Integration Runtime (IR)
- Still use SQL Server Data Tools (**SSDT**) to design/deploy SSIS packages
- Still use SQL Server Management Studio (**SSMS**) to execute/manage SSIS packages
- Serve SSIS customers who want to move all/part of their on-premises workloads and just "lift & shift" many existing packages to Azure
- Independent Software Vendors (ISVs) can build extensions/Software as a Service (**SaaS**) on SSIS PaaS

# Microsoft ETL/ELT Services in Azure




- Introducing Azure-SSIS IR: Managed cluster of Azure VMs (nodes) dedicated to run your SSIS packages and no other activities
  - You can scale it up/out by specifying the node size /number of nodes in the cluster
  - You can bring your own Azure SQL Database (DB)/Managed Instance (MI) server to host the catalog of SSIS projects/packages (**SSISDB**) that will be attached to it
  - You can join it to a Virtual Network (VNet) that is connected to your on-prem network to enable on-prem data access
  - Once provisioned, you can enter your Azure SQL DB/MI server endpoint on SSDT/SSMS to deploy SSIS projects/packages and configure/execute them just like using SSIS on premises

# Microsoft ETL/ELT Services in Azure




- Customer cohorts for Phase 1:

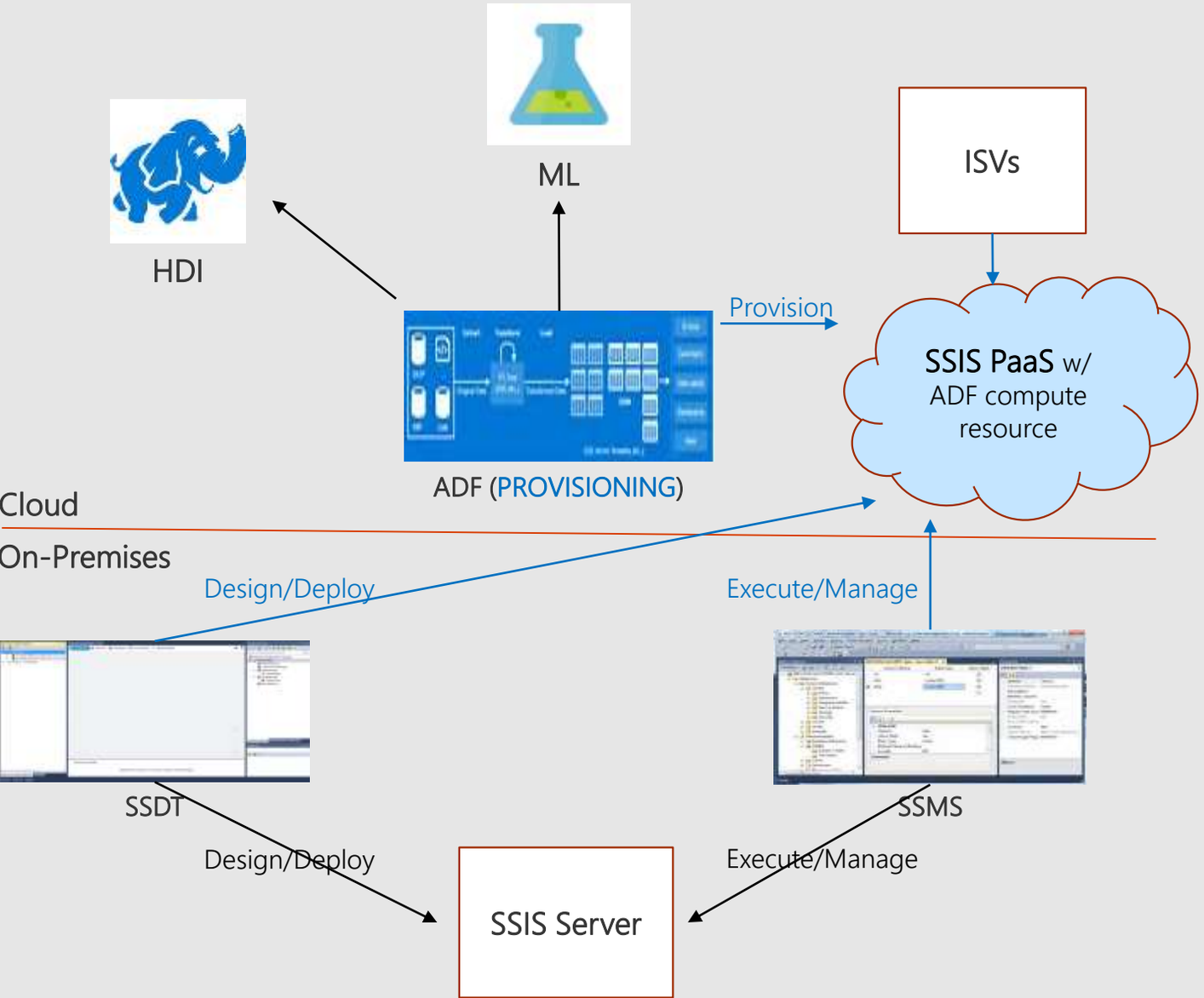
1. "SQL Migrators"

These are SSIS customers who want to retire their on-prem SQL Servers and migrate all apps + data ("complete/full lift & shift") into Azure SQL MI – For them, SSISDB can be hosted by Azure SQL MI inside their VNet

2. "ETL Cost Cutters"

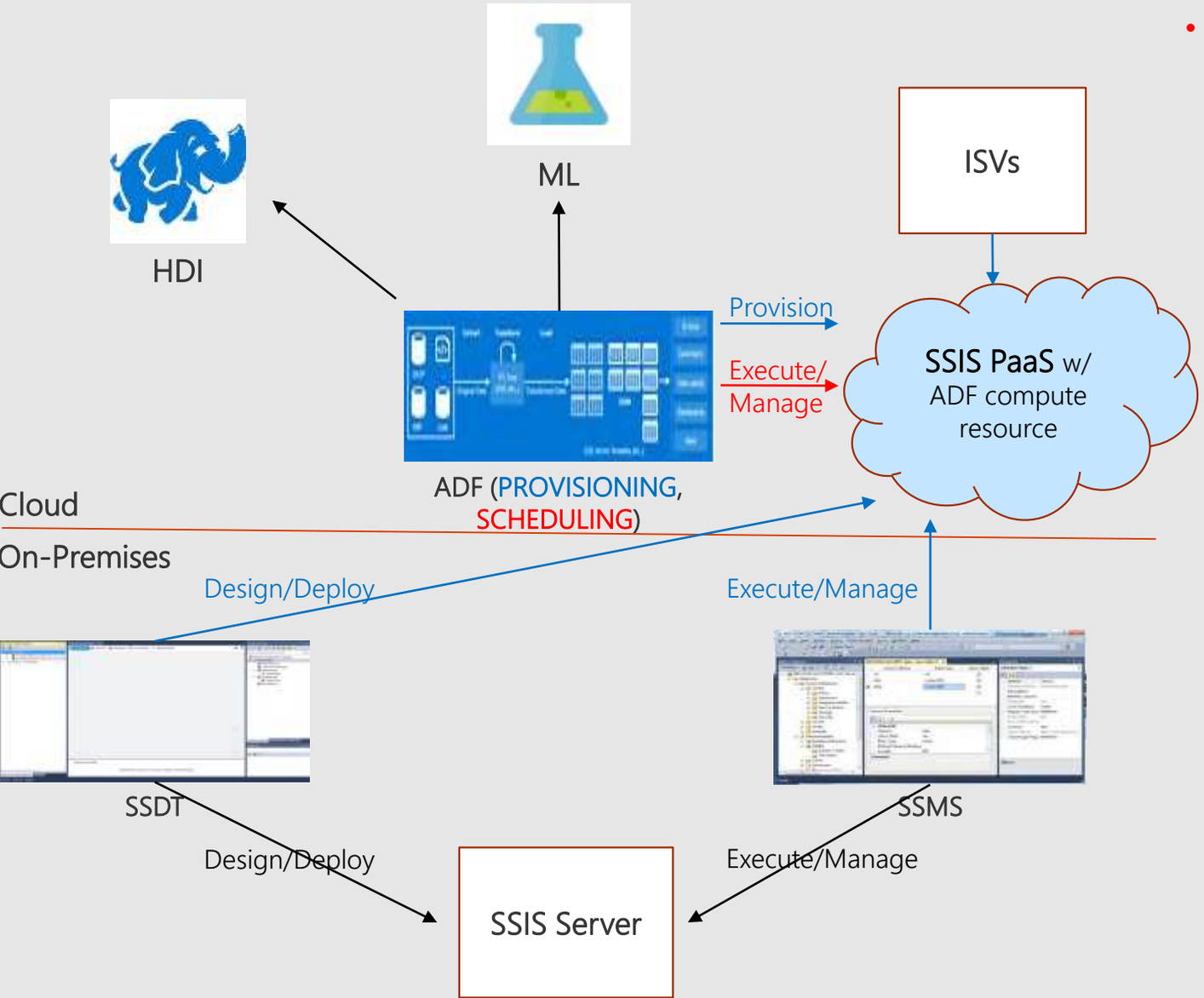
These are SSIS customers who want to lower their operational costs and gain High Availability (HA)/scalability for just their ETL workloads w/o managing their own infra ("partial lift & shift") – For them, SSISDB can be hosted by Azure SQL DB in the public network

# Microsoft ETL/ELT Services in Azure



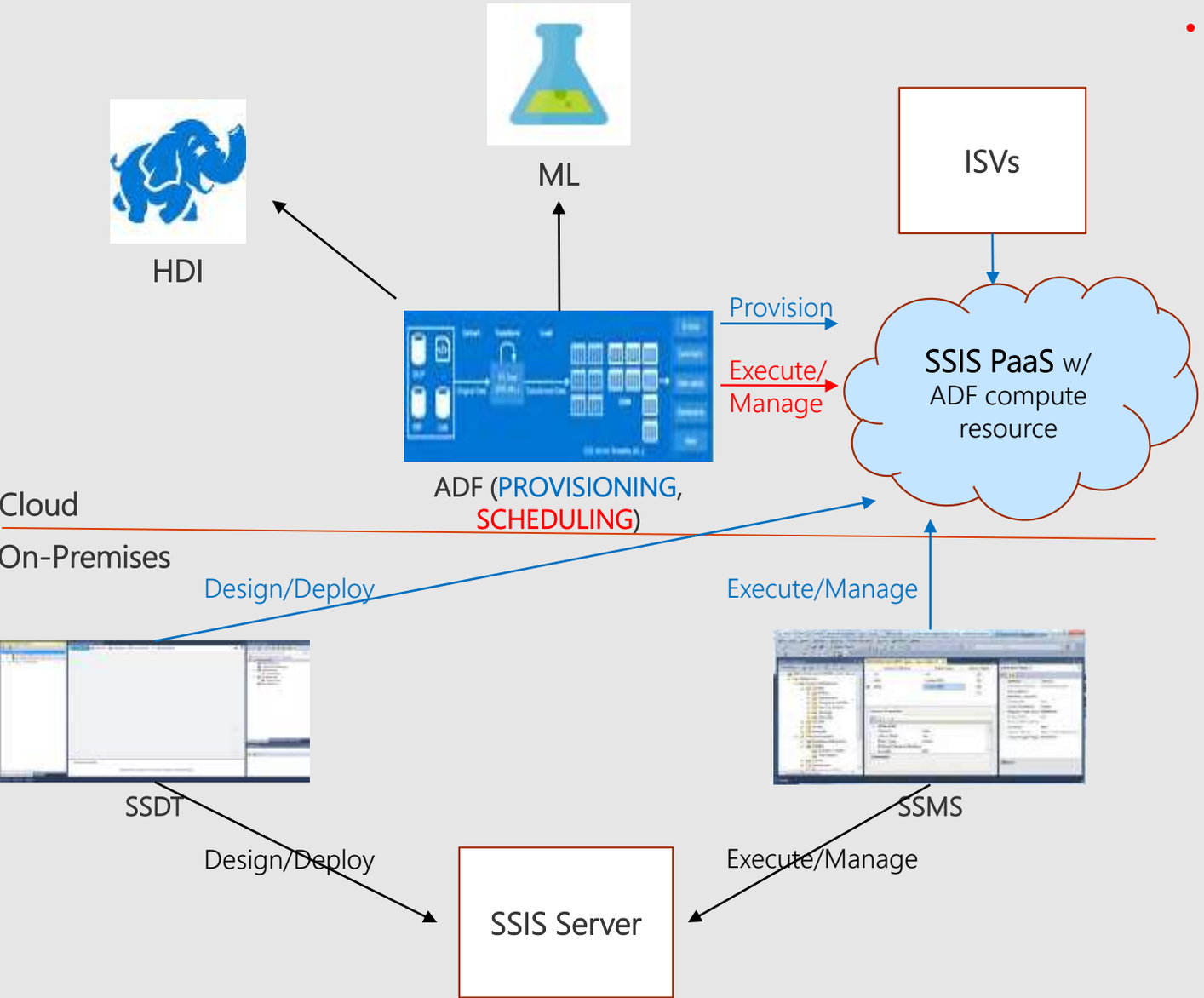

- LIVE now in Public Preview
  - Six locations: East US/East US 2/Central US/North Europe/West Europe/Australia East
  - Six node sizes: Av2/Dv2 series VMs
  - Classic VNet support
  - Standard edition/license
  - 24/7 live-site support
- What's new
  - Azure Resource Manager (ARM) VNet support
  - Enterprise edition/license (Private Preview)
  - Custom setup interface (Private Preview)
  - ADFv2 App/GUI web tool
- Coming soon
  - More locations, e.g. West US, UK South
  - More node sizes, e.g. Dv3/Ev3 series VMs
  - Azure Hybrid Use Benefit (AHUB)/Bring Your Own License (BYOL) support

# Microsoft ETL/ELT Services in Azure




- **Phase 2**: Use ADFv2 to execute/manage SSIS packages deployed to SSIS PaaS as first-class activities – ETA Q2CY18 (TBD)
  - Chain/group them with Azure HDInsight (HDI)/Machine Learning (ML)/other activities inside data pipelines
  - Serve SSIS/ADF customers who want to combine ETL/ELT workflows on a single platform

# Microsoft ETL/ELT Services in Azure




- Customer cohorts for Phase 2:

3. "ETL Modernizers"

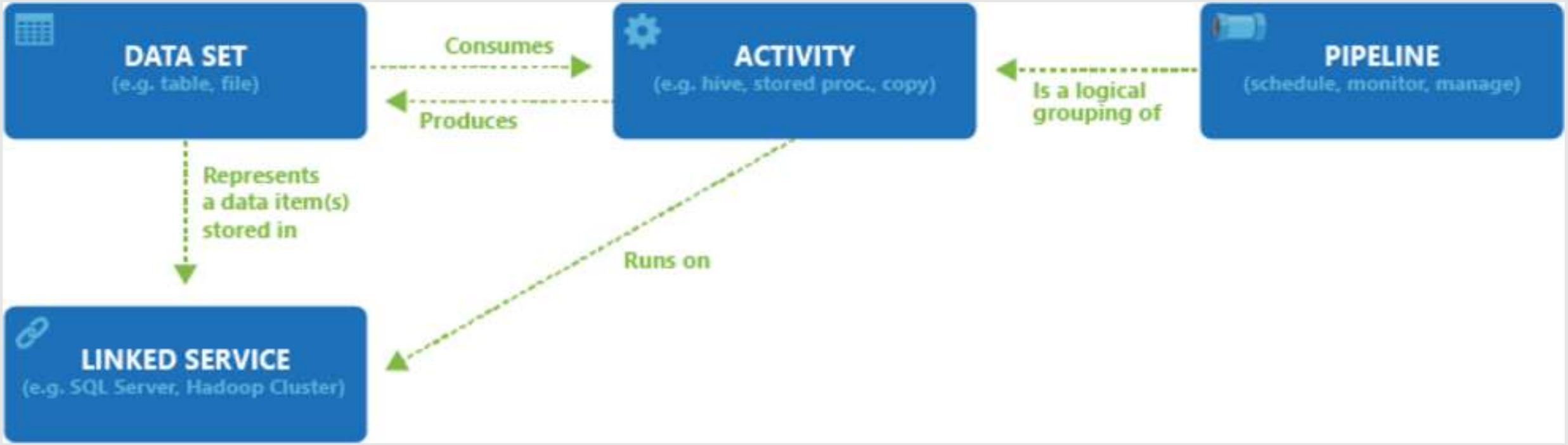
These are SSIS customers who want to modernize their workflows and explore Big Data Analytics in the cloud

4. "ELT Gap Fillers"

These are ADF customers who want to fill some gaps in their workflows w/ code-free authoring of transformations/built-in tasks from SSIS

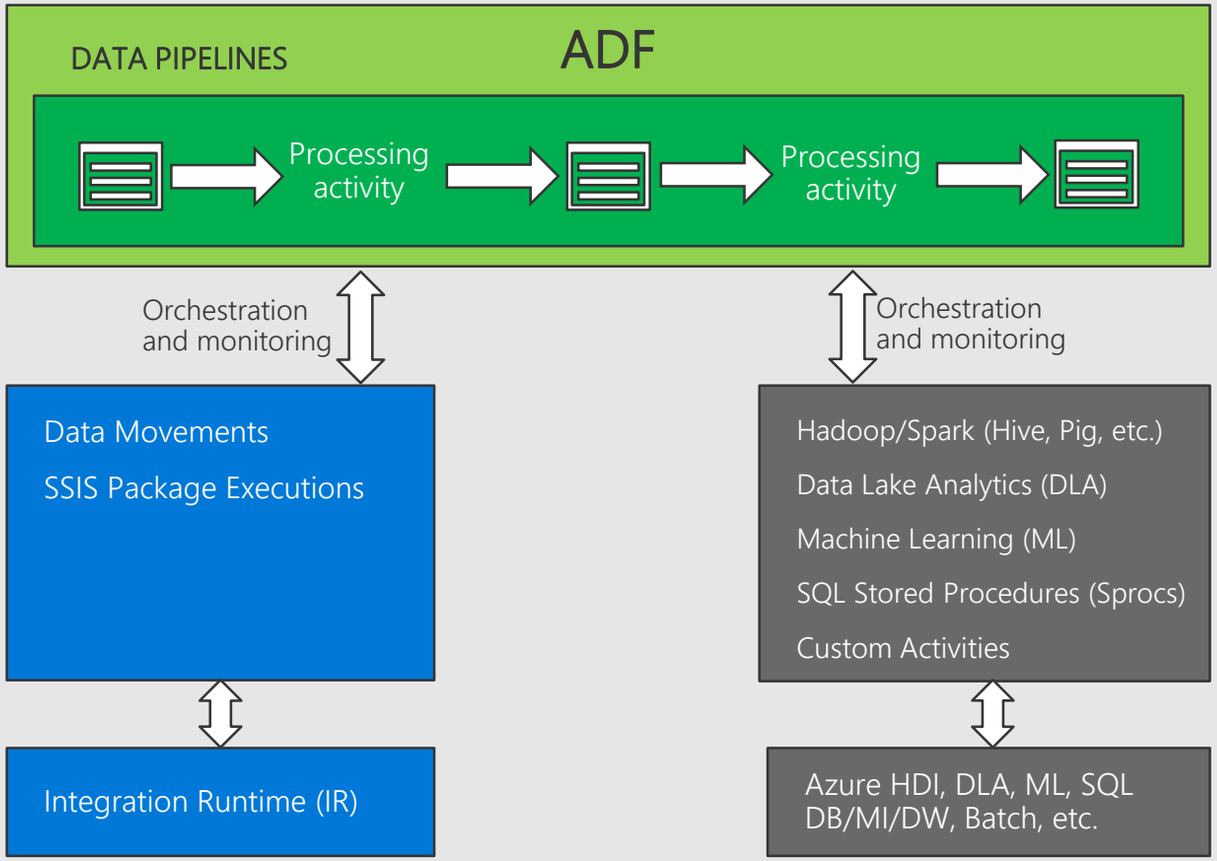
# ADF Concepts

# ADF Basic Concepts




- ADF is Microsoft's unified platform for ETL/ELT services in the cloud
- ADF allows you to build data pipelines and trigger/schedule their runs
- Data pipeline is a chain/group of activities to be performed on your data, e.g. data movements/transformations
- Activities take data sets, which are named references/pointers to data, as inputs/outputs
- Some activities target data store/compute resources allocated outside ADF, e.g. ADLS/HDI/ML/etc.
- Linked services represent those resources and provide the connection info for ADF to orchestrate activities targeting them

# ADFv2 Integration Runtimes




- Integration Runtime (IR) is ADF compute resource that can perform data movements/transformations, including SSIS package executions
- Customers can deploy one/many instances of IR per ADF as required to run pipelines/process data
- IR can run in the public network, inside VNet, or behind corporate firewalls
- SSIS PaaS runs on Azure-SSIS IR, internal/native to ADF that provisions it, while SSISDB is hosted by your own Azure SQL DB/MI server, external to ADF
- Consequently, Azure-SSIS IR is billed under your ADF subscription, separately from SSISDB that is billed under your Azure SQL DB/MI subscription

# Enterprise Edition

# Enterprise Edition – Introduction

- Enterprise Edition of Azure-SSIS IR allows you to use advanced/premium features:
  - Change Data Capture (CDC) components
  - Oracle/Teradata/SAP BW connectors
  - SQL Server Analysis Services (SSAS)/Azure Analysis Services (AAS) connectors/transforms
  - Fuzzy Grouping/Lookup transforms
  - Term Extraction/Lookup transforms

- Some of these features will also require you to install additional components, essentially customizing your Azure-SSIS IR (via Custom Setup Interface)

# Enterprise Edition – Features

| Enterprise Features | Descriptions |
| --- | --- |
| CDC components | • CDC Source/Splitter/Control Task are preinstalled on your Azure-SSIS IR Enterprise Edition<br>• To connect to Oracle, you will also need to install CDC Designer/Service on another machine |
| Oracle connectors | • Oracle Connection Manager/Source/Destination are preinstalled on your Azure-SSIS IR Enterprise Edition<br>• You will also need to install Oracle Call Interface (OCI) driver, and if necessary configure Oracle Transport Network Substrate (TNS), on your Azure-SSIS IR (via Custom Setup Interface) |
| Teradata connectors | • You will need to install Teradata Connection Manager/Source/Destination and Teradata Parallel Transporter (TPT) API + Teradata ODBC driver on your Azure-SSIS IR Enterprise Edition (via Custom Setup Interface) |

# Enterprise Edition – Features

| Enterprise Features | Descriptions |
| --- | --- |
| SAP BW connectors | • SAP BW Connection Manager/Source/Destination are preinstalled on your Azure-SSIS IR Enterprise Edition<br>• You will also need to install SAP BW driver on your Azure-SSIS IR (via Custom Setup Interface)<br>• These connectors support SAP BW 7.0 or earlier versions<br>• To connect to later versions, you can install SAP connectors from our partners (e.g. Theobald Software) on your Azure-SSIS IR (via Custom Setup Interface) |

# Enterprise Edition – Features

| Enterprise Features | Descriptions |
| --- | --- |
| SSAS/AAS components | • Data Mining Model Training/Dimension Processing/Partition Processing Destinations and Data Mining Query Transform are preinstalled on your Azure-SSIS IR Enterprise Edition<br>• All support SSAS, but only Partition Processing Destination supports AAS<br>• To connect to SSAS, you will also need to configure Windows Authentication credentials in your SSISDB<br>• On top of these components, Analysis Services Execute DDL/Processing and Data Mining Query Tasks are also preinstalled on your Azure-SSIS IR Standard/Enterprise Edition |
| Fuzzy Grouping/Lookup Transforms | • They are preinstalled on your Azure-SSIS IR Enterprise Edition and support SQL Server/Azure SQL DB for storing reference data |
| Term Extraction/Lookup Transforms | • They are preinstalled on your Azure-SSIS IR Enterprise Edition and support SQL Server/Azure SQL DB for storing reference data |

# Enterprise Edition – Instructions

- Please send us your Azure subscription ID that you will use for provisioning/reconfiguring your Azure-SSIS IR with this feature – We will whitelist it for preview

- Please download and install our private version of Azure PSH

- When provisioning/reconfiguring your Azure-SSIS IR via PSH, execute Set-AzureRmDataFactoryV2IntegrationRuntime cmdlet with "Enterprise" as the value for new Edition parameter before starting your Azure-SSIS IR, e.g.

```
$MyAzureSsisIrEdition = "Enterprise"

Set-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $MyDataFactoryName
-Name $MyAzureSsisIrName -ResourceGroupName $MyResourceGroupName
-Edition $MyAzureSsisIrEdition

Start-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $MyDataFactoryName
-Name $MyAzureSsisIrName -ResourceGroupName $MyResourceGroupName
```

# Azure-SSIS IR Pricing

| Node Size | Cores (CPU) | Memory (RAM) | Temp. Storage | Preview Prices | |
|---|---|---|---|---|---|
| | | | | Standard | Enterprise |
| Standard_A4_v2 | 4 | 8.00 GiB | 40 GiB | $0.420/hour | $0.956/hour |
| Standard_A8_v2 | 8 | 16.00 GiB | 80 GiB | $0.862/hour | $1.935/hour |
| Standard_D1_v2 | 1 | 3.50 GiB | 50 GiB | $0.296/hour | $0.832/hour |
| Standard_D2_v2 | 2 | 7.00 GiB | 100 GiB | $0.397/hour | $0.933/hour |
| Standard_D3_v2 | 4 | 14.00 GiB | 200 GiB | $0.599/hour | $1.136/hour |
| Standard_D4_v2 | 8 | 28.00 GiB | 400 GiB | $1.199/hour | $2.271/hour |

# Custom Setup Interface

# Custom Setup Interface – Introduction

- Custom Setup Interface allows you to alter the default operating configuration/environment (e.g. to start additional Windows services) and or install additional components (e.g. assemblies/drivers/extensions) on each node of your Azure-SSIS IR

- In general, it provides an interface to add your own setup steps during the provisioning/reconfiguration of your Azure-SSIS IR

- You can specify your custom setup by preparing a script + associated files, uploading them into a blob container in your Azure Storage account, and providing Shared Access Signature (SAS) Uniform Resource Identifier (URI) of your container when you provision/reconfigure your Azure-SSIS IR

- Each node of your Azure-SSIS IR will then download the script + files from your container and execute your custom setup with an elevated privilege

- Upon completion, each node will upload the standard output of execution and other logs into your container

# Custom Setup Interface – Limitations

- We only support this feature in East US/North Europe/West Europe regions for now
- We only support the installation of free/unlicensed/Enterprise Edition components for now
  - We are in discussion with various Independent Software Vendor (ISV) partners to support the installation of premium/licensed components
- We do not support scripts invoking xcopy/robocopy tools directly to copy files for now
  - Please use *start /wait cmd /c "call install.cmd > %CUSTOM_SETUP_SCRIPT_LOG_DIR%\install.cmd.log"* as a workaround where install.cmd contains scripts invoking xcopy/robocopy tool
- We do not support scripts invoking gacutil tool to install assemblies in Global Assembly Cache (GAC) for now
  - Please use gacinstall.cmd provided in our Private Preview container instead

# Custom Setup Interface – Instructions

- Please send us your Azure subscription ID that you will use for provisioning/reconfiguring your Azure-SSIS IR with this feature – We will whitelist it for preview

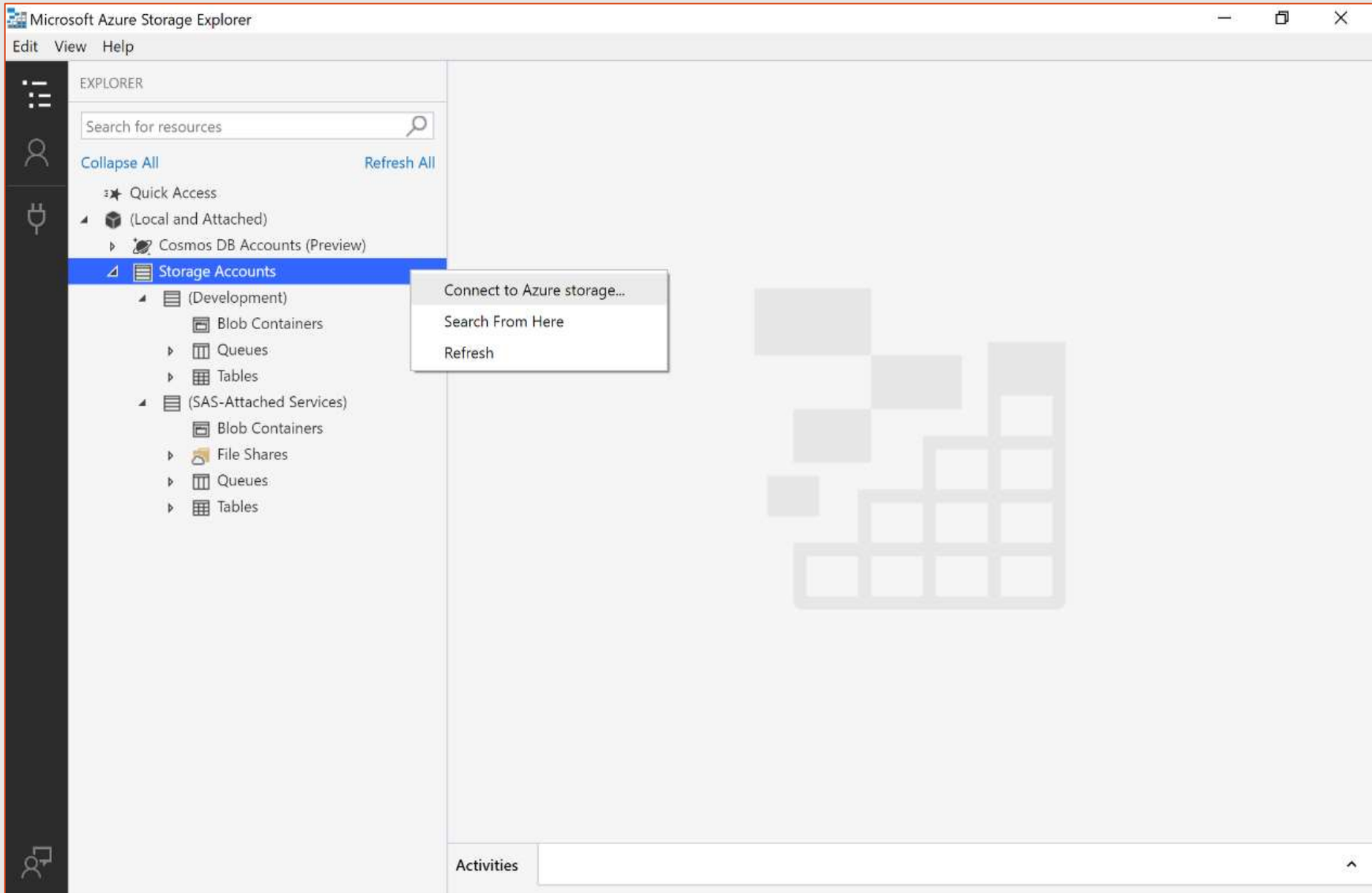
- Please download and install our private version of Azure PSH

- Please prepare your custom setup script and associated files (e.g. .bat/.cmd/.exe/.dll/.msi/.ps1 files)
  - You must have a script file named "main.cmd" that will be the entry point of your custom setup (see Limitations when preparing scripts)
  - If you want additional logs generated by some tools (e.g. msiexec) to be also uploaded into your container, please specify our predefined environment variable, CUSTOM_SETUP_SCRIPT_LOG_DIR, as the log folder in your scripts (e.g. msiexec /i xxx.msi /quiet /lv %CUSTOM_SETUP_SCRIPT_LOG_DIR%\install.log)

- Download, install, and launch Azure Storage Explorer app, if you have not done it already

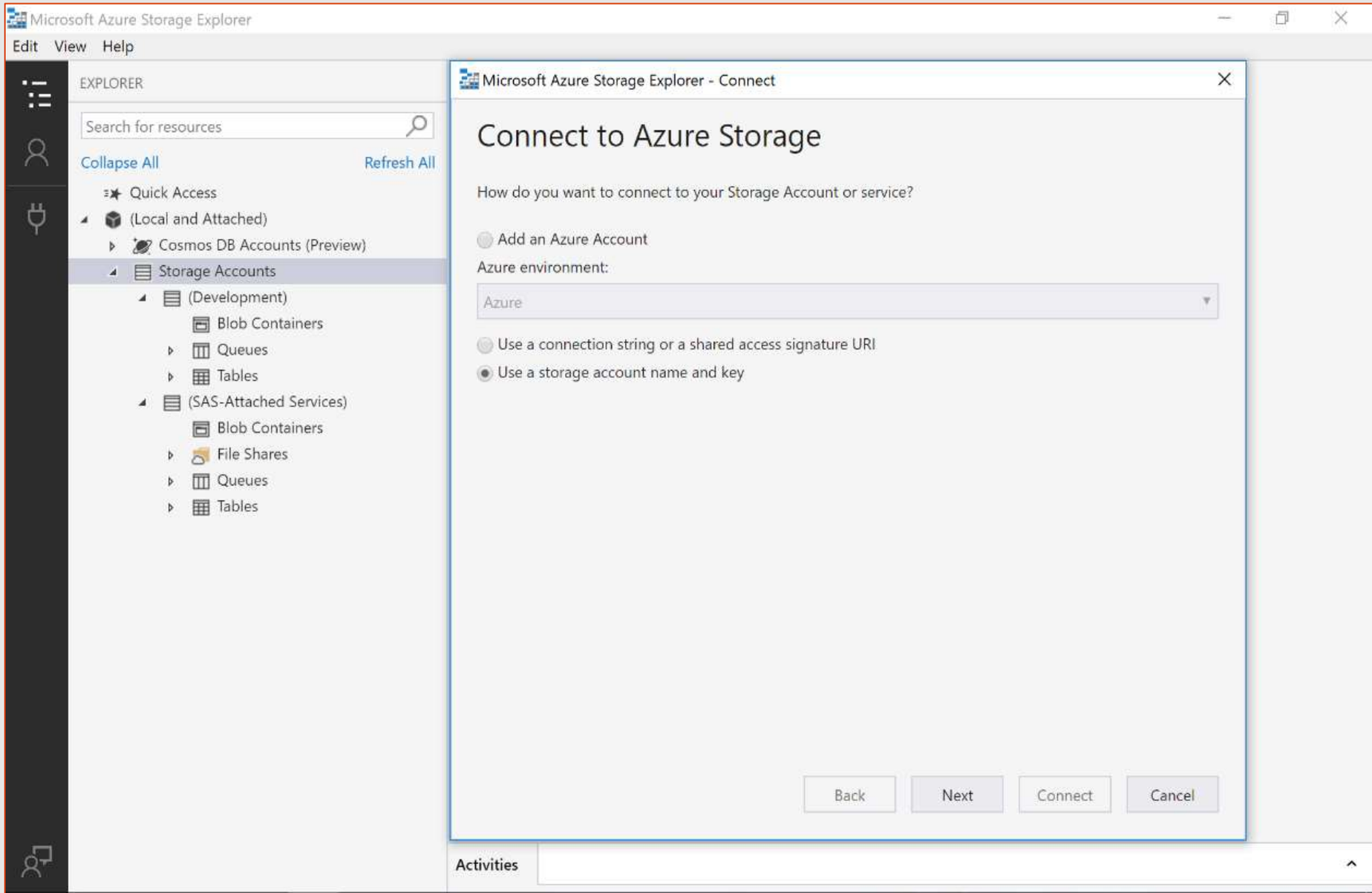
# Custom Setup Interface – Instructions

- Under “(Local and Attached)” menu item, right-click “Storage Accounts”, and select “Connect to Azure storage…



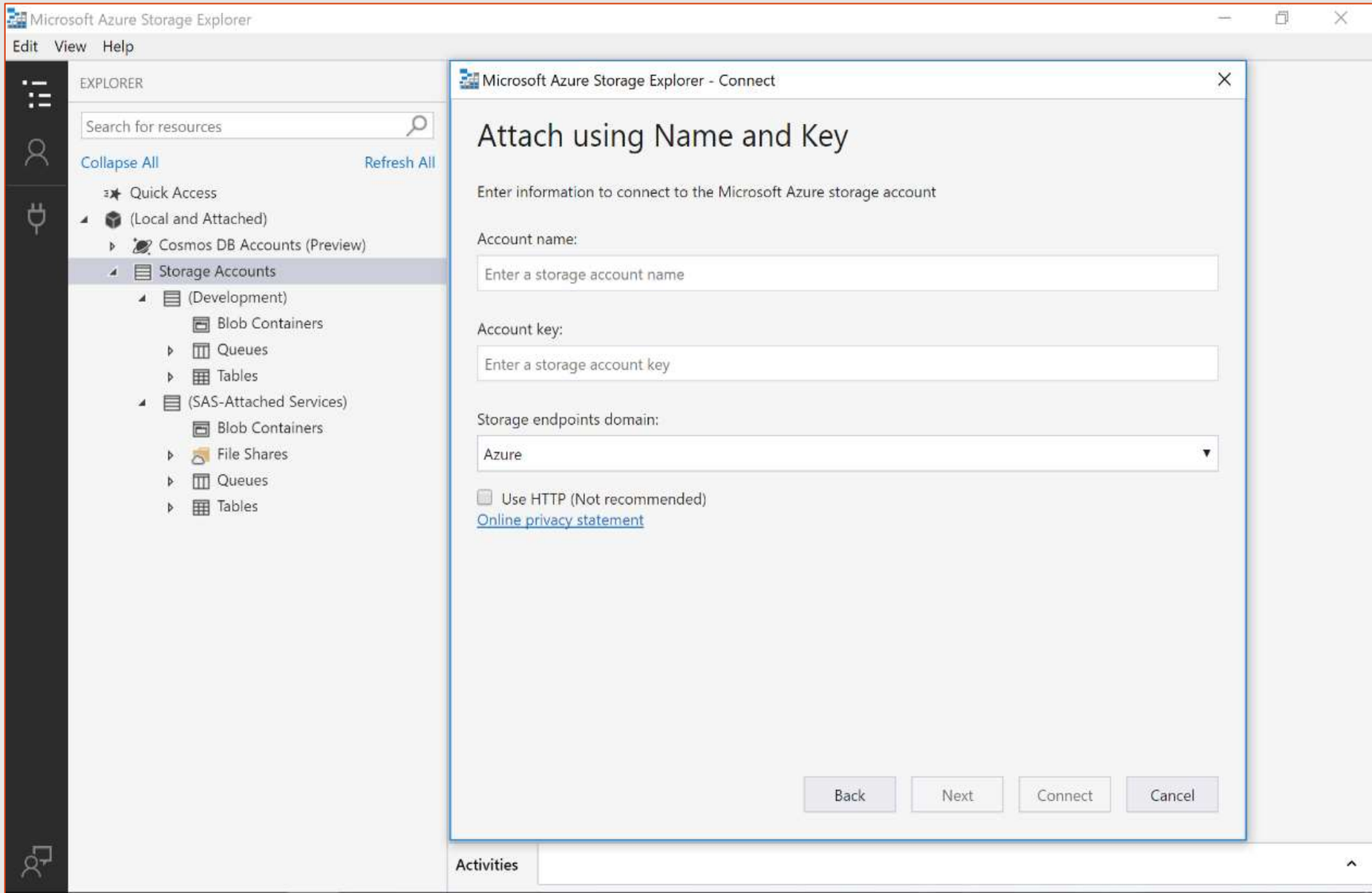
# Custom Setup Interface – Instructions

- Select “Use a storage account name and key” and click “Next”



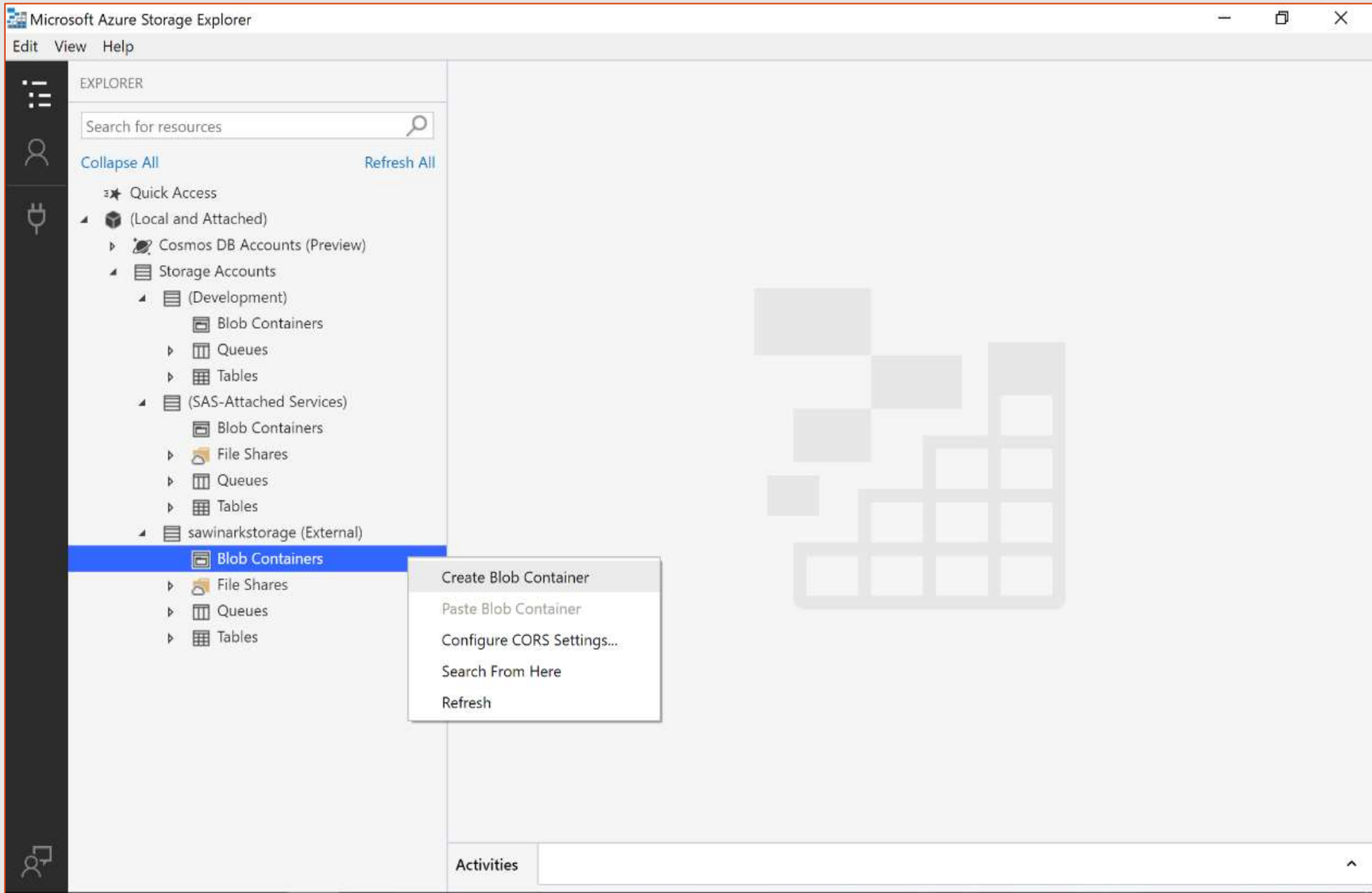
# Custom Setup Interface – Instructions

- Enter your Azure Storage account name + key, click “Next”, and click “Connect”



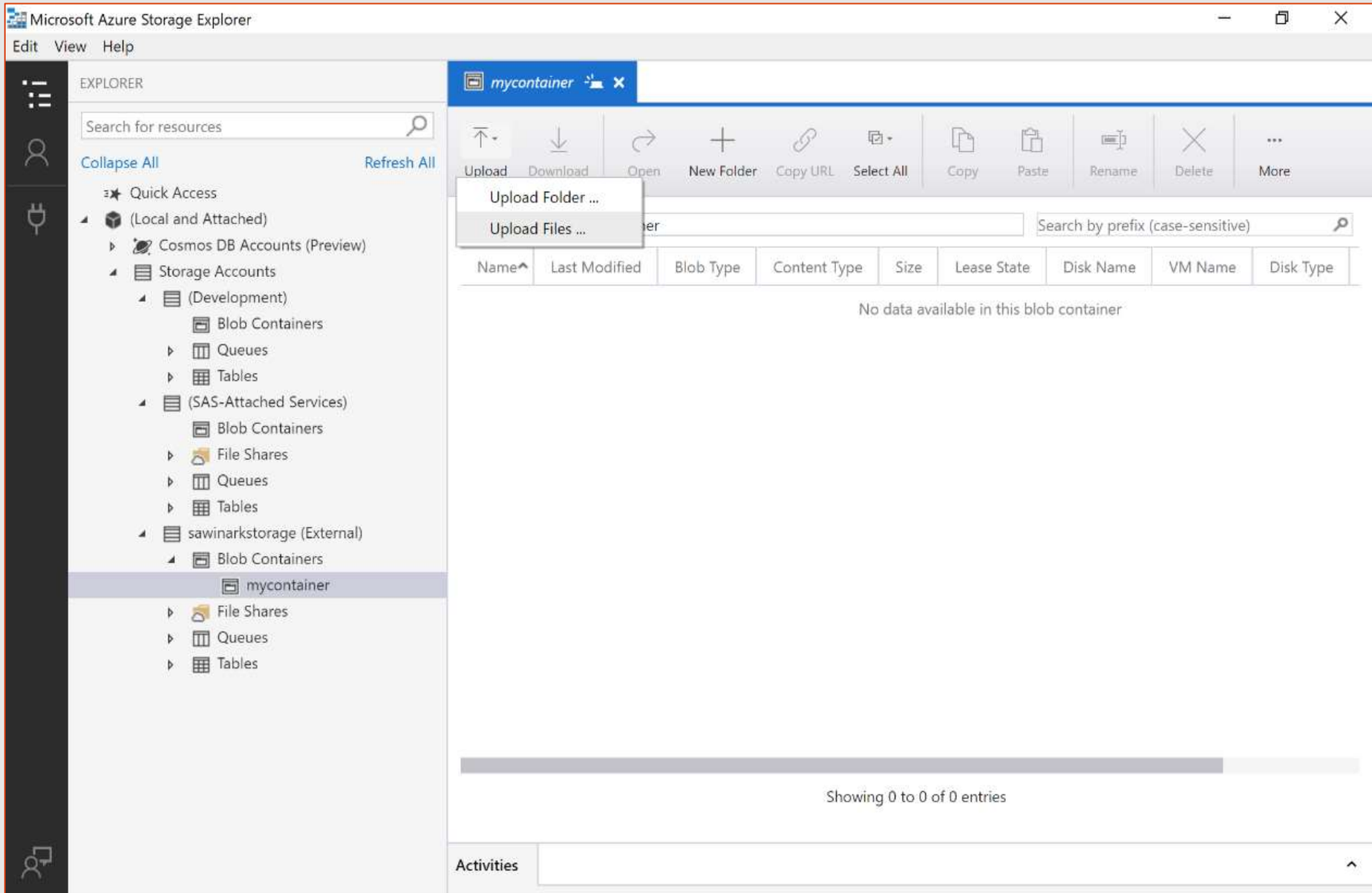
# Custom Setup Interface – Instructions

- Under your connected Azure Storage account, right-click “Blob Containers”, select “Create Blob Container”, and name your new container



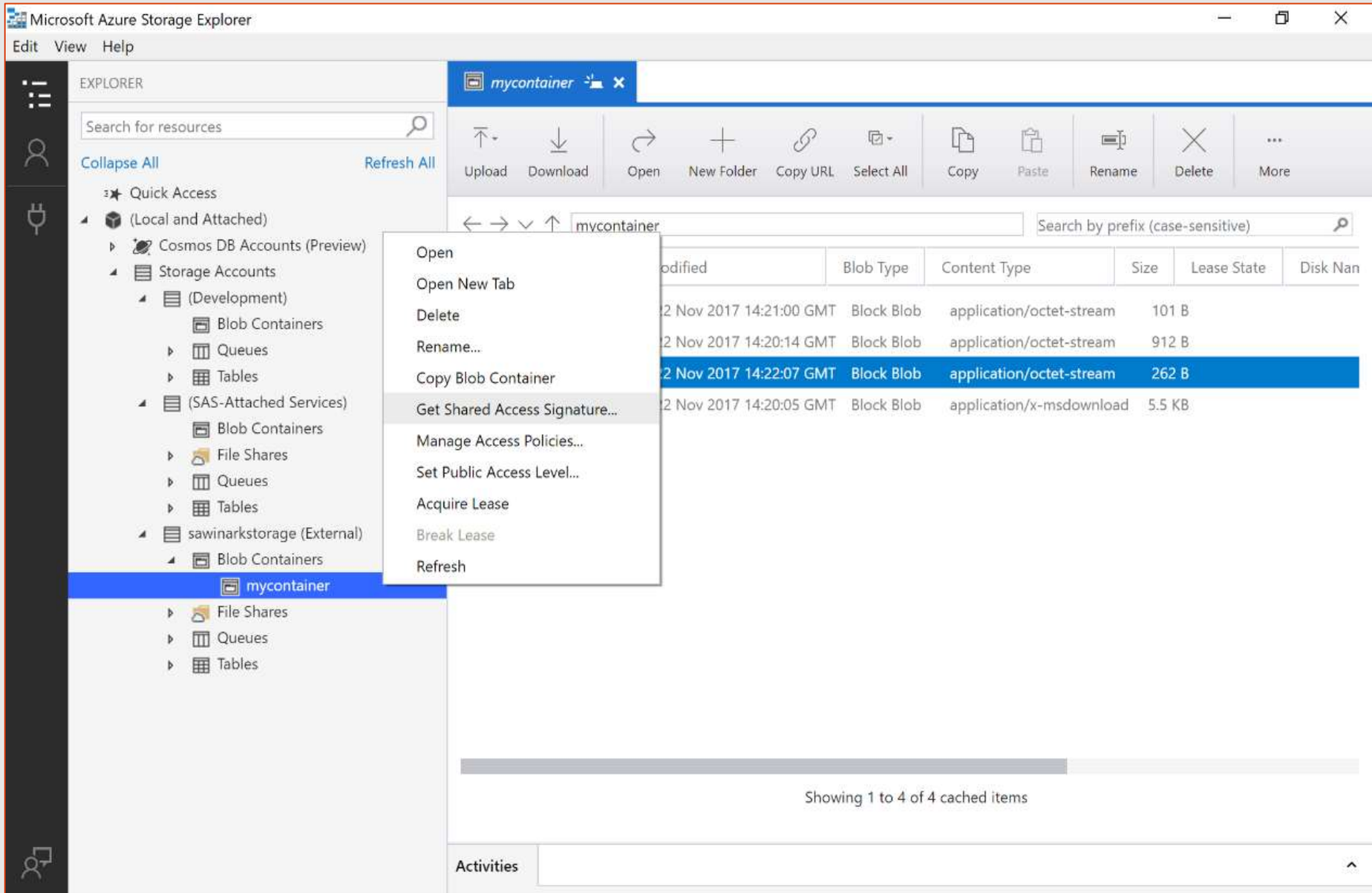
# Custom Setup Interface – Instructions

- Select your container, upload your custom setup script + associated files, and make sure main.cmd is uploaded at the top level, not in any folder, of your container



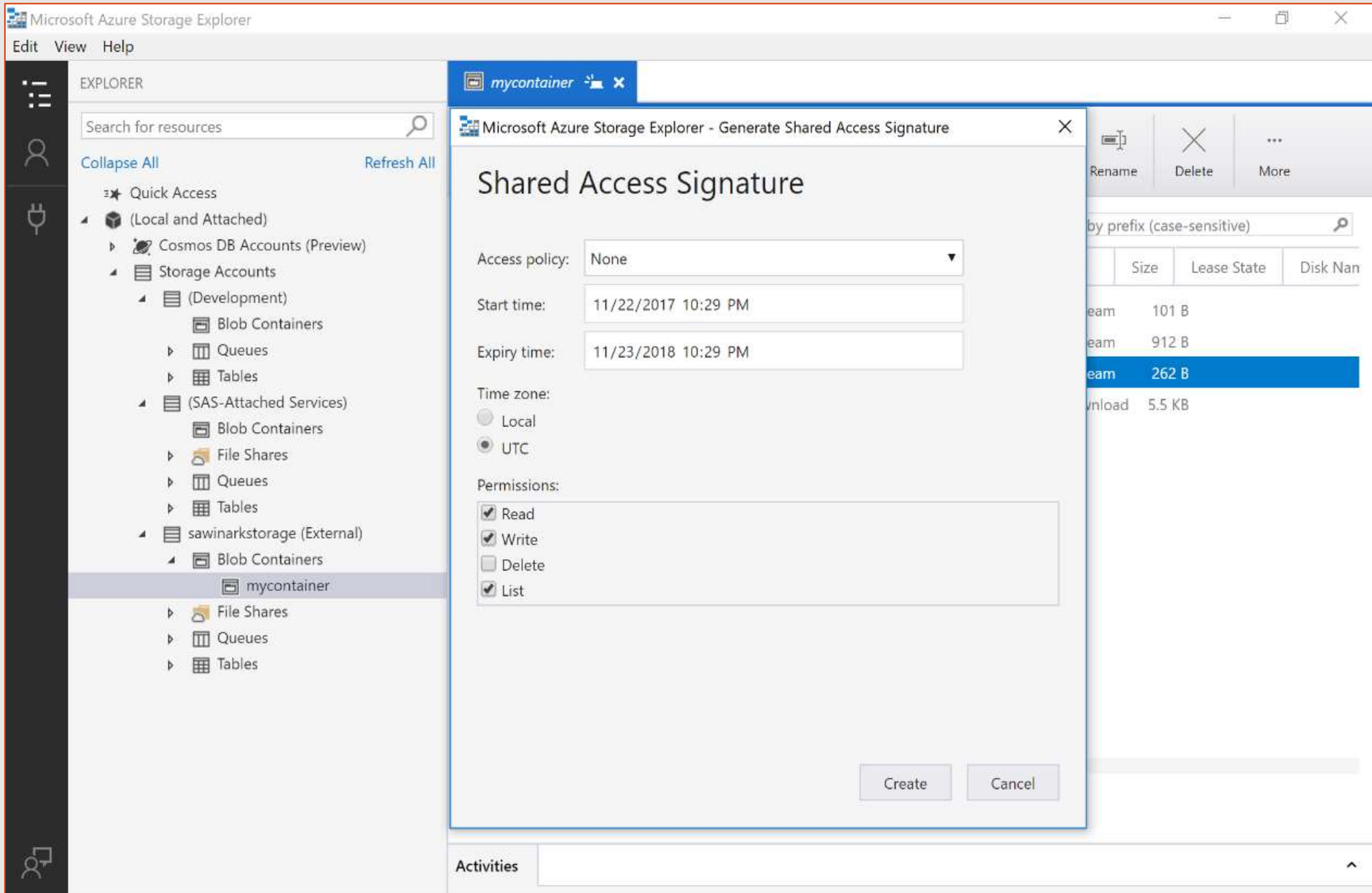
# Custom Setup Interface – Instructions

- Right-click your container and select “Get Shared Access Signature…”



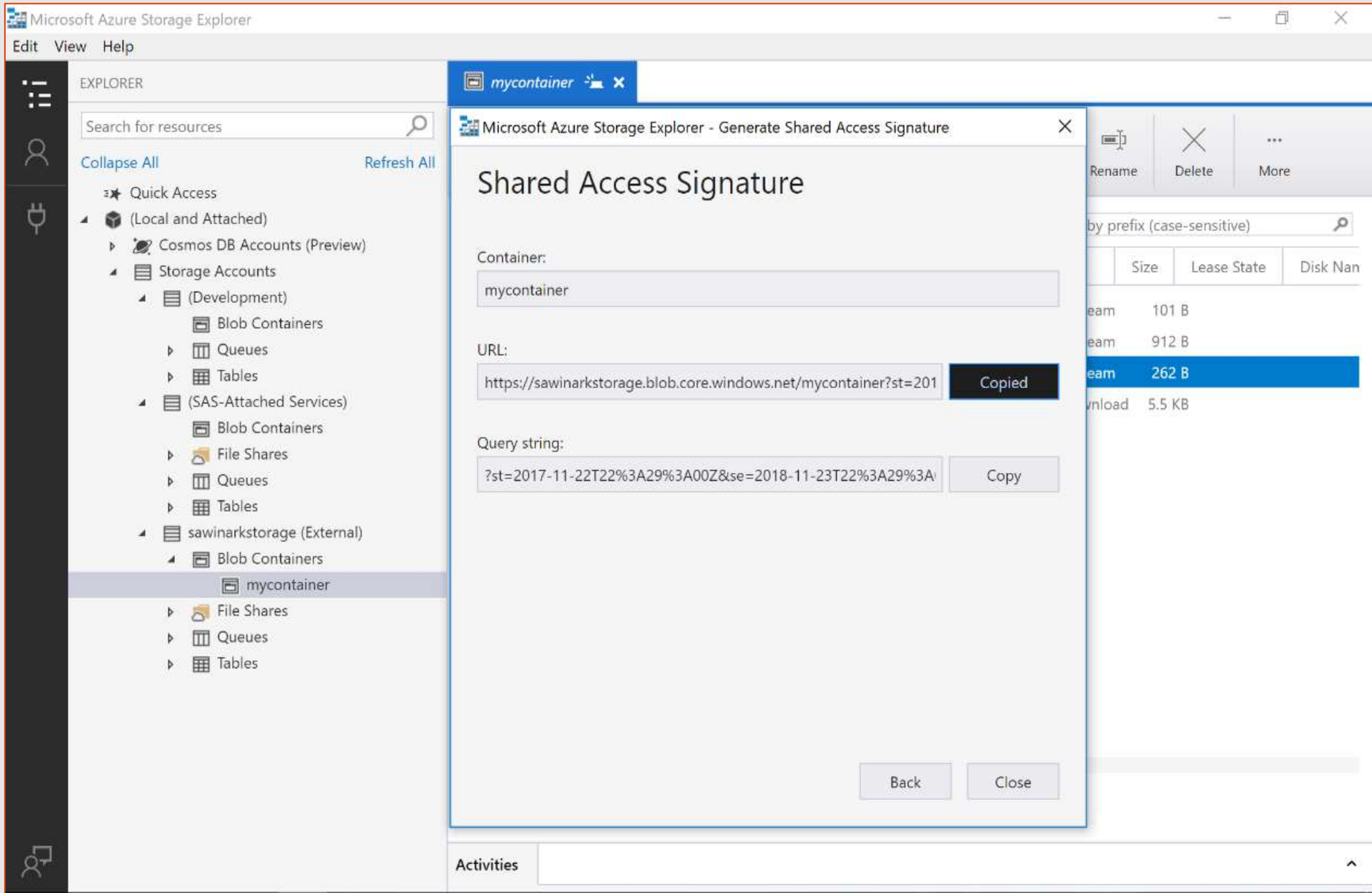
# Custom Setup Interface – Instructions

- Create SAS URI of your container with sufficiently long expiry time and read + write + list permissions, since your custom setup script + associated files need to be redownloaded and re-executed whenever any node of your Azure-SSIS IR is reimaged, while write permission is required for uploading setup execution logs



# Custom Setup Interface – Instructions

- Copy and save SAS URI of your container



# Custom Setup Interface – Instructions

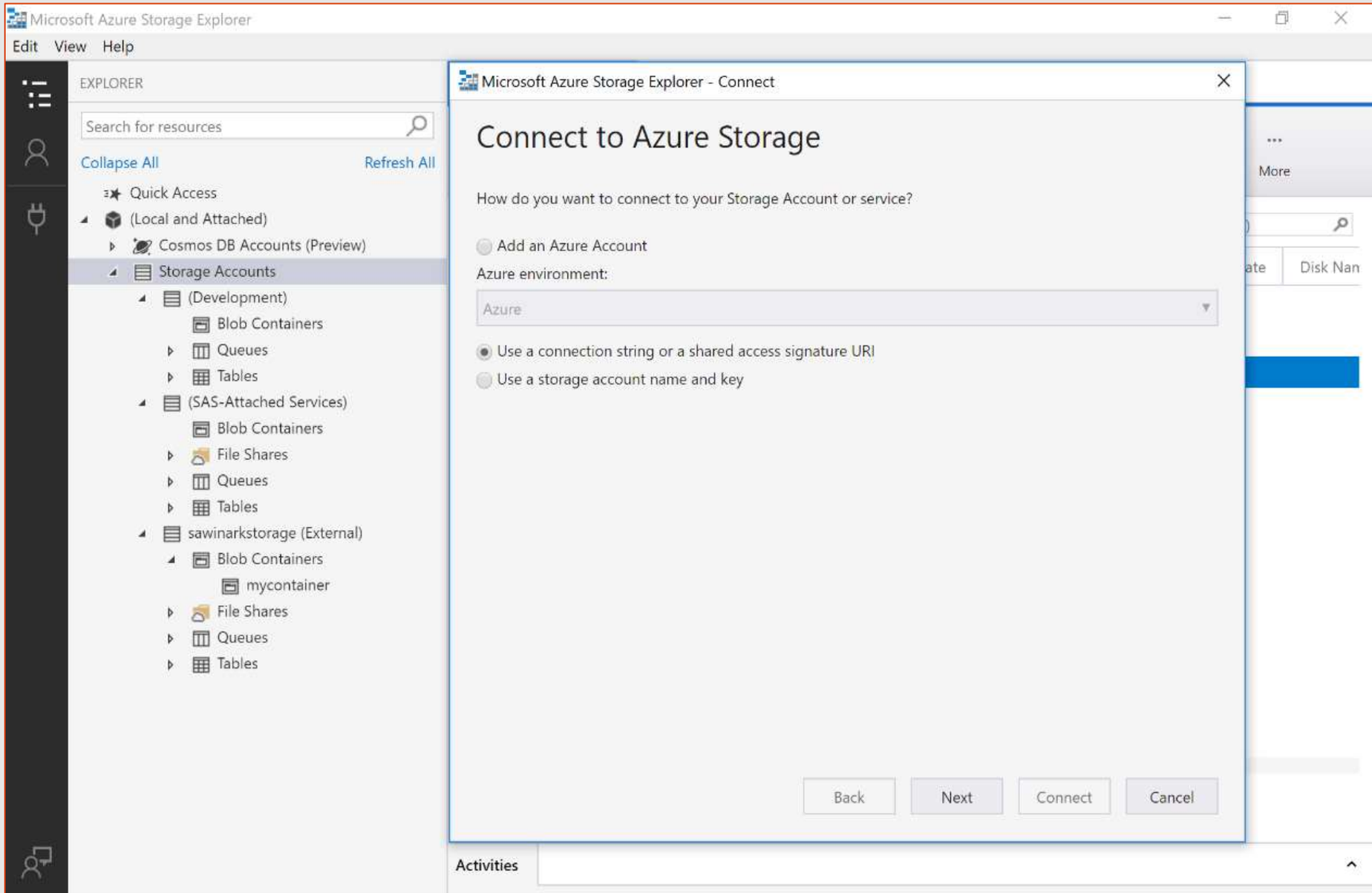
- When provisioning/reconfiguring your Azure-SSIS IR via PSH, execute Set-AzureRmDataFactoryV2IntegrationRuntime cmdlet with SAS URI of your container as the value for new SetupScriptContainerSasUri parameter before starting your Azure-SSIS IR, e.g.

```
Set-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $MyDataFactoryName
-Name $MyAzureSsisIrName -ResourceGroupName $MyResourceGroupName
-SetupScriptContainerSasUri $MySetupScriptContainerSasUri

Start-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $MyDataFactoryName
-Name $MyAzureSsisIrName -ResourceGroupName $MyResourceGroupName
```

- When your custom setup is completed/Azure-SSIS IR is started, you can find the standard output of main.cmd and other execution logs in main.cmd.log folder of your container

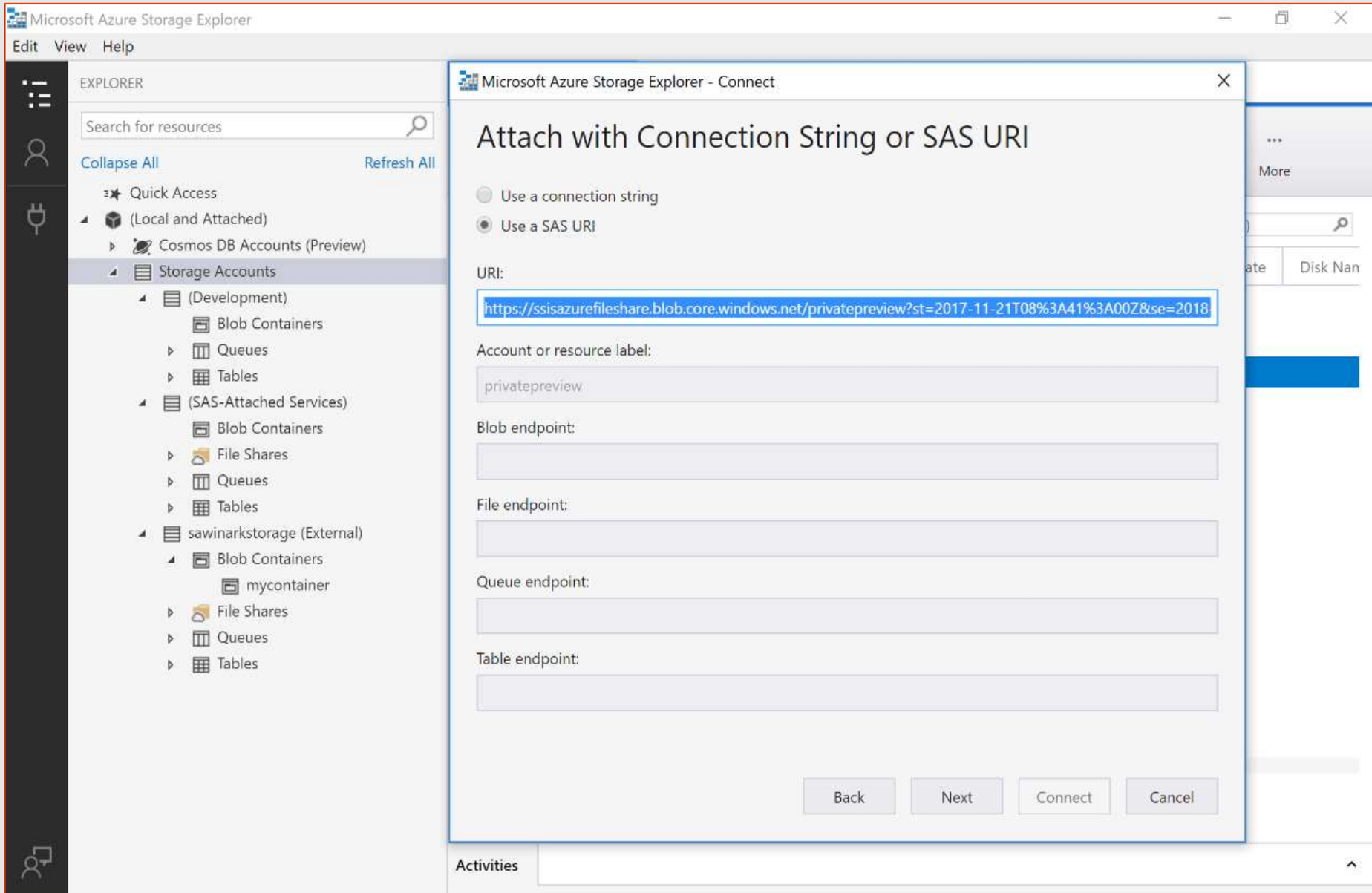
# Custom Setup Interface – Examples

- Under “(Local and Attached)” menu item, right-click “Storage Accounts”, select “Connect to Azure storage…”, select “Use a connection string or a shared access signature URI”, and click “Next”



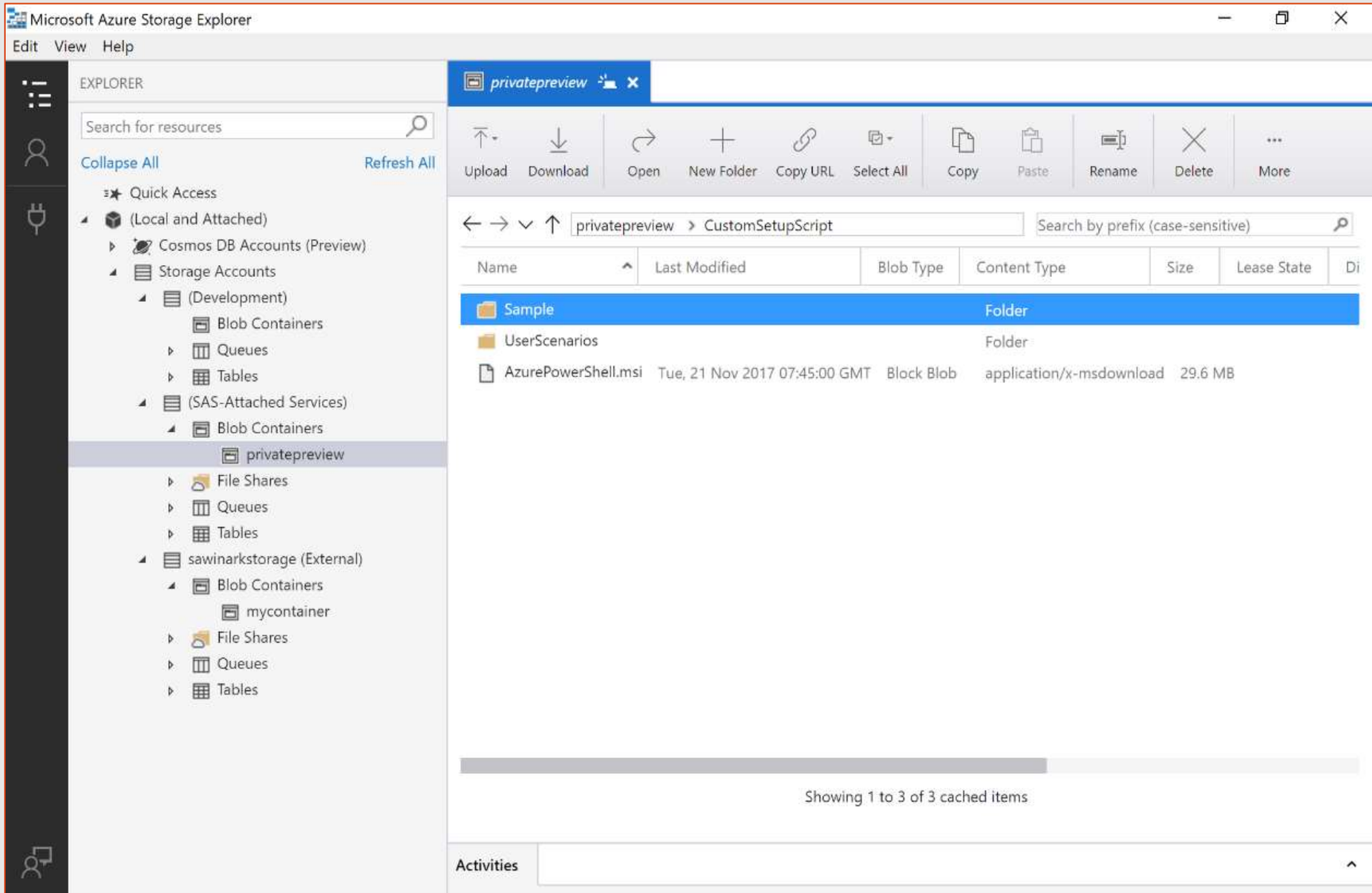
# Custom Setup Interface – Examples

- Select “Use a SAS URI”, enter SAS URI of our Private Preview container (https://ssisazurefileshare.blob.core.windows.net/privatepreview?st=2017-11-21T08%3A41%3A00Z&se=2018-11-22T08%3A41%3A00Z&sp=rl&sv=2017-04-17&sr=c&sig=kalXup1%2B4x49f57GKmUKTTFSZB%2FKRYlwN5tU%2F6Mk%2B4I%3D), click “Next”, and click “Connect”



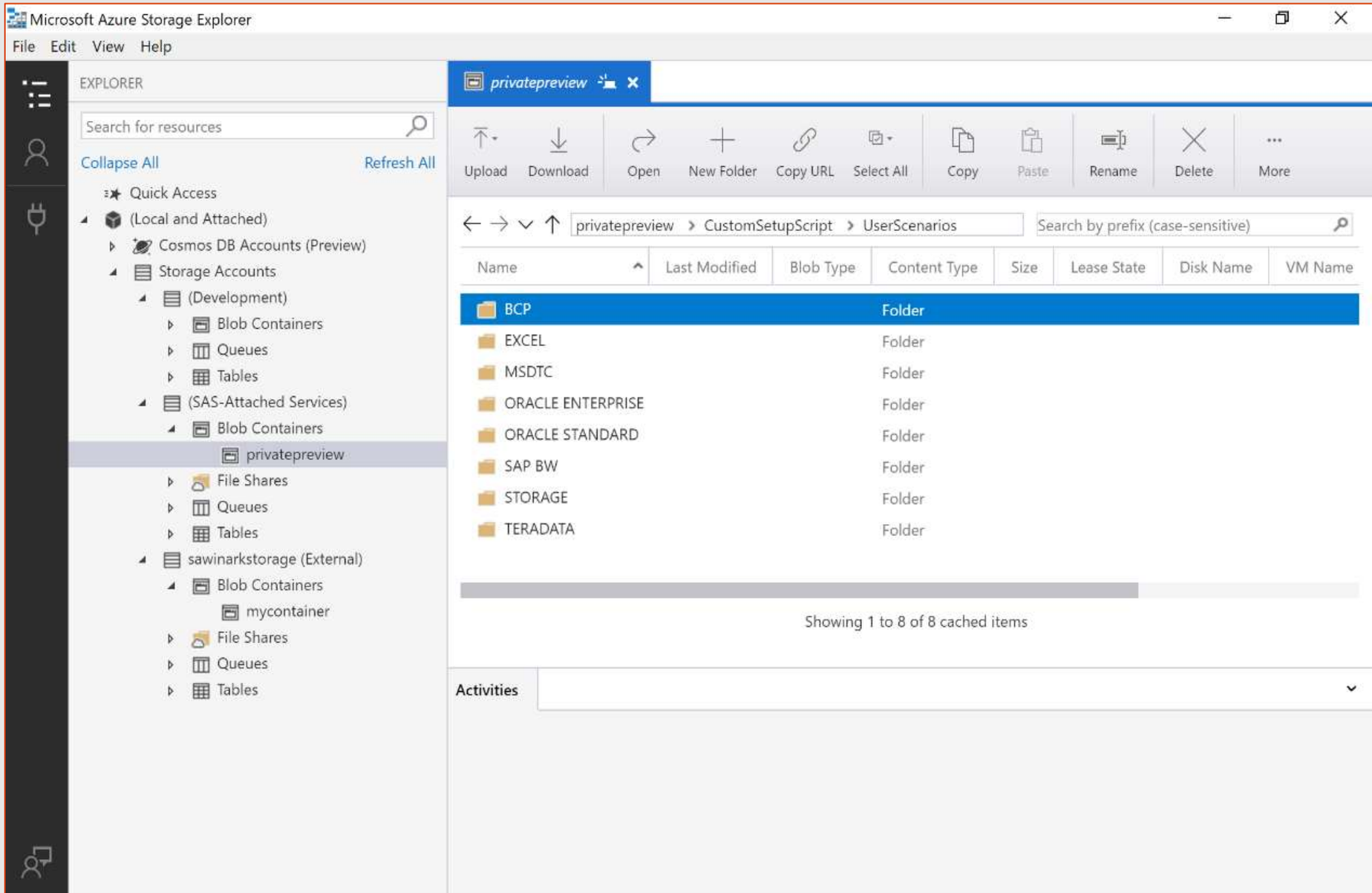
# Custom Setup Interface – Examples

- Select our connected Private Preview container, double click “CustomSetupScript”, and you can find
  - “Sample” folder that contains a custom setup to install simple task that just sleeps for a few seconds on each node of your Azure-SSIS IR – It also contains gacinstall.cmd that replaces gacutil tool (see Limitations)
  - “UserScenarios” folder that contains 8 custom setups for real user scenarios
  - AzurePowerShell.msi, which is our private version of Azure PSH



# Custom Setup Interface – Examples

- Double click "UserScenarios" and you can find
  - "BCP" folder that contains a custom setup to install SQL Server command line utilities (MsSqlCmdLnUtils.msi), including bulk copy program (bcp), on each node of your Azure-SSIS IR
  - "EXCEL" folder that contains a custom setup to install Open Source assemblies (DocumentFormat.OpenXml.dll, ExcelDataReader.DataSet.dll, and ExcelDataReader.dll) on each node of your Azure-SSIS IR



# Custom Setup Interface – Examples

- "MSDTC" folder that contains a custom setup to enable/start Microsoft Distributed Transaction Coordinator (MSDTC) service on each node of your Azure-SSIS IR

- "ORACLE ENTERPRISE" folder that contains a custom setup script (main.cmd) and silent install config file (client.rsp) to install Oracle OCI driver on each node of your Azure-SSIS IR Enterprise Edition, so you can use Oracle Connection Manager/Source/Destination – You need to first download winx64_12102_client.zip from Oracle's site and then upload it together with main.cmd and client.rsp into your container – If you use TNS to connect to Oracle, you also need to download tnsnames.ora file, then edit and upload it into your container, so it will be copied into Oracle installation folder during setup

- "ORACLE STANDARD" folder that contains a custom setup script (main.cmd) to install Oracle ODP.NET driver on each node of your Azure-SSIS IR, so you can use it with ADO.NET Connection Manager/Source/Destination – You need to first download ODP.NET_Managed_ODAC122cR1.zip from Oracle's site and then upload it together with main.cmd into your container

# Custom Setup Interface – Examples

- "SAP BW" folder that contains a custom setup script (main.cmd) to install SAP .NET connector assembly (librfc32.dll) on each node of your Azure-SSIS IR Enterprise Edition, so you can use SAP BW Connection Manager/Source/Destination – You need to first upload the 64-bit/32-bit version of librfc32.dll from SAP installation folder into your container together with main.cmd that will then copy it into %windir%\System32/%windir%\SysWow64 folder, respectively, during setup

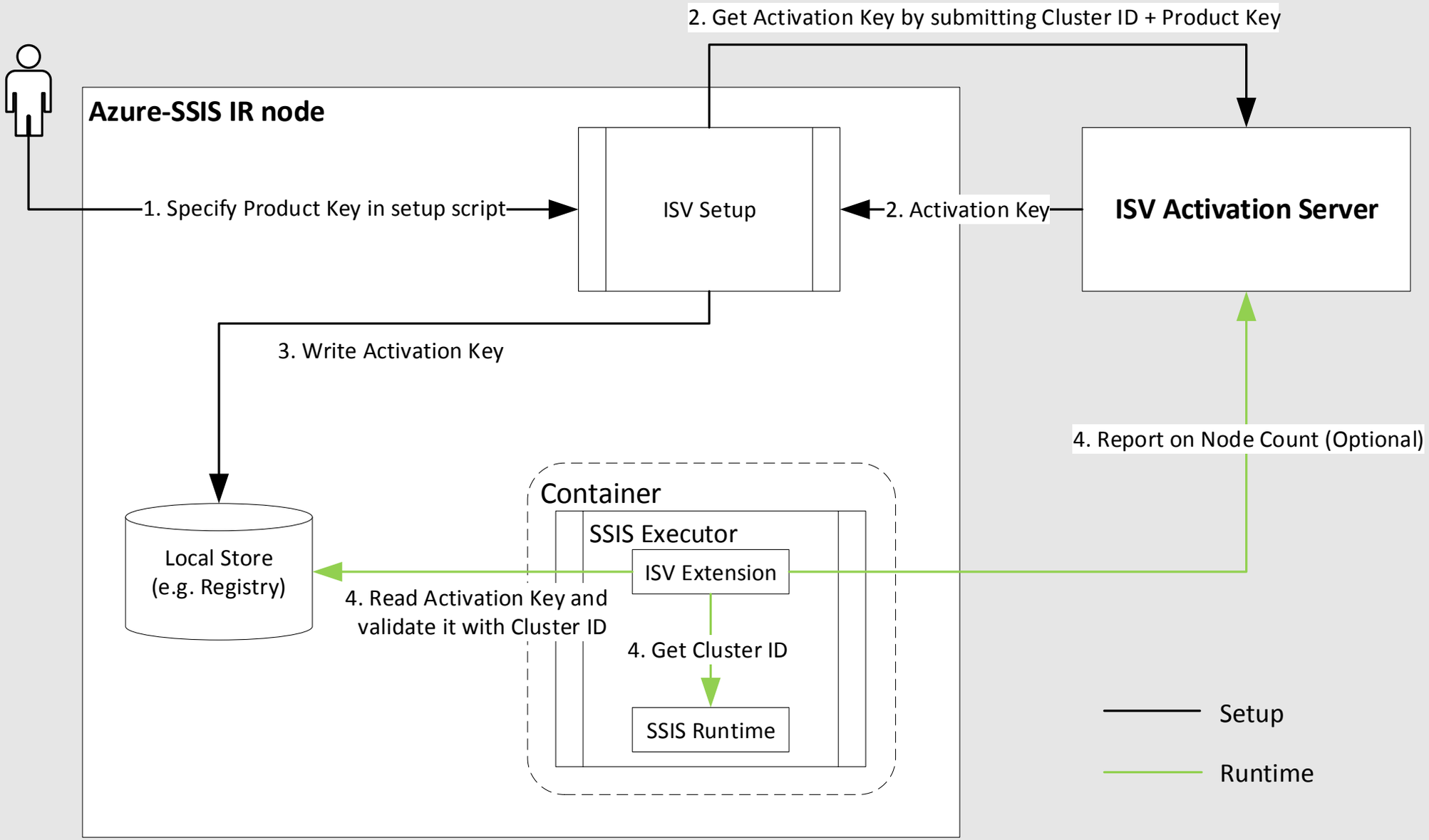
- "STORAGE" folder that contains a custom setup to install Azure PSH on each node of your Azure-SSIS IR, so you can deploy and run SSIS packages that execute PSH scripts to manipulate your Azure Storage account – You need to copy main.cmd + AzurePowerShell.msi + storage.ps1 to your container and use PowerShell.dtsx as a template for your packages, combining Azure Blob Download Task that downloads storage.ps1 as a modifiable PSH script and Execute Process Task that executes it on each node

- "TERADATA" folder that contains a custom setup script (main.cmd), its associated file (install.cmd), and installer package (.msi) files to install Teradata connectors + TPT API + ODBC driver on each node of your Azure-SSIS IR Enterprise Edition, so you can use Teradata Connection Manager/Source/Destination – You need to first download Teradata Tools and Utilities (TTU) 15.x zip file (e.g. TeradataToolsAndUtilitiesBase__windows_indep.15.10.22.00.zip) from Teradata's site and then upload it together with the above .cmd/.msi files into your container

# Custom Setup Interface – Licensing

- To support the installation of premium/licensed components from our ISV partners, we face challenges from the nature of Azure-SSIS IR:
  - The nodes of Azure-SSIS IR are volatile in the sense that they can be allocated/released at any time, e.g. customers can start/stop their nodes to manage the running cost or scale up/down through various node sizes as they see fit, so binding a component installation to any particular node by collecting machine-specific info that is traditionally used for on-premises installations, e.g. MAC address, CPU ID, etc. isn't viable
  - Customers can also scale in/out their Azure-SSIS IR, so that the number of nodes can shrink/expand at any time between 1 – 10 (or more by a special request)
- Consequently, we propose to introduce new system variables in SSIS runtime that can be referenced by ISV components as the unique/persistent info for Azure-SSIS IR, e.g. Cluster ID and Cluster Node Count
- ISVs can now bind their component installations to Azure-SSIS IR as a cluster, whose ID is invariant when customers start/stop, scale up/down, scale in/out, or reconfigure their Azure-SSIS IR in any way

# Custom Setup Interface – Licensing



# Custom Setup Interface – Licensing

- ISVs can offer their components in various SKUs/tiers (e.g. single node, up-to-5 nodes, up-to-10 nodes, etc.) and provide the corresponding Product Key when customers purchase any of them
- They can also provide Azure Storage blob container containing ISV Setup script and associated files that customers can copy & paste into their own container and modify with their own purchased Product Key (e.g. IsvSetup.exe -pid xxxx-xxxx-xxxx)
- Customers can then provision/reconfigure their Azure-SSIS IR with SAS URI of their container as parameter
- As Azure-SSIS IR is being provisioned/reconfigured, ISV Setup will be executed on each of its nodes to submit its Cluster ID and the purchased Product Key to ISV Activation Server that will generate an Activation Key
- After receiving an Activation Key, ISV Setup can store it locally on the node (e.g. in Registry)
- When customers use ISV Extension in their packages that run on a node, it will read the locally stored Activation Key and validate it with the node's Cluster ID
- It can also optionally report Cluster Node Count to ISV Activation Server

# Custom Setup Interface – Licensing

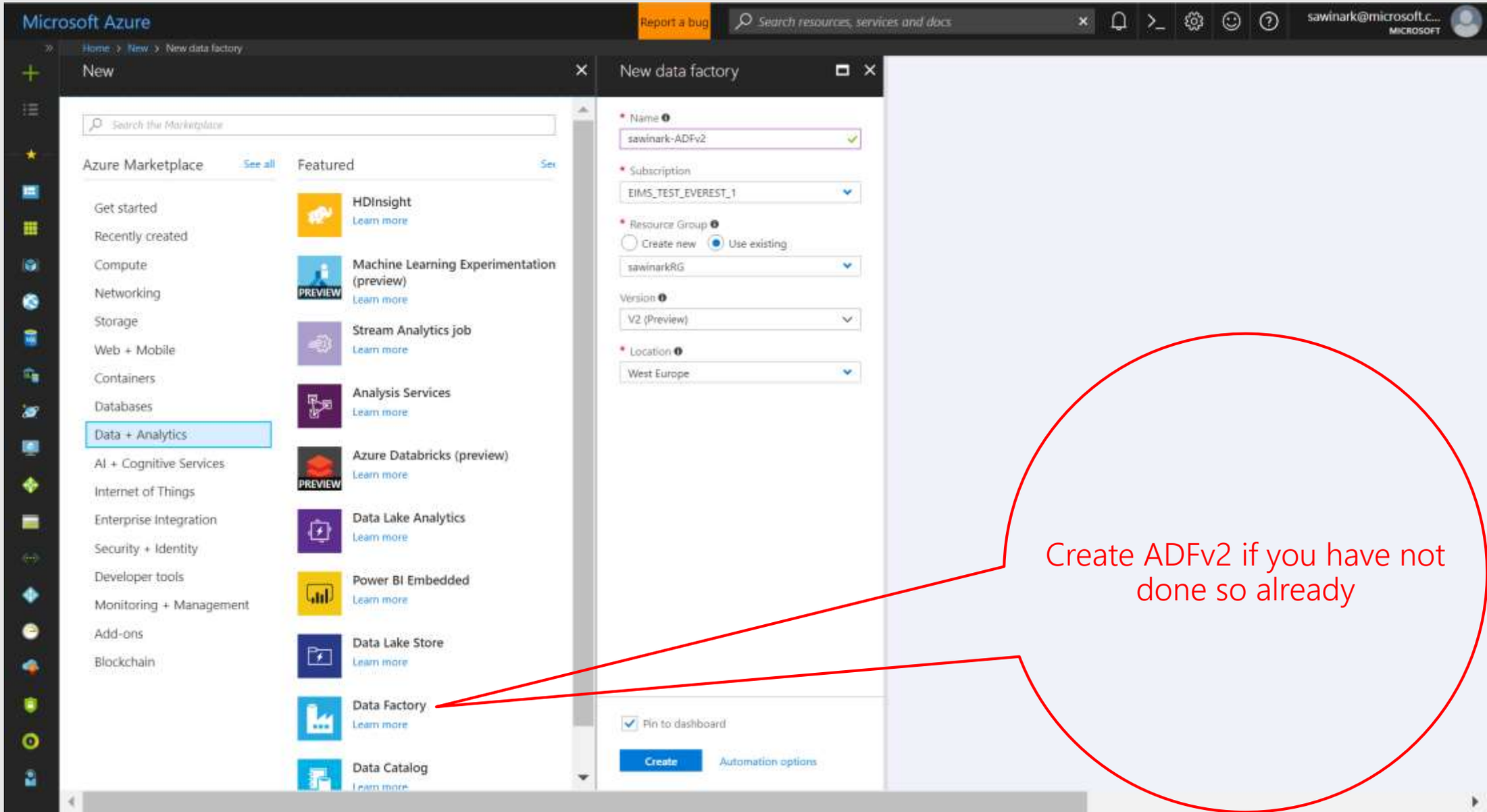
```
// Validation code example
public override DTSExecResult Validate(Connections, VariableDispenser,
IDTSComponentEvents componentEvents, IDTSLogging log)
{
    Variables vars = null;
    variableDispenser.LockForRead("System::ClusterId");
    variableDispenser.LockForRead("System::ClusterNodeCount");
    variableDispenser.GetVariables(ref vars);

    // Validate Activation Key with ClusterId
    // Report on ClusterNodeCount

    vars.Unlock();

    return base.Validate(connections, variableDispenser, componentEvents, log);
}
```

# Provisioning Methods

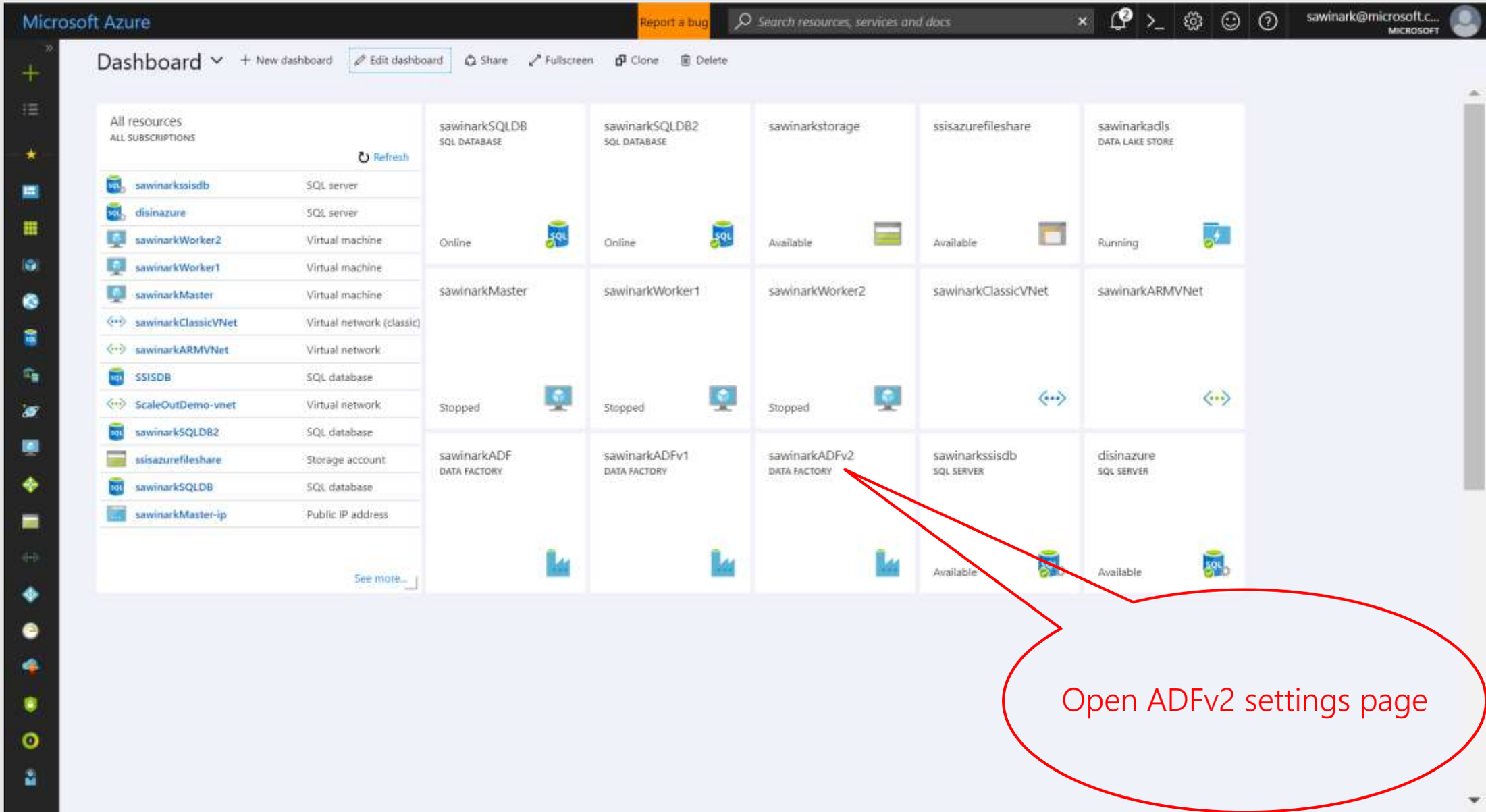
# Provisioning Methods

- Azure-SSIS IR can be provisioned via ADFv2 App
- Azure-SSIS IR can be provisioned via PowerShell (PSH)/custom code using ADFv2 .NET SDK/API

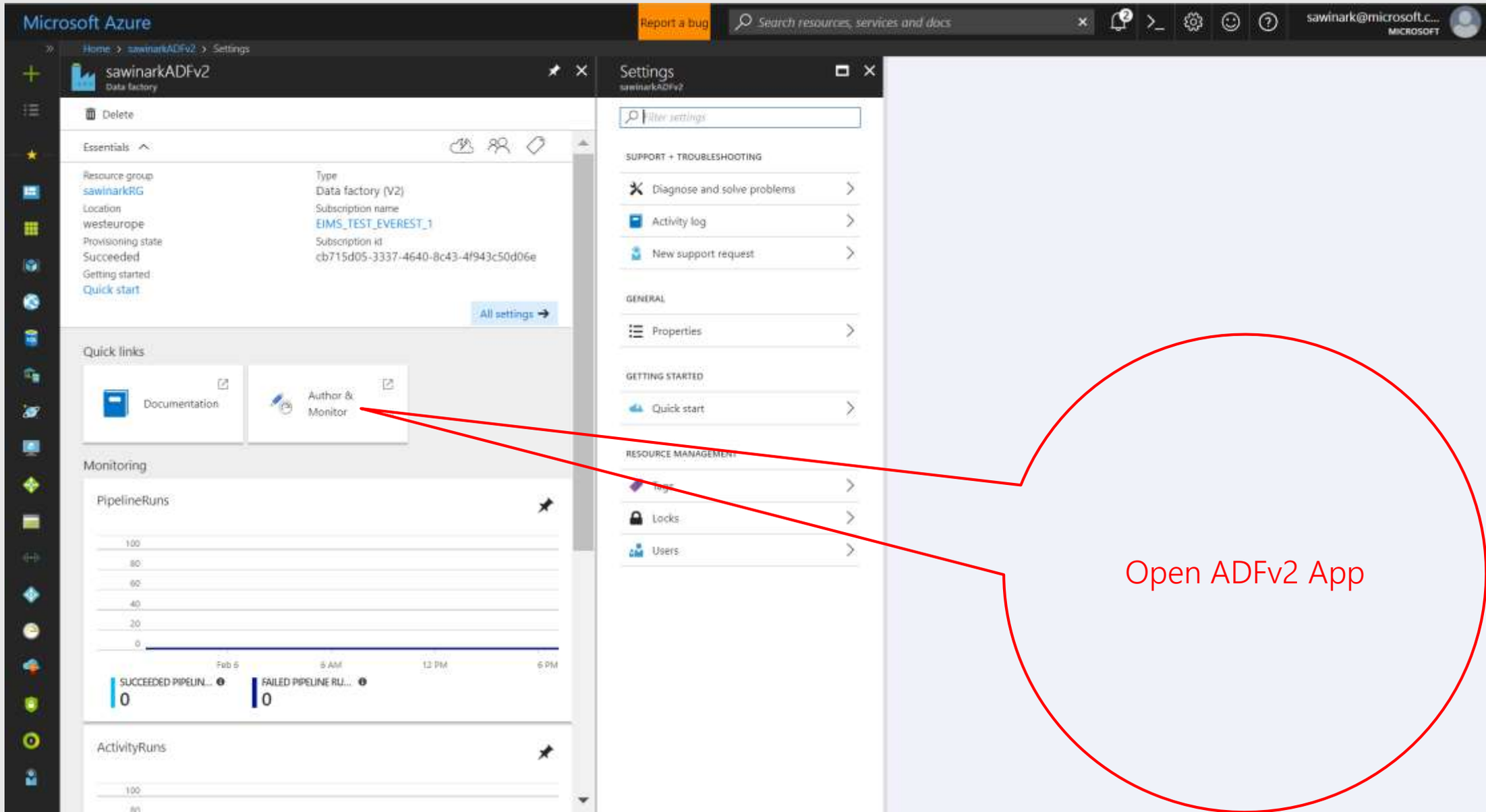
# Provisioning via ADFv2 App



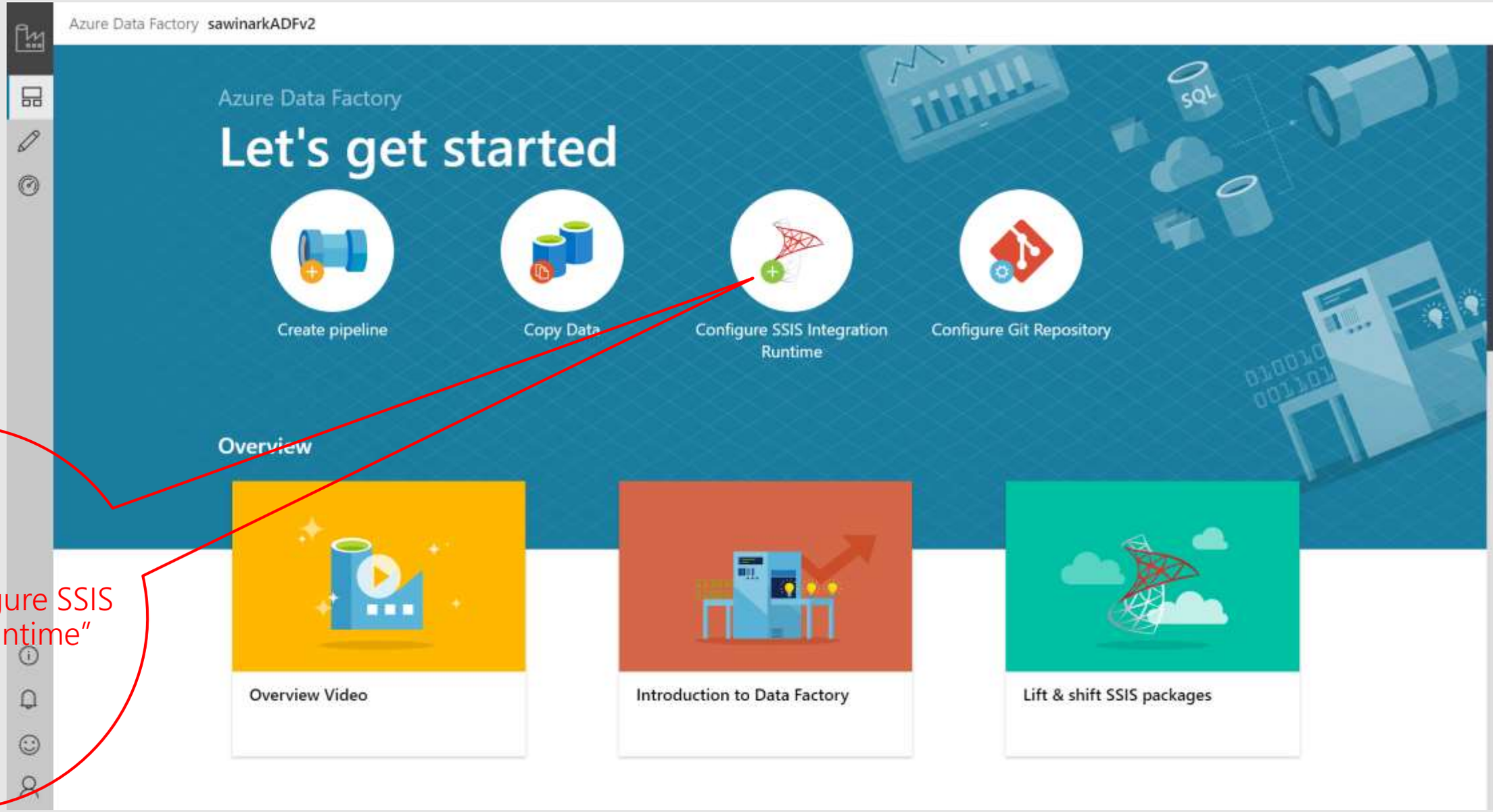
# Provisioning via ADFv2 App



# Provisioning via ADFv2 App

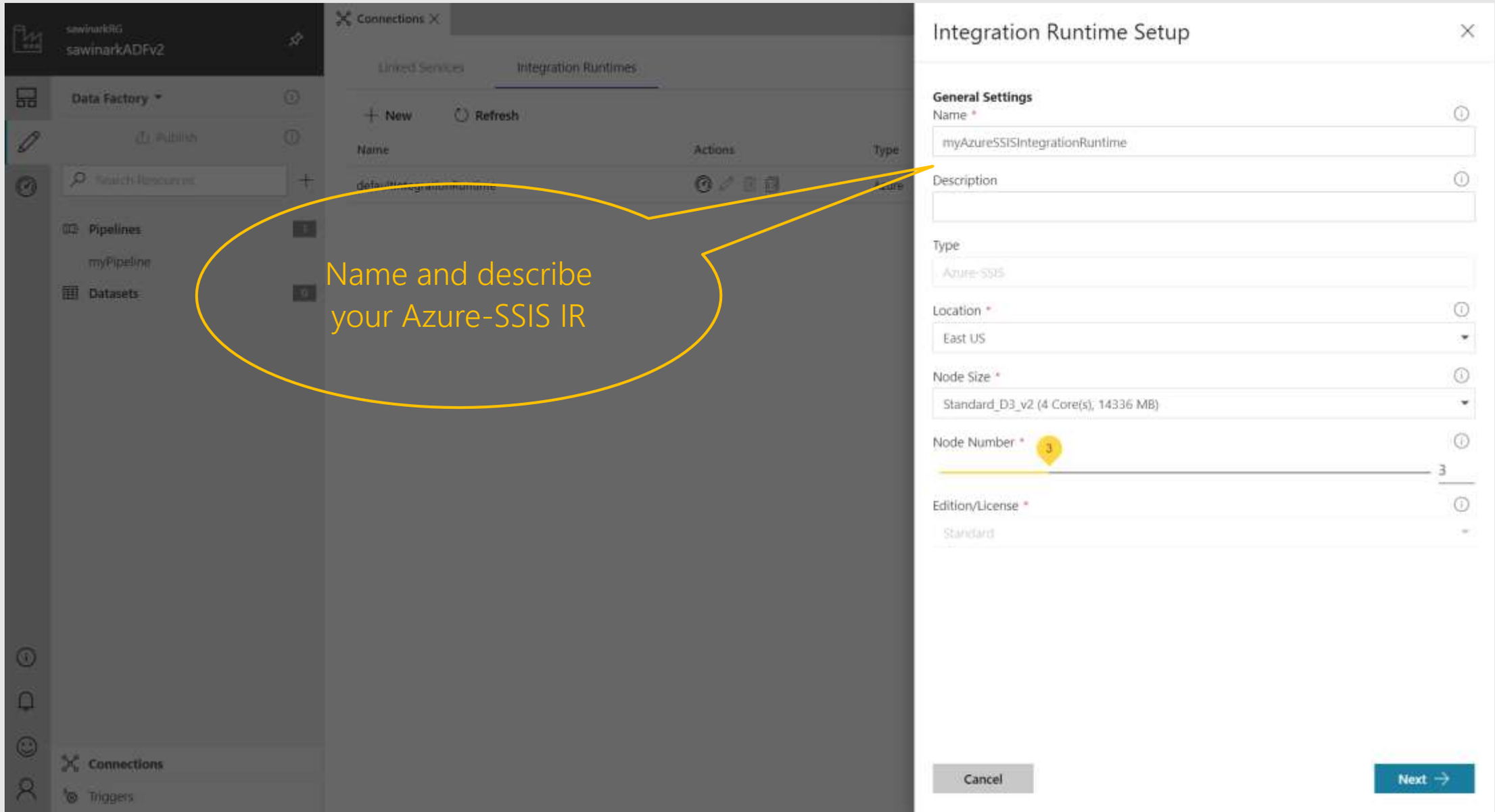


# Provisioning via ADFv2 App

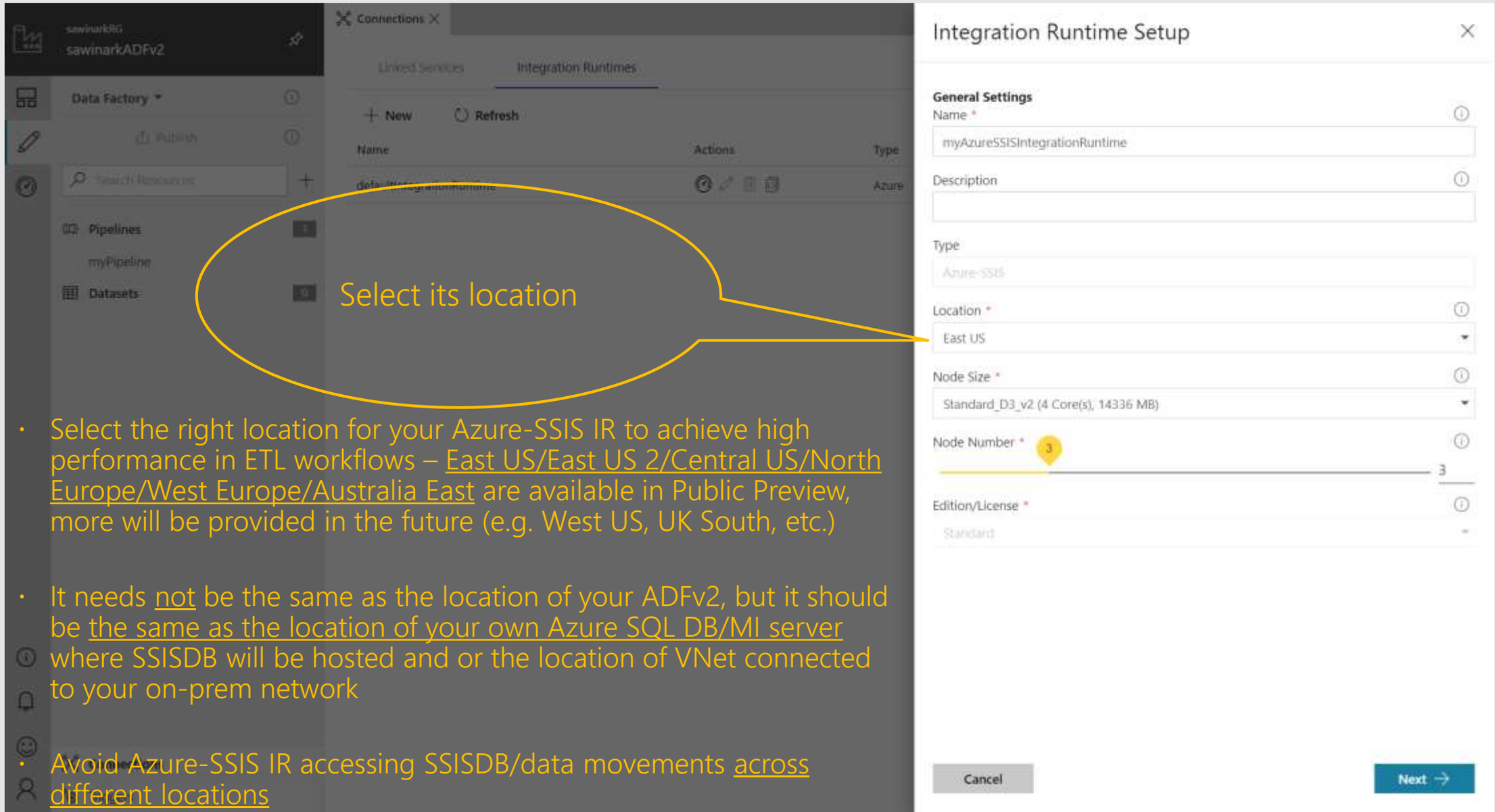



Click on "Configure SSIS Integration Runtime"

# Provisioning via ADFv2 App

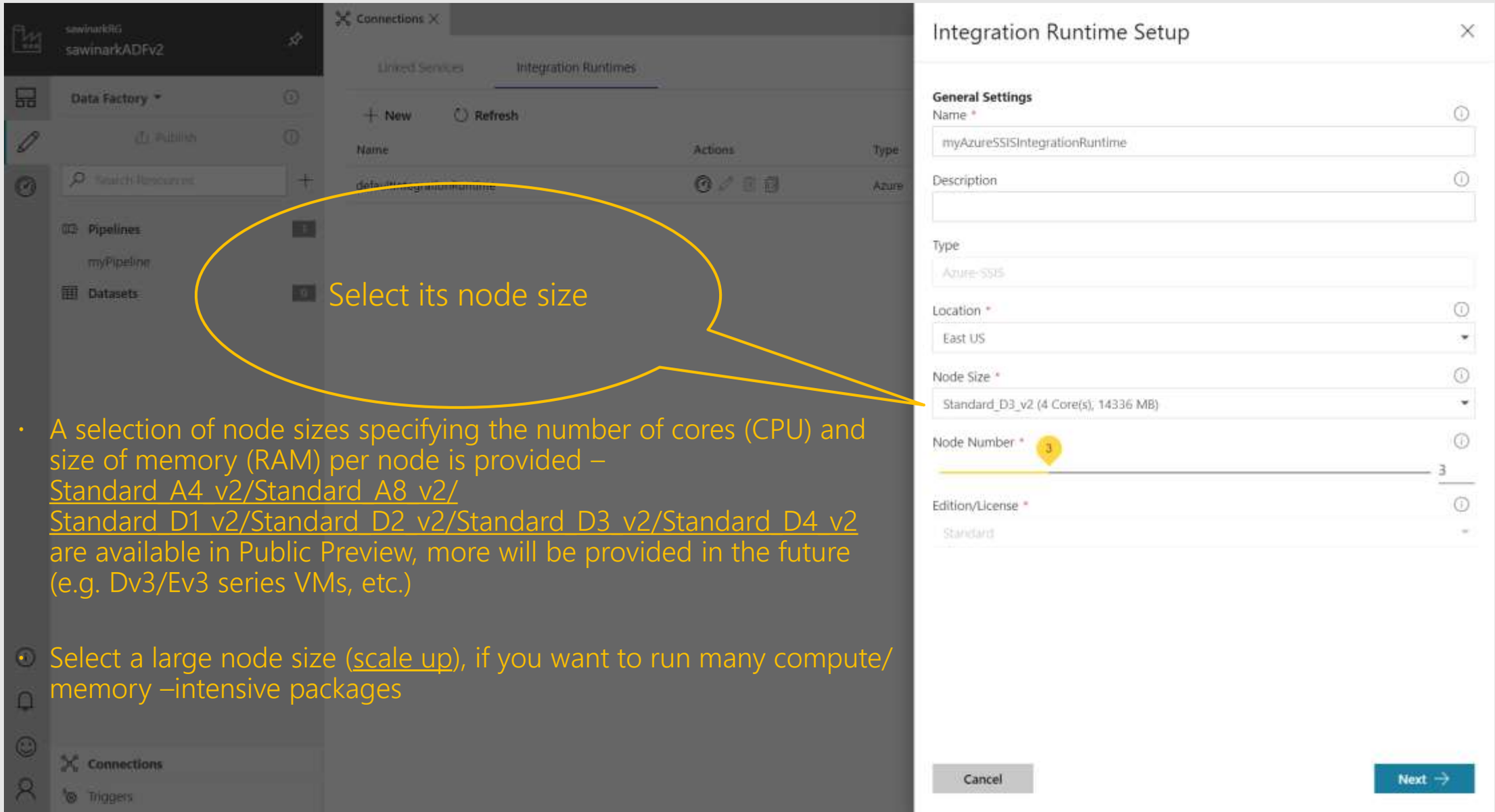


# Provisioning via ADFv2 App




- Select the right location for your Azure-SSIS IR to achieve high performance in ETL workflows – East US/East US 2/Central US/North Europe/West Europe/Australia East are available in Public Preview, more will be provided in the future (e.g. West US, UK South, etc.)
- It needs not be the same as the location of your ADFv2, but it should be the same as the location of your own Azure SQL DB/MI server where SSISDB will be hosted and or the location of VNet connected to your on-prem network
- Avoid Azure-SSIS IR accessing SSISDB/data movements across different locations

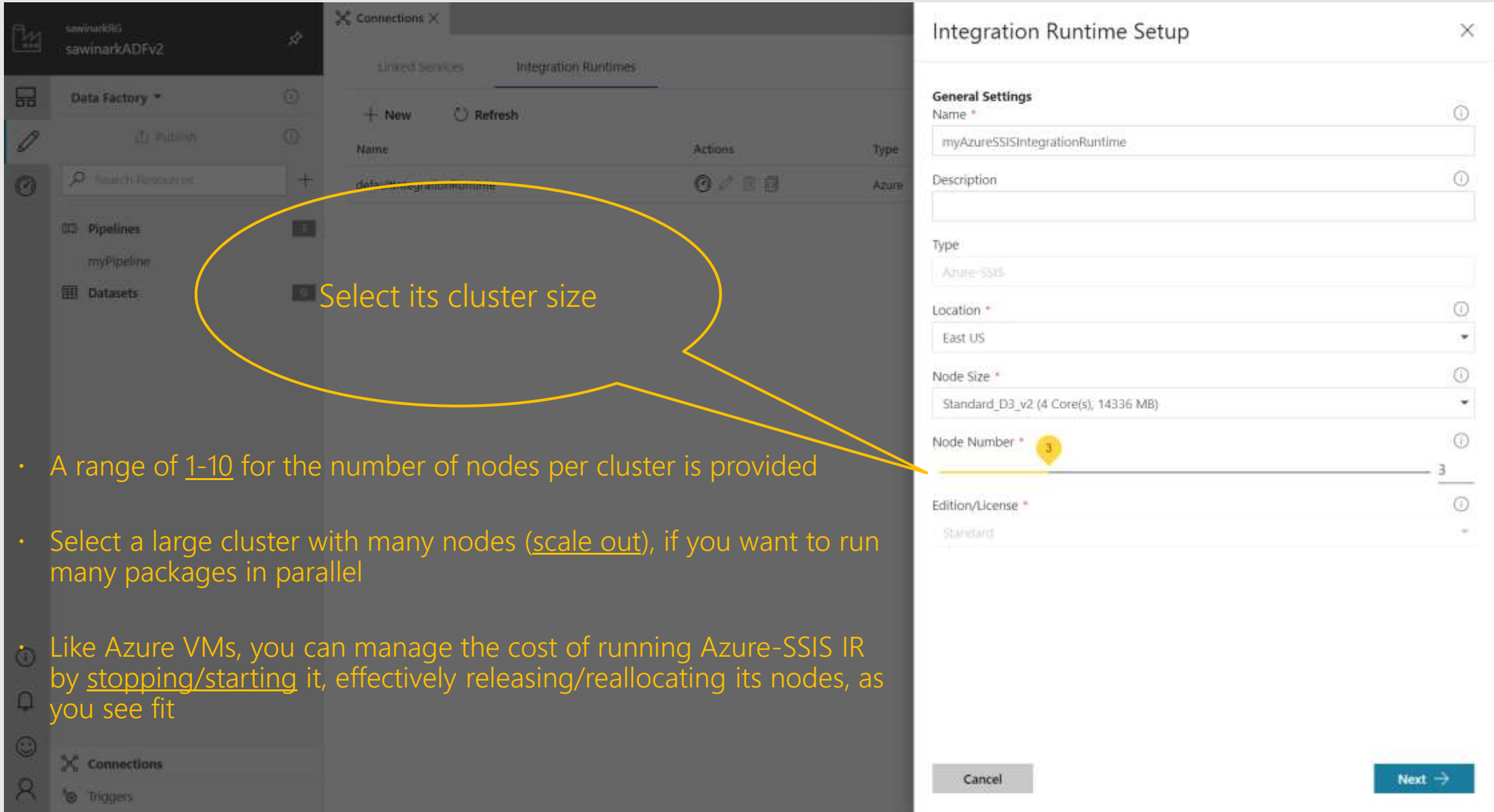
# Provisioning via ADFv2 App




- A selection of node sizes specifying the number of cores (CPU) and size of memory (RAM) per node is provided – Standard_A4_v2/Standard_A8_v2/ Standard_D1_v2/Standard_D2_v2/Standard_D3_v2/Standard_D4_v2 are available in Public Preview, more will be provided in the future (e.g. Dv3/Ev3 series VMs, etc.)

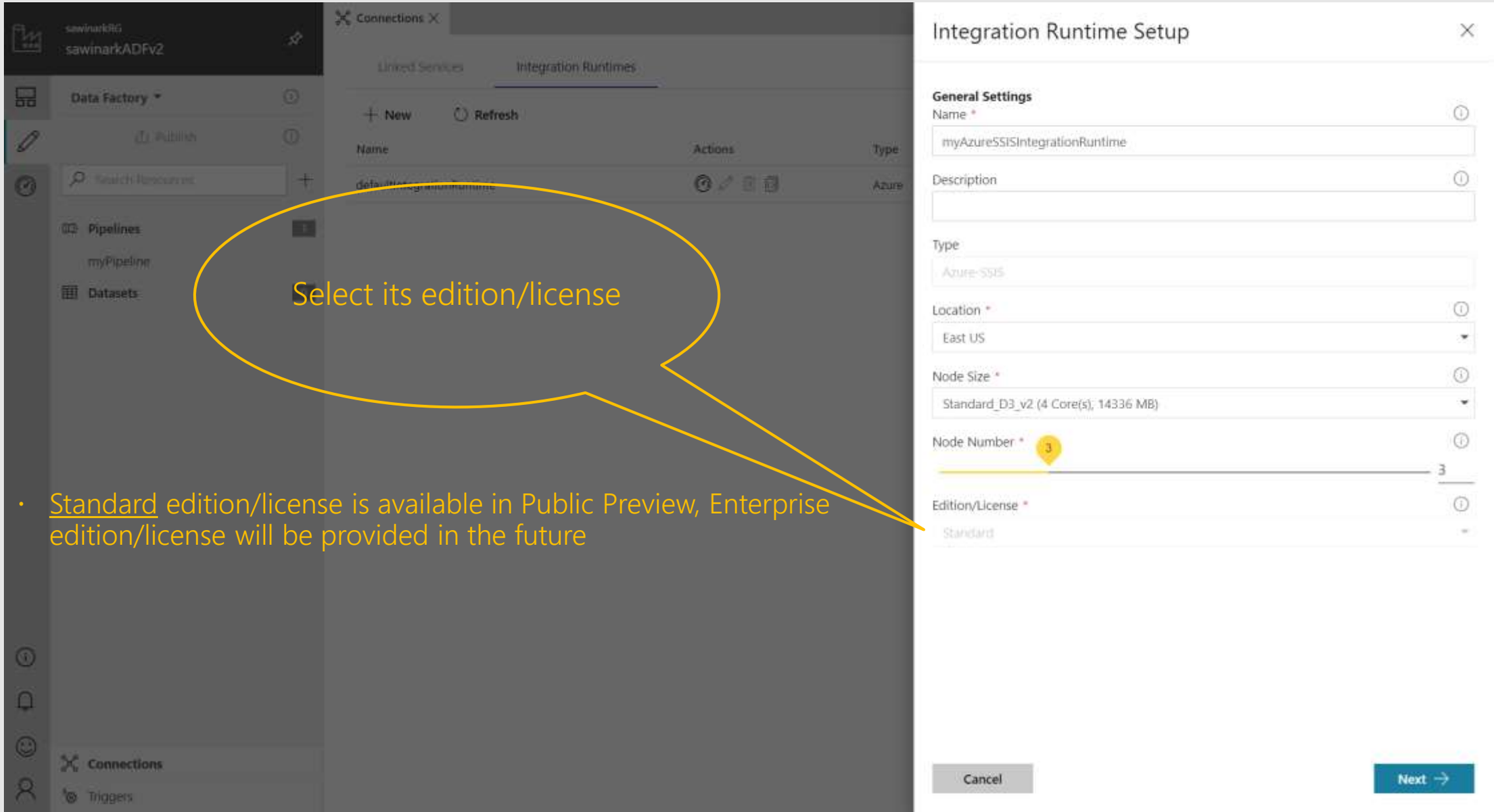
- Select a large node size (scale up), if you want to run many compute/ memory –intensive packages

# Provisioning via ADFv2 App



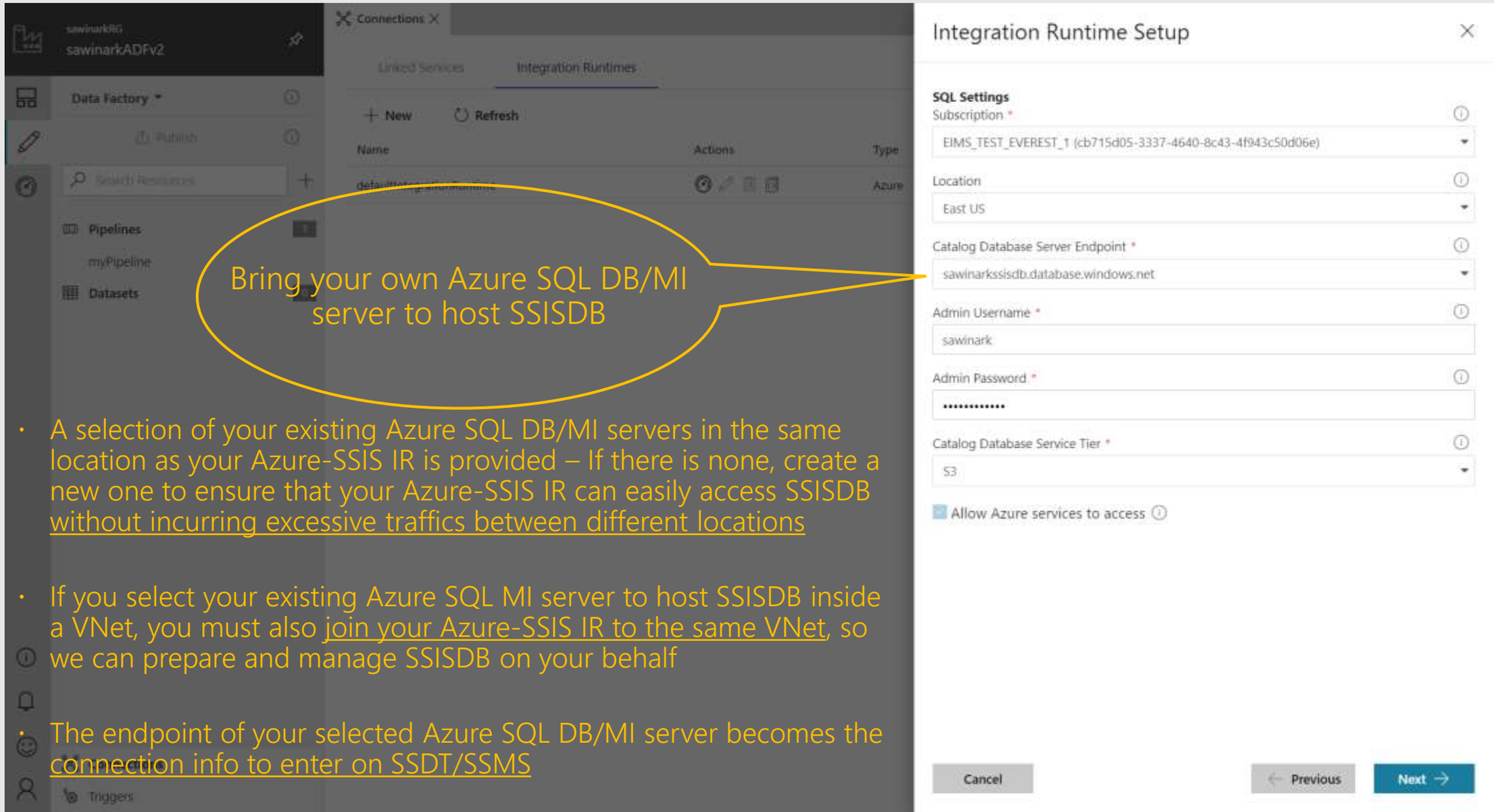

- A range of 1-10 for the number of nodes per cluster is provided
- Select a large cluster with many nodes (scale out), if you want to run many packages in parallel
- Like Azure VMs, you can manage the cost of running Azure-SSIS IR by stopping/starting it, effectively releasing/reallocating its nodes, as you see fit

# Provisioning via ADFv2 App



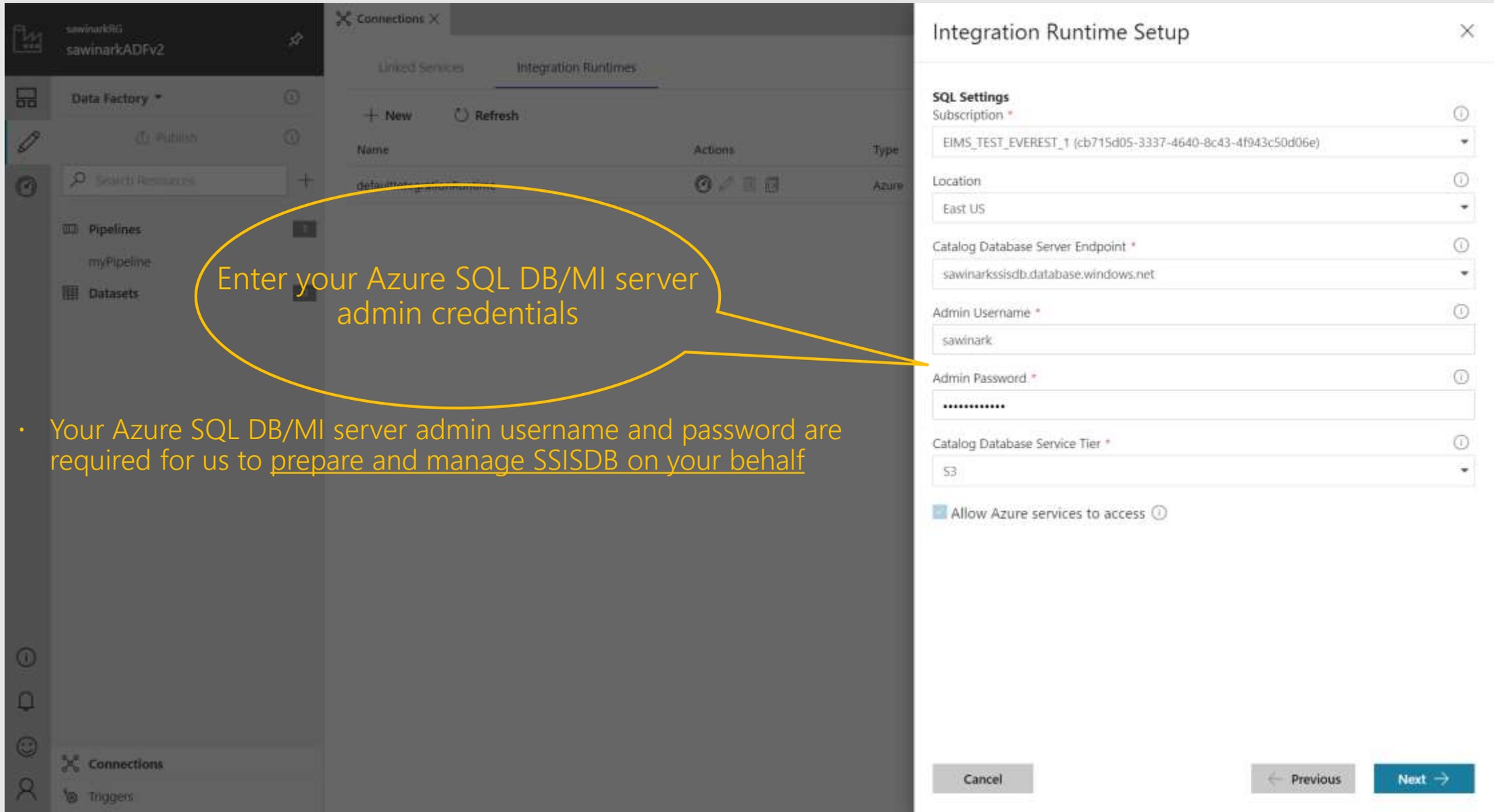

- Standard edition/license is available in Public Preview, Enterprise edition/license will be provided in the future

# Provisioning via ADFv2 App



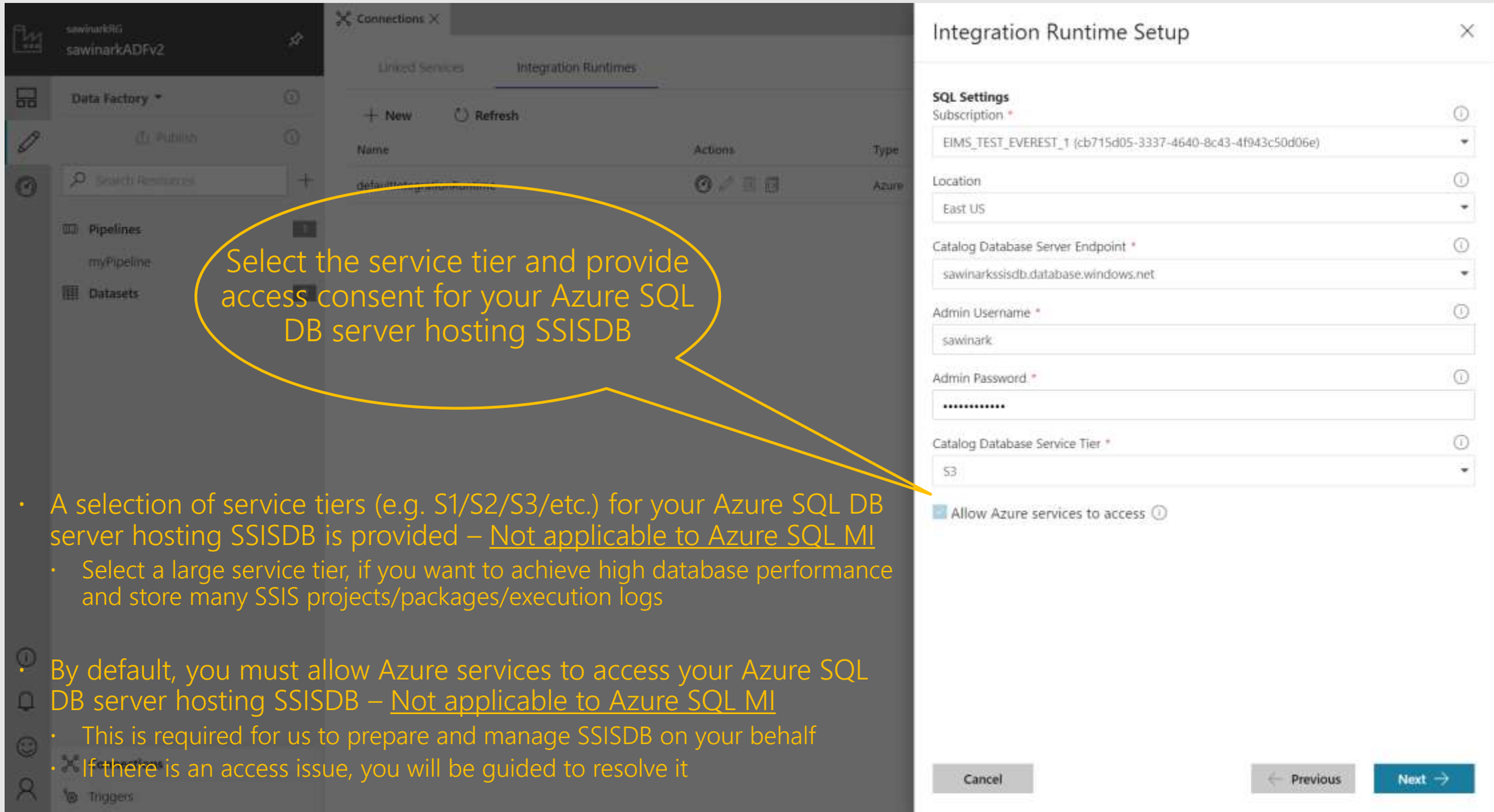

- A selection of your existing Azure SQL DB/MI servers in the same location as your Azure-SSIS IR is provided – If there is none, create a new one to ensure that your Azure-SSIS IR can easily access SSISDB without incurring excessive traffics between different locations
- If you select your existing Azure SQL MI server to host SSISDB inside a VNet, you must also join your Azure-SSIS IR to the same VNet, so we can prepare and manage SSISDB on your behalf
- The endpoint of your selected Azure SQL DB/MI server becomes the connection info to enter on SSDT/SSMS

# Provisioning via ADFv2 App




- Your Azure SQL DB/MI server admin username and password are required for us to prepare and manage SSISDB on your behalf

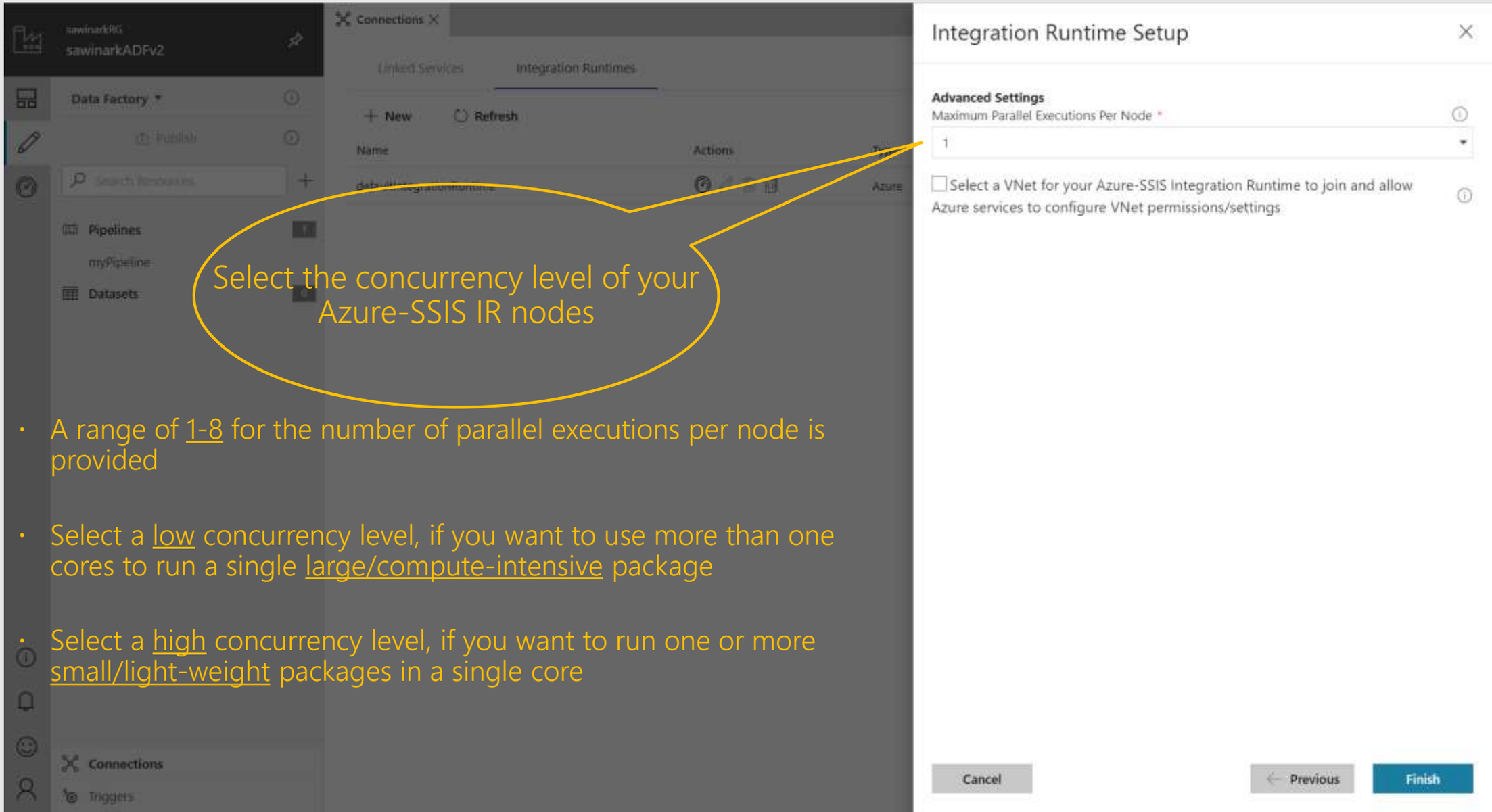
# Provisioning via ADFv2 App




- A selection of service tiers (e.g. S1/S2/S3/etc.) for your Azure SQL DB server hosting SSISDB is provided – Not applicable to Azure SQL MI
  - Select a large service tier, if you want to achieve high database performance and store many SSIS projects/packages/execution logs

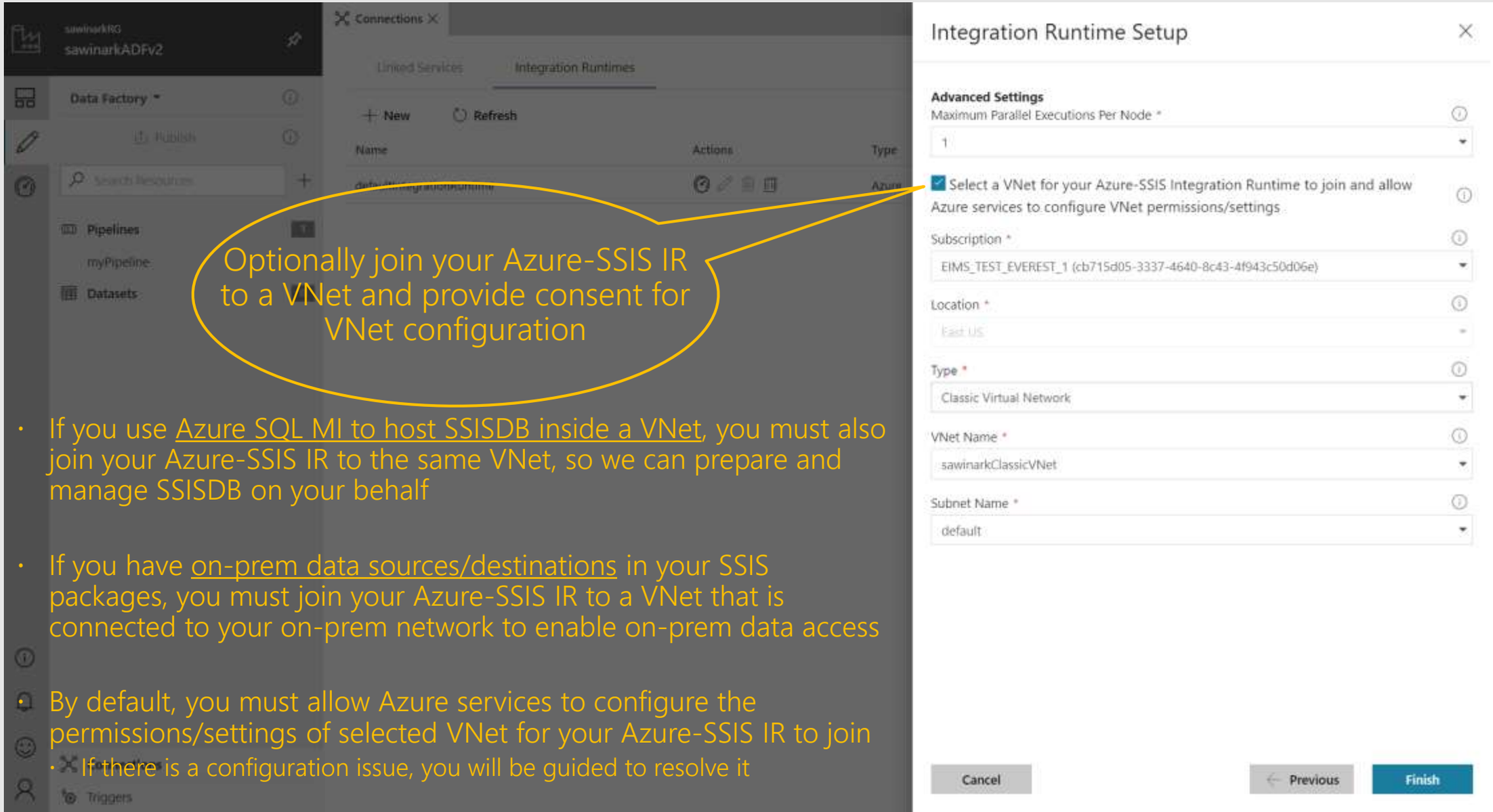
- By default, you must allow Azure services to access your Azure SQL DB server hosting SSISDB – Not applicable to Azure SQL MI
  - This is required for us to prepare and manage SSISDB on your behalf
  - If there is an access issue, you will be guided to resolve it

# Provisioning via ADFv2 App



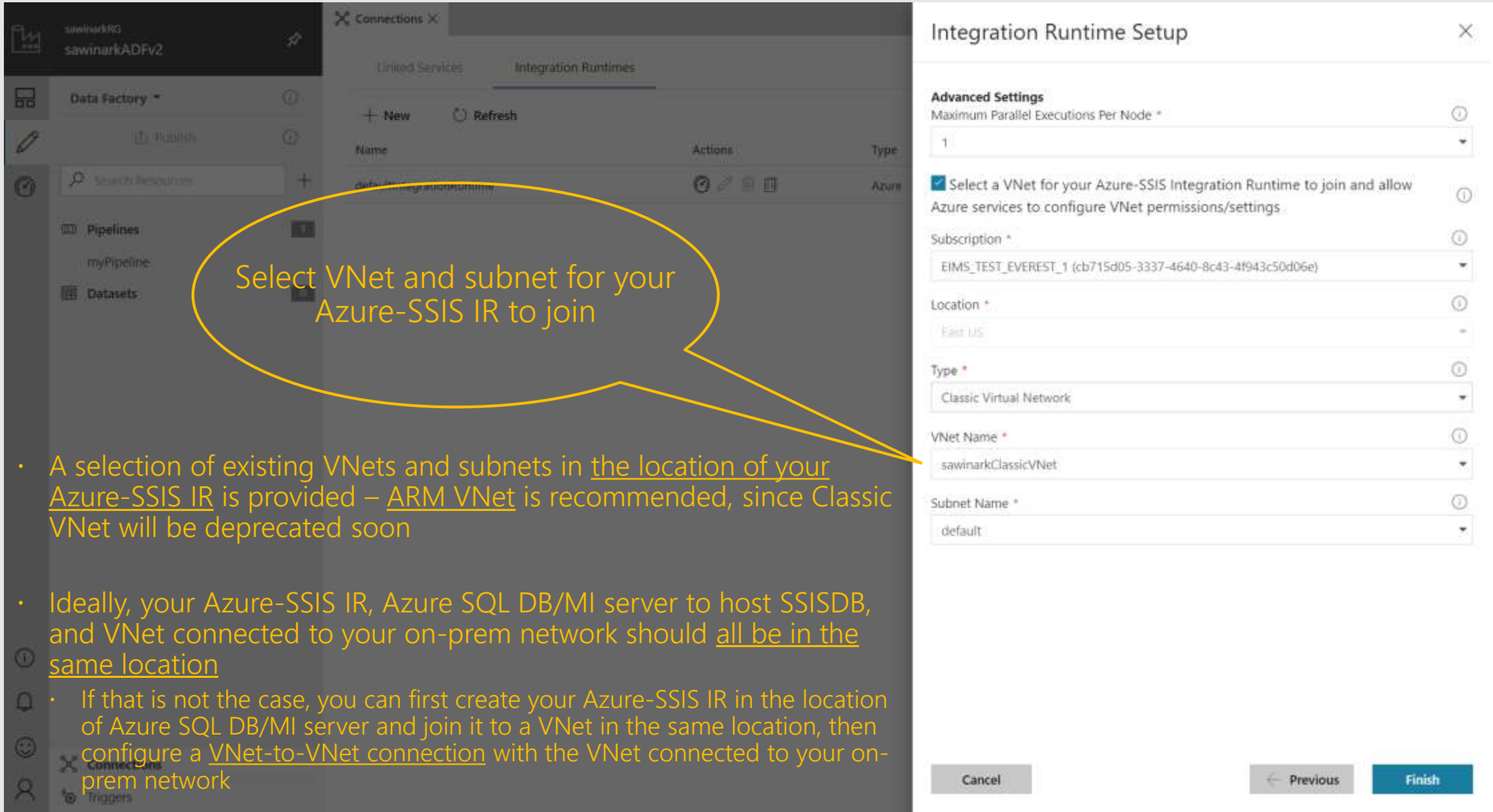

- A range of 1-8 for the number of parallel executions per node is provided
- Select a low concurrency level, if you want to use more than one cores to run a single large/compute-intensive package
- Select a high concurrency level, if you want to run one or more small/light-weight packages in a single core

# Provisioning via ADFv2 App



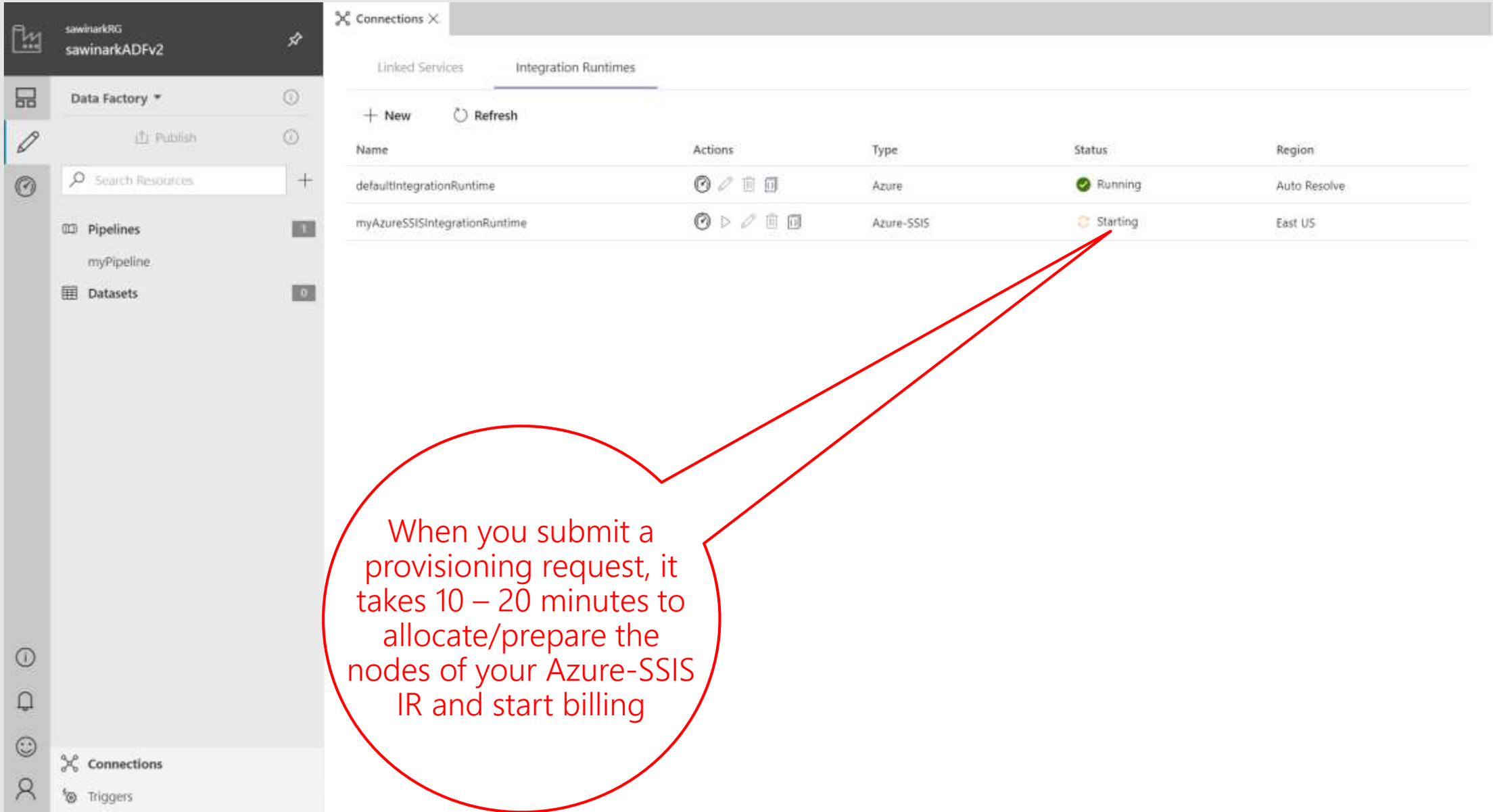

- If you use Azure SQL MI to host SSISDB inside a VNet, you must also join your Azure-SSIS IR to the same VNet, so we can prepare and manage SSISDB on your behalf
- If you have on-prem data sources/destinations in your SSIS packages, you must join your Azure-SSIS IR to a VNet that is connected to your on-prem network to enable on-prem data access
- By default, you must allow Azure services to configure the permissions/settings of selected VNet for your Azure-SSIS IR to join
- If there is a configuration issue, you will be guided to resolve it

# Provisioning via ADFv2 App

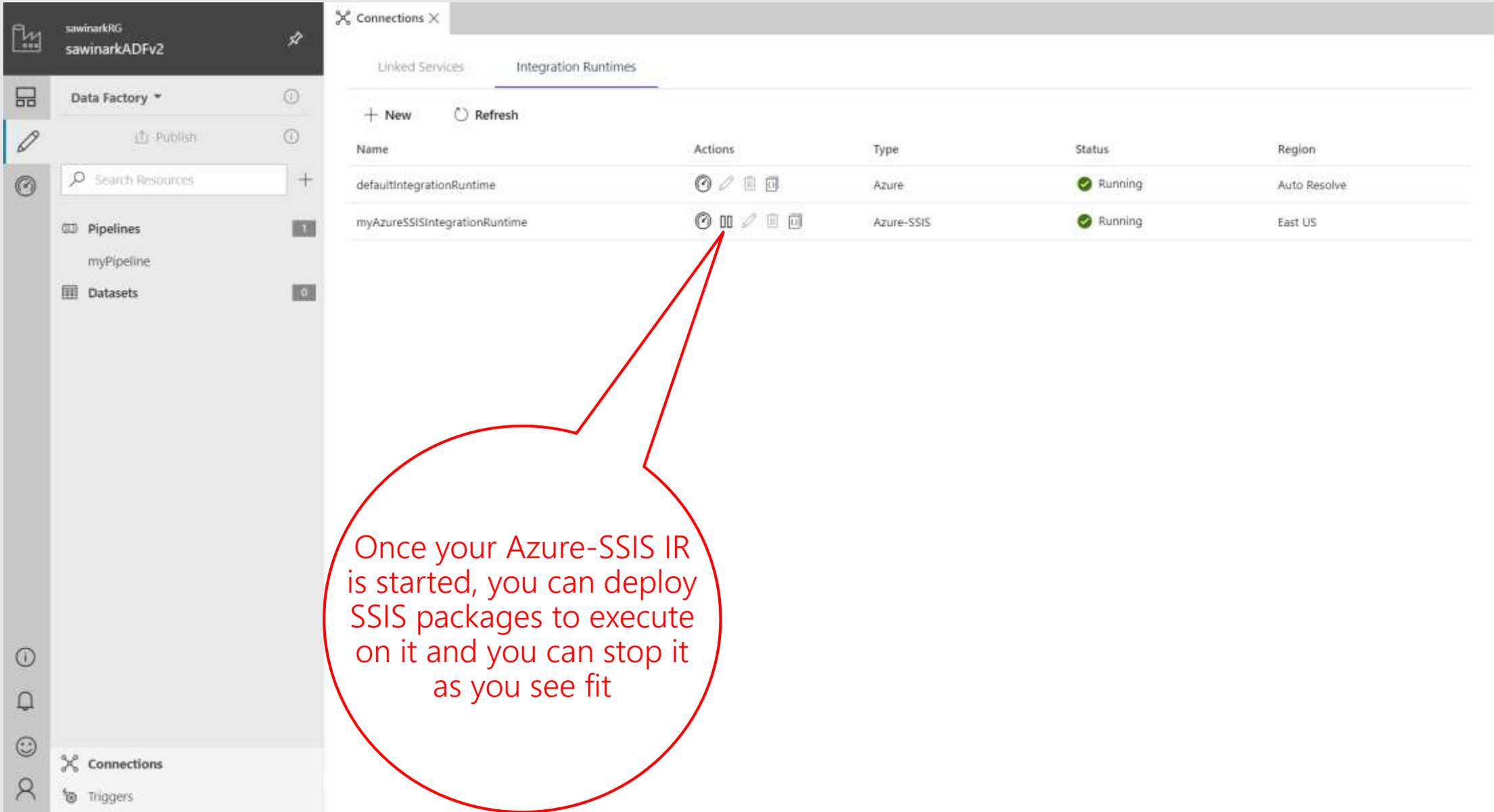



- A selection of existing VNets and subnets in the location of your Azure-SSIS IR is provided – ARM VNet is recommended, since Classic VNet will be deprecated soon
- Ideally, your Azure-SSIS IR, Azure SQL DB/MI server to host SSISDB, and VNet connected to your on-prem network should all be in the same location
  - If that is not the case, you can first create your Azure-SSIS IR in the location of Azure SQL DB/MI server and join it to a VNet in the same location, then configure a VNet-to-VNet connection with the VNet connected to your on-prem network

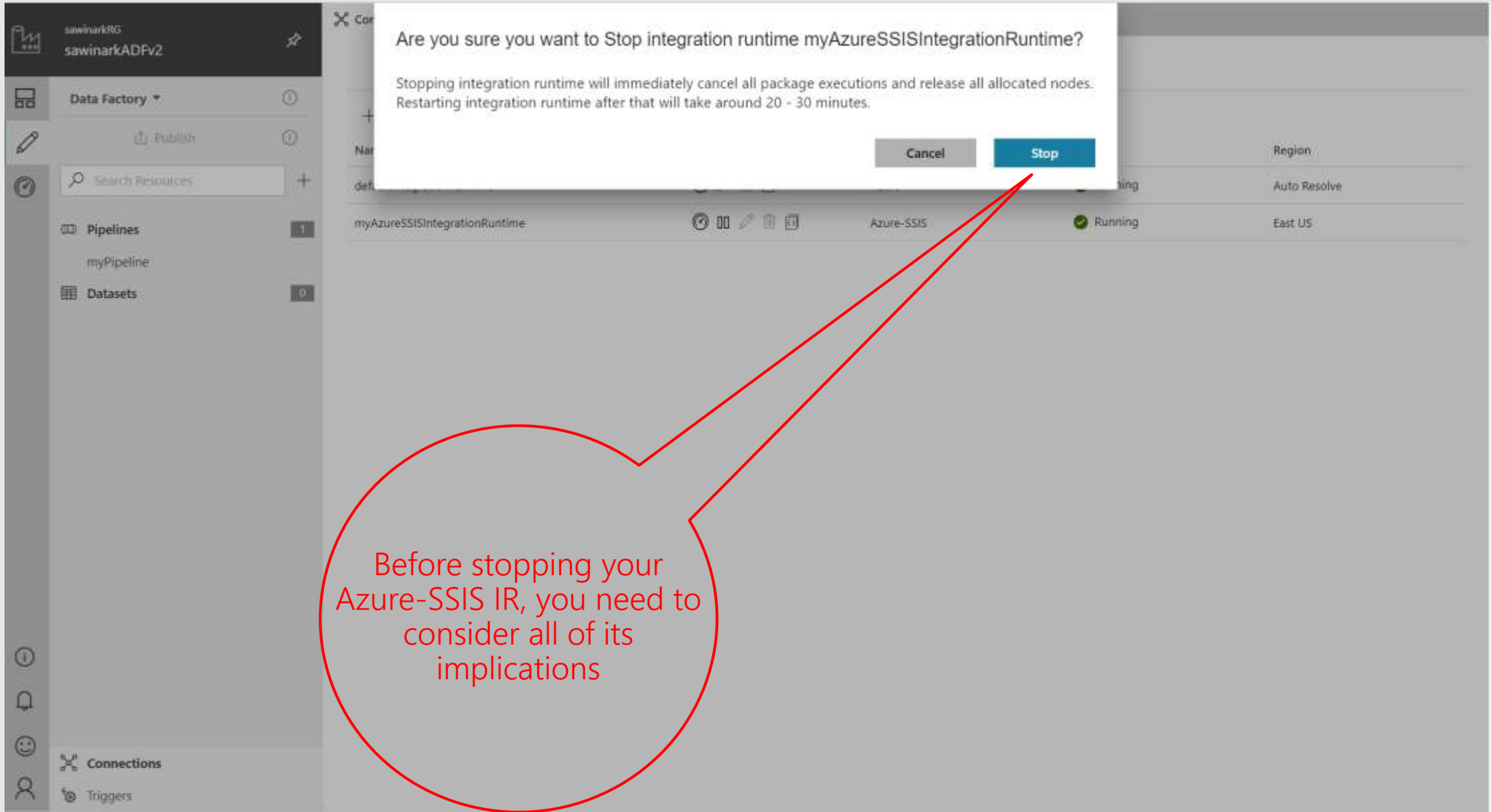
# Provisioning via ADFv2 App



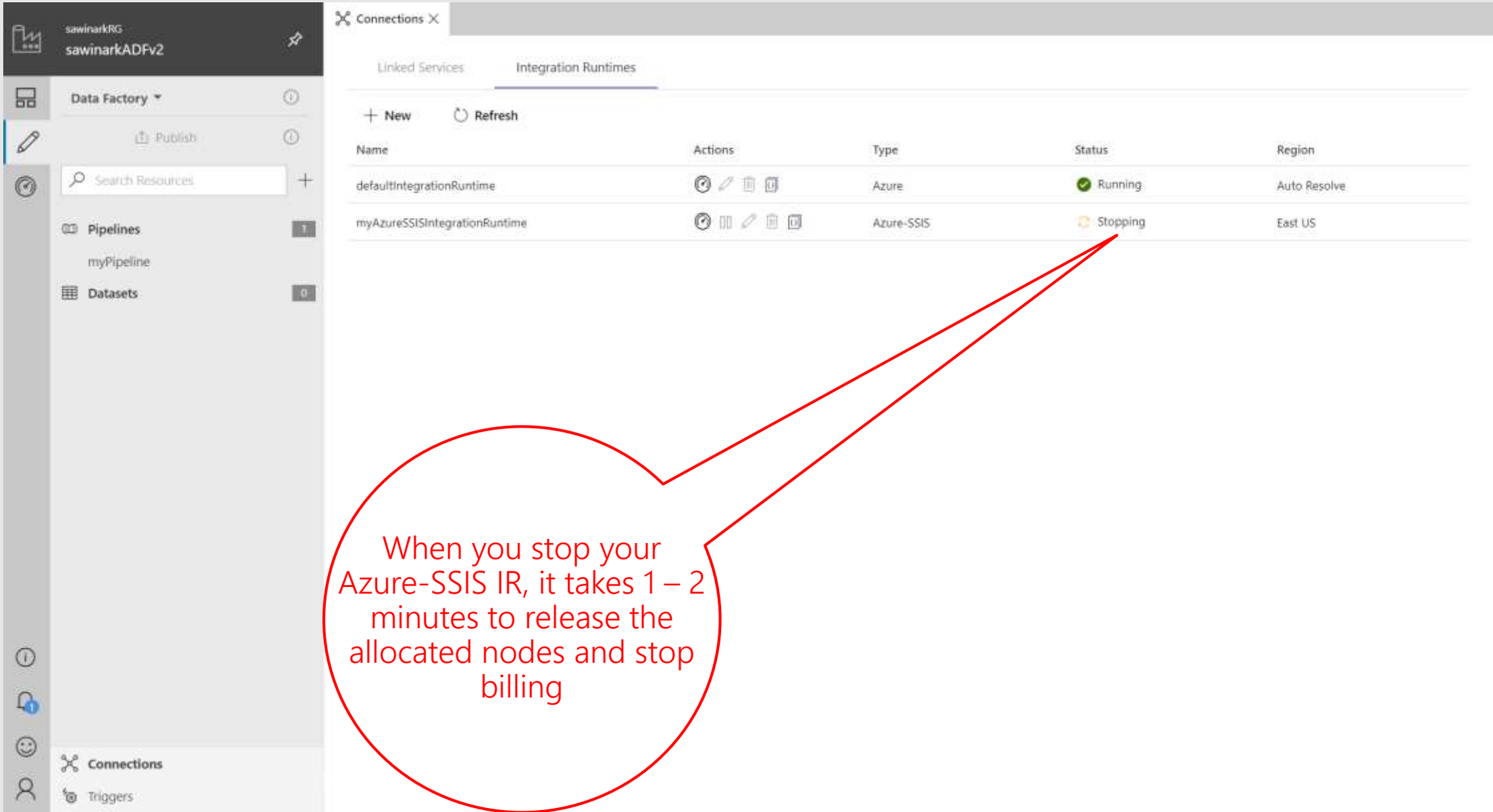
# Provisioning via ADFv2 App



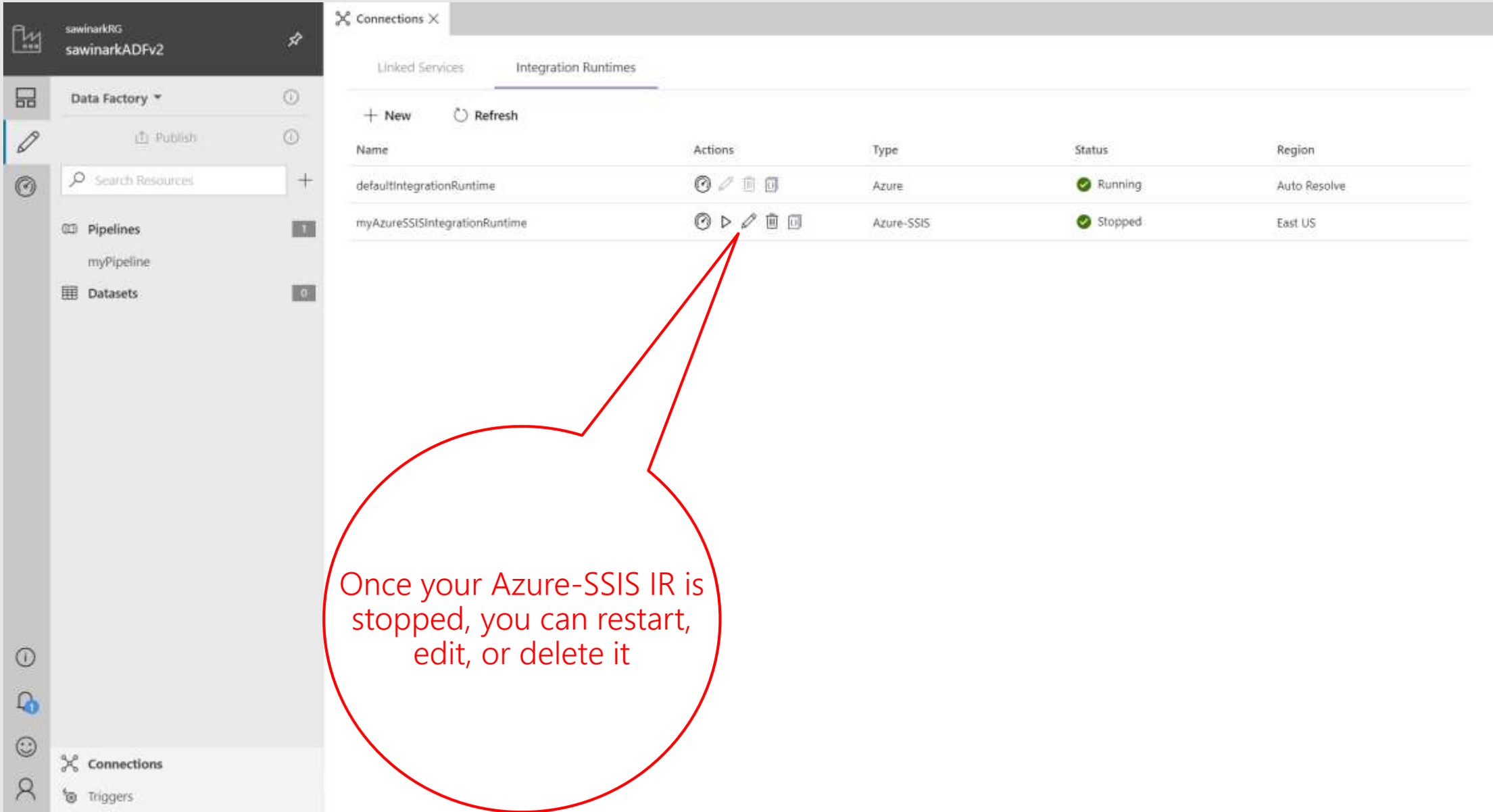
# Reconfiguring via ADFv2 App



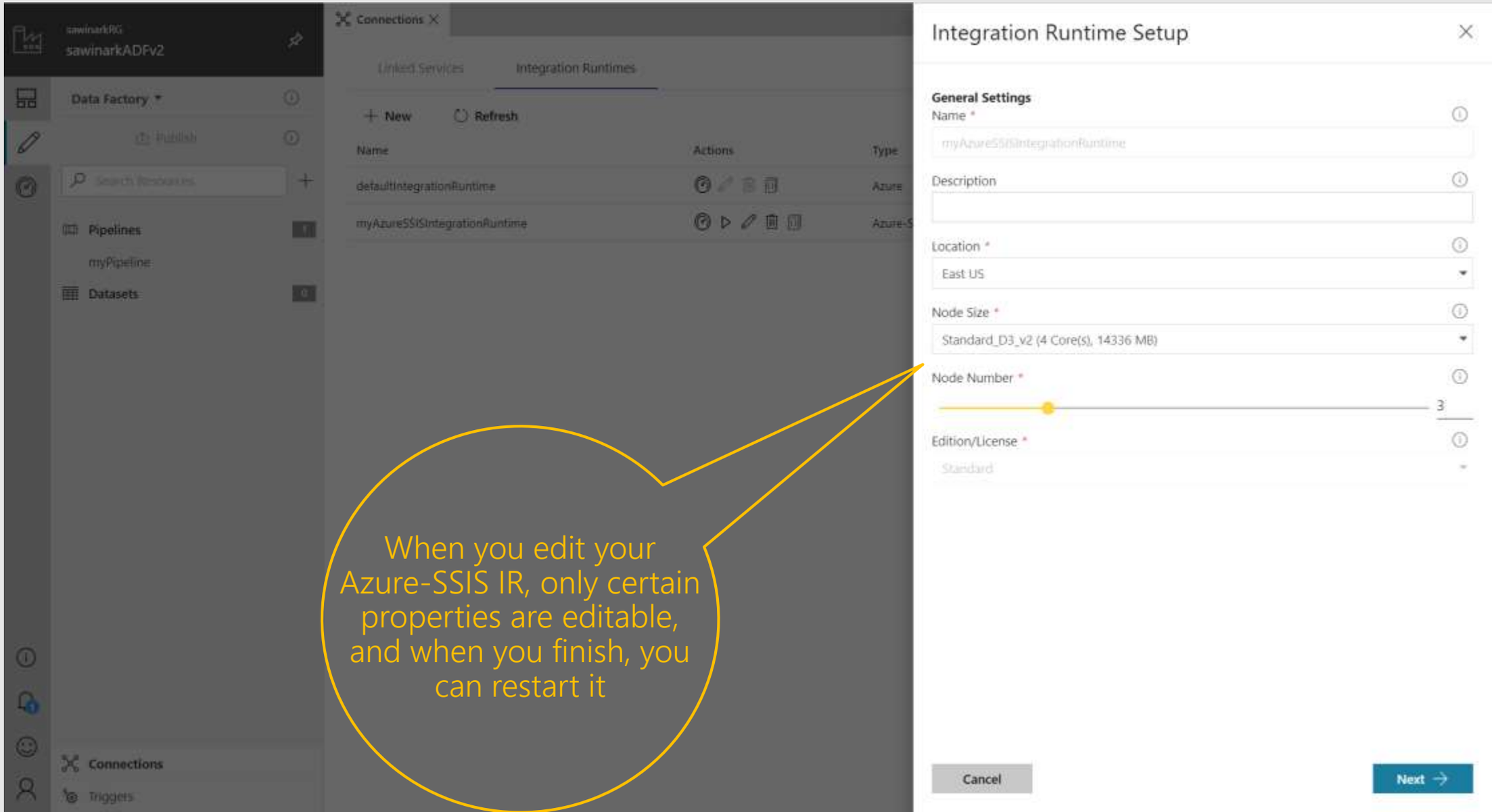
# Reconfiguring via ADFv2 App



# Reconfiguring via ADFv2 App



# Reconfiguring via ADFv2 App



# Provisioning via PSH

```
##### SSIS in ADFv2 specifications (please refer to SSIS in ADFv2 Public Preview documentation for field descriptions) #####
# If your inputs contain PSH special characters, e.g. "$", please precede it with the escape character "`" like "`$".
$SubscriptionName = "[your Azure Subscription name]"
$ResourceGroupName = "[your Azure Resource Group name]"

# ADFv2 info
$DataFactoryName = "[your ADFv2 name]"
$DataFactoryLocation = "EastUS" # In Public Preview, only EastUS|EastUS2|WestEurope are supported for now

# Azure-SSIS Integration Runtime info - This is ADFv2 compute resource for running SSIS packages
$AzureSSISName = "[your Azure-SSIS Integration Runtime name]"
$AzureSSISDescription = "This is my Azure-SSIS Integration Runtime"
$AzureSSISLocation = "EastUS" # In Public Preview, only EastUS|EastUS2|CentralUS|NorthEurope|WestEurope|AustraliaEast are supported for now
$AzureSSISNodeSize = "Standard_A4_v2" # In Public Preview, only Standard_A4_v2|Standard_A8_v2|Standard_D1_v2|Standard_D2_v2|Standard_D3_v2|Standard_D4_v2 are supported for now
$AzureSSISNodeNumber = 2 # In Public Preview, only 1-10 nodes are supported for now
$AzureSSISEdition = “Standard” # Standard Edition is in Public Preview, Enterprise Edition is in Private Preview
$AzureSSISMaxParallelExecutionsPerNode = 4 # In Public Preview, only 1-8 parallel executions per node are supported for now
```

# Provisioning via PSH

```
# Custom setup info
$SetupScriptContainerSasUri = "[SAS URI of your Azure Storage blob container for custom setup or leave it empty]" # Custom setup interface is in Private Preview

# VNet info
$VnetId = "[your VNet resource ID or leave it empty]" # OPTIONAL: ARM VNet is recommended, since Classic VNet will be deprecated soon
#$VnetId = "/subscriptions/your Azure Subscription ID/resourceGroups/your Azure Resource Group name/providers/Microsoft.Network/virtualNetworks/your ARM VNet name" # ARM VNet example
#$VnetId = "/subscriptions/your Azure Subscription ID/resourceGroups/your Azure Resource Group name/providers/Microsoft.ClassicNetwork/virtualNetworks/your Classic VNet name“ # Classic VNet example
$SubnetName = "[your subnet name or leave it empty]" # OPTIONAL: ARM VNet is recommended, since Classic VNet will be deprecated soon

# SSISDB info
$SSISDBServerEndpoint = "[your Azure SQL Database server name.database.windows.net or your Azure SQL Managed Instance server endpoint]"
$SSISDBServerAdminUserName = "[your server admin username]"
$SSISDBServerAdminPassword = "[your server admin password]"
$SSISDBPricingTier = "[your Azure SQL Database pricing tier, e.g. S3, or leave it empty for Azure SQL Managed Instance]" # Not applicable for Azure SQL Managed Instance
##### End of SSIS in ADFv2 specifications #####
```

# Provisioning via PSH

```
##### Validate your Azure SQL Database/Managed Instance server #####
$SSISDBConnectionString = "Data Source=" + $SSISDBServerEndpoint + ";User ID="+ $SSISDBServerAdminUserName
+";Password="+ $SSISDBServerAdminPassword
$sqlConnection = New-Object System.Data.SqlClient.SqlConnection $SSISDBConnectionString;
Try
{
    $sqlConnection.Open();
}
Catch [System.Data.SqlClient.SqlException]
{
    Write-Warning "Cannot connect to your Azure SQL DB logical server/Azure SQL MI server, exception: $_"  ;
    Write-Warning "Please make sure the server you specified has already been created. Do you want to proceed? [Y/N]"
    $yn = Read-Host
    if(!($yn -ieq "Y"))
    {
        Return;
    }
}

##### Login and select your Azure subscription #####
Login-AzureRmAccount
Select-AzureRmSubscription -SubscriptionName $SubscriptionName
```

# Provisioning via PSH

```
##### Automatically configure VNet permissions/settings for your Azure SSIS Integration Runtime to join #####
# Register to Azure Batch resource provider
if(![string]::IsNullOrEmpty($VnetId) -and ![string]::IsNullOrEmpty($SubnetName))
{
    Register-AzureRmResourceProvider -ProviderNamespace Microsoft.Batch
    while(!(Get-AzureRmResourceProvider -ProviderNamespace "Microsoft.Batch").RegistrationState.Contains("Registered"))
    {
        Start-Sleep -s 10
    }
    if($VnetId -match "/providers/Microsoft.ClassicNetwork/")
    {
        # Assign VM contributor role to Azure Batch
        $BatchObjectId = (Get-AzureRmADServicePrincipal -ServicePrincipalName "MicrosoftAzureBatch").Id
        New-AzureRmRoleAssignment -ObjectId $BatchObjectId -RoleDefinitionName "Classic Virtual Machine Contributor" -Scope $VnetId
    }
}
```

# Provisioning via PSH

```
##### Provision your ADFv2 + Azure SSIS Integration Runtime #####
New-AzureRmResourceGroup -Location $DataFactoryLocation -Name $ResourceGroupName
Register-AzureRmResourceProvider -ProviderNamespace Microsoft.DataFactory

$secpasswd = ConvertTo-SecureString $SSISDBServerAdminPassword -AsPlainText -Force
$serverCreds = New-Object
System.Management.Automation.PSCredential($SSISDBServerAdminUserName, $secpasswd)

Set-AzureRmDataFactoryV2 -ResourceGroupName $ResourceGroupName `
                         -Location $DataFactoryLocation `
                         -Name $DataFactoryName
```

# Provisioning via PSH

```
Set-AzureRmDataFactoryV2IntegrationRuntime -ResourceGroupName $ResourceGroupName `
                                           -DataFactoryName $DataFactoryName `
                                           -Type Managed `
                                           -Name $AzureSSISName `
                                           -Description $AzureSSISDescription `
                                           -Location $AzureSSISLocation `
                                           -NodeSize $AzureSSISNodeSize `
                                           -NodeCount $AzureSSISNodeNumber `
                                           -Edition $AzureSSISEdition `
                                           -MaxParallelExecutionsPerNode $AzureSSISMaxParallelExecutionsPerNode `
                                           -SetupScriptContainerSasUri $SetupScriptContainerSasUri `
                                           -VnetId $VnetId `
                                           -Subnet $SubnetName `
                                           -CatalogServerEndpoint $SSISDBServerEndpoint `
                                           -CatalogAdminCredential $serverCreds `
                                           -CatalogPricingTier $SSISDBPricingTier
write-host("##### Starting #####")
Start-AzureRmDataFactoryV2IntegrationRuntime -ResourceGroupName $ResourceGroupName `
                                             -DataFactoryName $DataFactoryName `
                                             -Name $AzureSSISName `
                                             -Force
write-host("##### Completed #####")
write-host("If any cmdlet is unsuccessful, please consider using -Debug option for diagnostics.")
```

# Reconfiguring via PSH

```
##### Reconfigure your Azure SSIS Integration Runtime, e.g. stopping/scaling out to 5 nodes/restarting #####
# Stopping your Azure-SSIS Integration Runtime will release its nodes and stop billing
#Stop-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $DataFactoryName -Name $AzureSSISName -ResourceGroupName $ResourceGroupName

# Scaling out your Azure-SSIS Integration Runtime to 5 nodes
#Set-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $DataFactoryName -Name $AzureSSISName -ResourceGroupName $ResourceGroupName -NodeCount 5

# Starting your Azure-SSIS Integration Runtime will allocate its nodes and start billing
#Start-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $DataFactoryName -Name $AzureSSISName -ResourceGroupName $ResourceGroupName

##### Clean up ######
#Stop-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $DataFactoryName -Name $AzureSSISName -ResourceGroupName $ResourceGroupName -Force
#Remove-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $DataFactoryName -Name $AzureSSISName -ResourceGroupName $ResourceGroupName -Force
#Remove-AzureRmDataFactoryV2 -Name $DataFactoryName -ResourceGroupName $ResourceGroupName -Force
#Remove-AzureRmResourceGroup -Name $ResourceGroupName -Force
```

# Deployment Methods

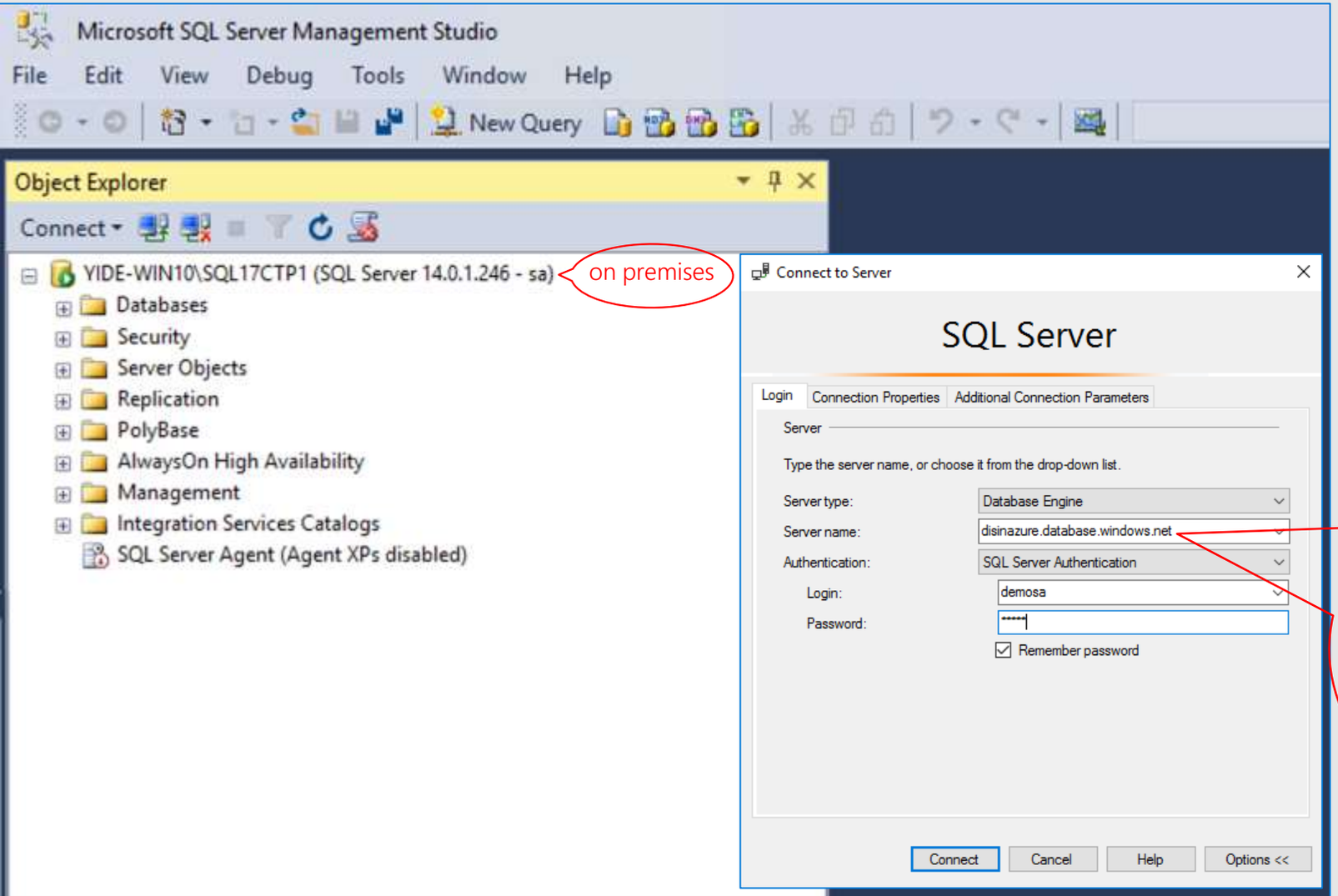
# Deployment Methods

- SSIS PaaS supports the project deployment model used in SSIS 2012/later versions
  - Projects built in the legacy package deployment model used in SSIS 2008/earlier versions can be converted into this model via SSDT/SSMS using Integration Services Project Conversion Wizard
  - Packages built in SSIS 2008/earlier versions can be upgraded to the latest version supported by SSIS PaaS via SSDT/SSMS using SSIS Package Upgrade Wizard
  - In this model, the whole project needs to be deployed after any package changes – An incremental package deployment feature will be provided in the near future
  - Projects containing environment references/run-time parameters can be saved into project deployment files (.ispac extension)
  - Projects are deployed into SSISDB hosted by Azure SQL DB/MI server, packages are run by creating/starting jobs via SSISDB sprocs that will be executed on Azure-SSIS IR, and execution logs are written back into SSISDB

# Deployment Methods

- SSIS projects can be deployed via SSDT/SSMS using Integration Services Deployment Wizard
- SSIS projects can be deployed via Command Line Interface (CLI)
  - Run isdeploymentwizard.exe from the command prompt (TBD)
- SSIS projects can be deployed via PSH/custom code using SSIS Managed Object Model (MOM) .NET SDK/API
  - Microsoft.SqlServer.Management.IntegrationServices.dll is installed in .NET Global Assembly Cache (GAC) with SQL Server/SSMS installation
- SSIS projects can be deployed via T-SQL scripts executing SSISDB sprocs
  - Execute SSISDB sproc [catalog].[deploy_project]

# Deployment via SSMS

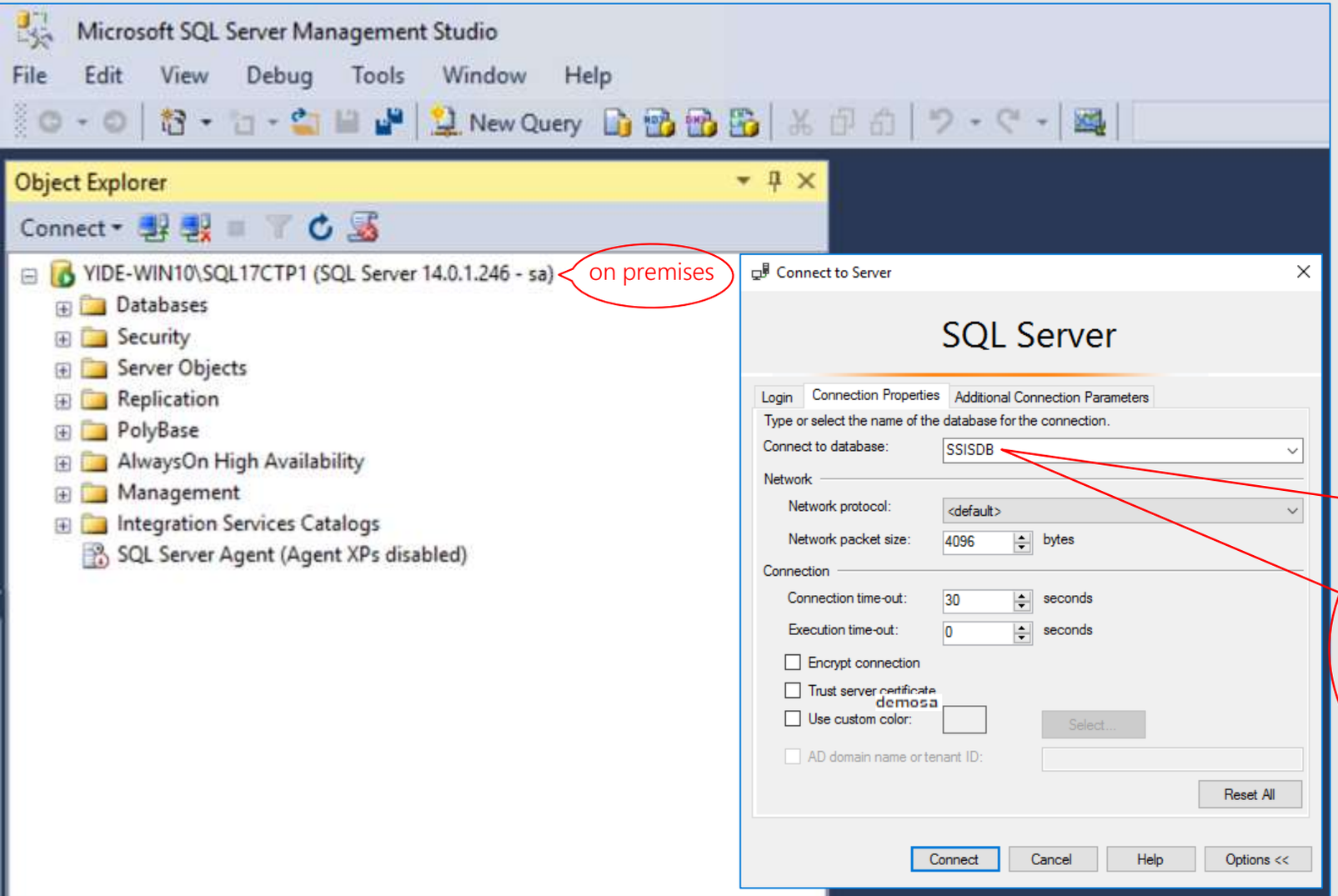






On SSMS, you can connect to SSIS PaaS using its connection info and SQL/AAD authentication credentials

# Deployment via SSMS

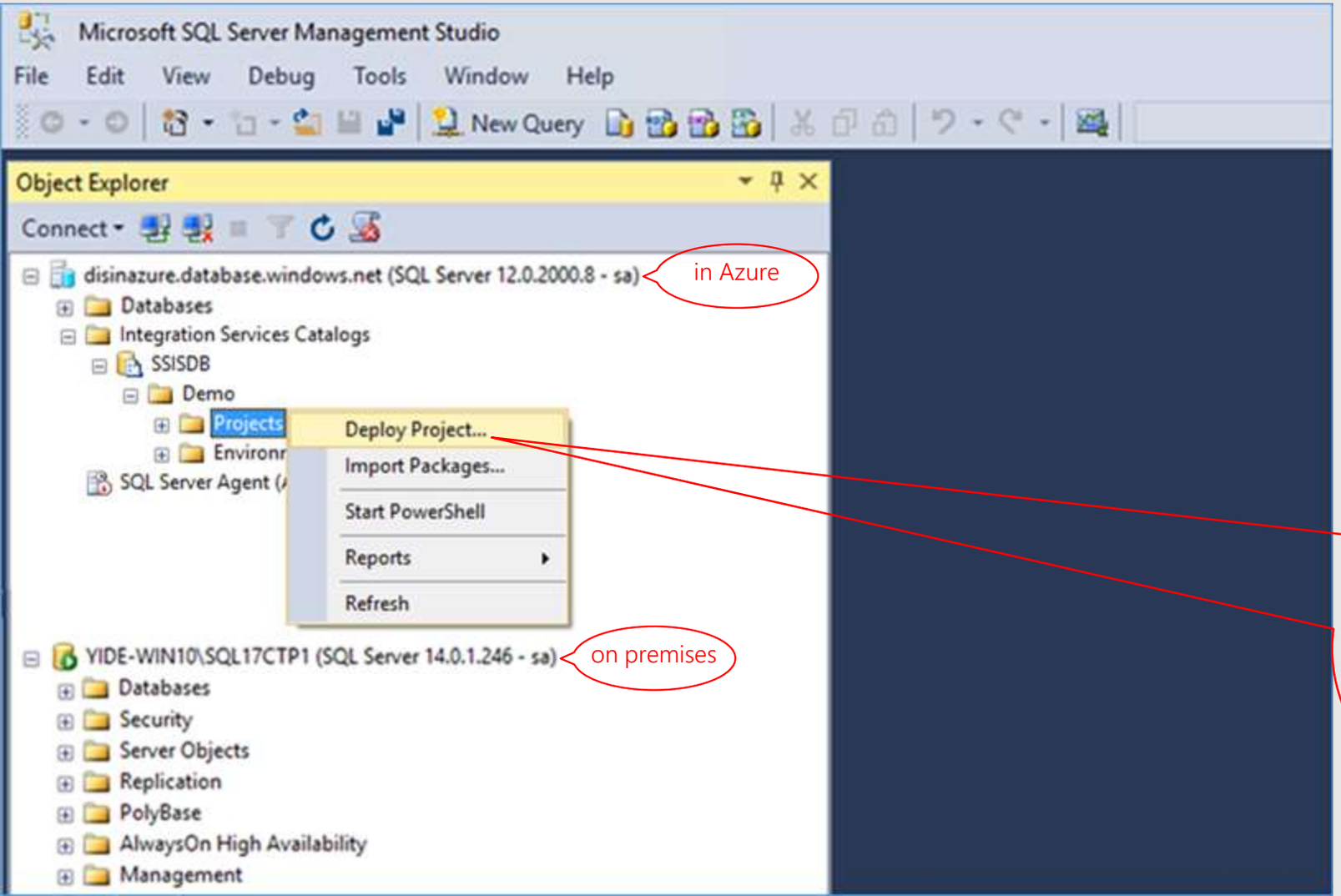






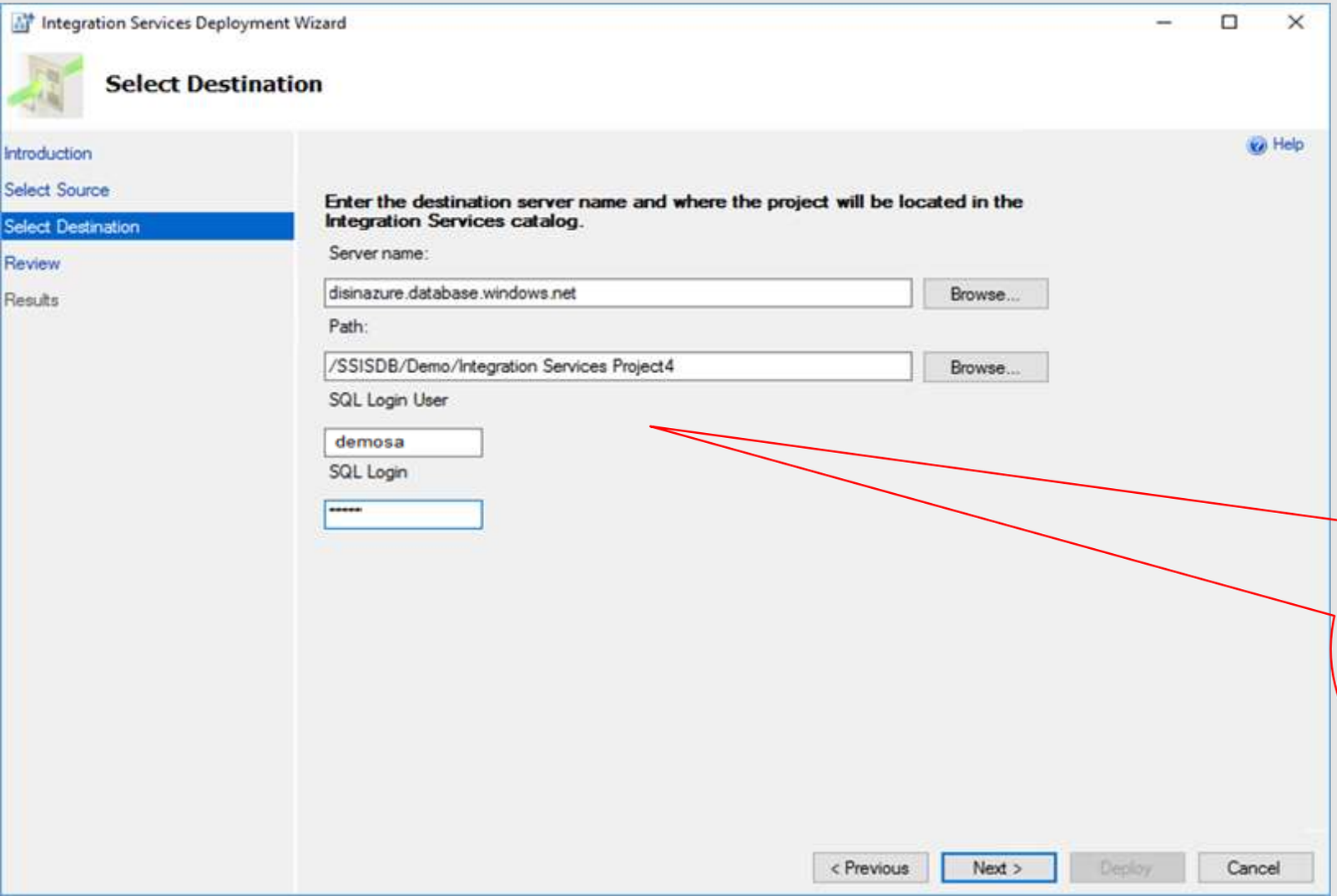
On “Connection Properties” tab, enter “SSISDB” as the target database to connect

# Deployment via SSMS




Once connected, you can deploy projects/packages to SSIS PaaS from your local file system/SSIS on premises

# Deployment via SSMS




On Integration Services Deployment Wizard, enter SSIS PaaS connection info and SQL authentication credentials

# Execution Methods

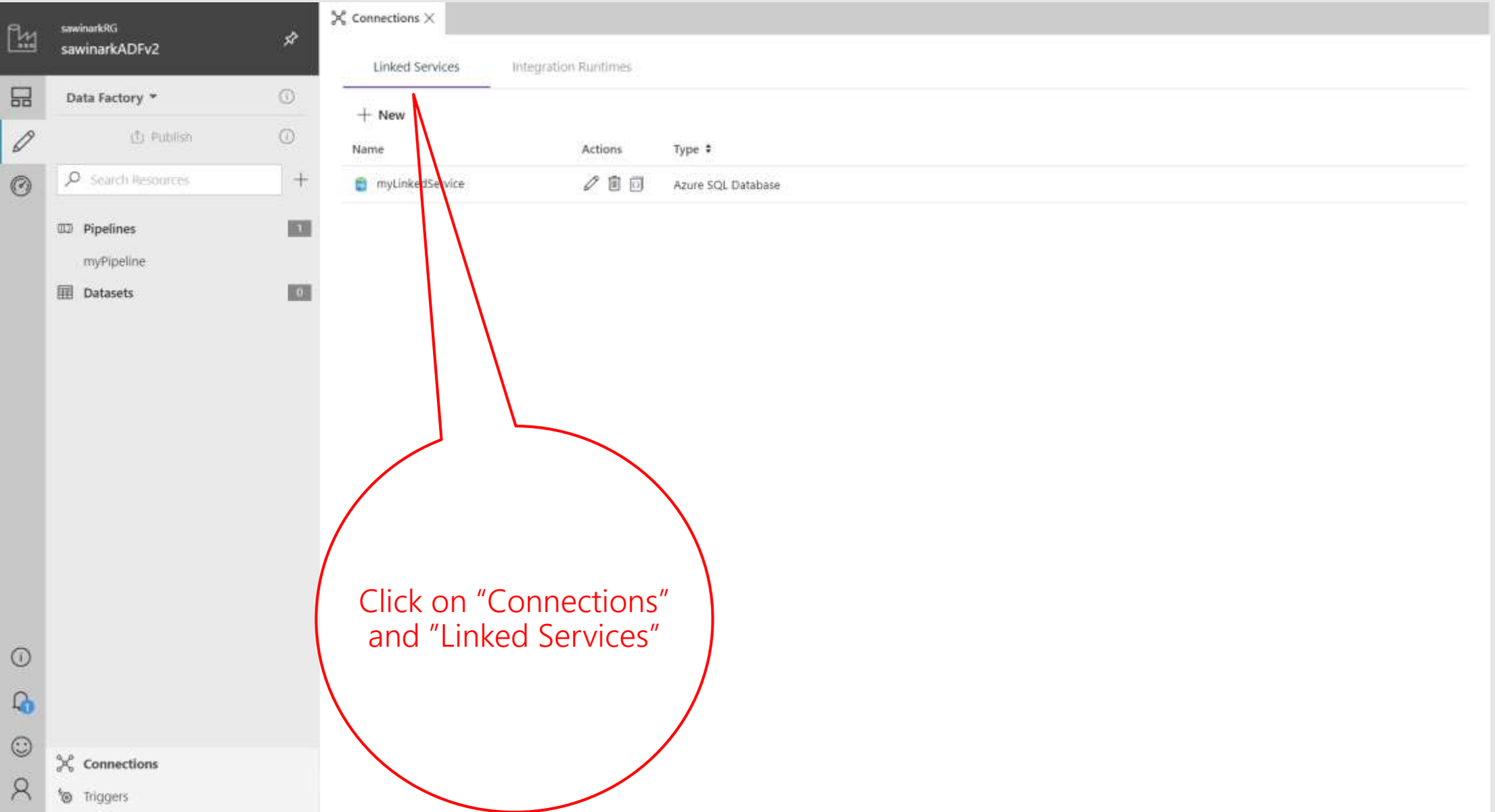
# Execution Methods

- SSIS packages can be directly executed as first-class SSIS activities in ADFv2 pipelines (Work in Progress)
  - For now, SSIS packages can be indirectly executed via ADFv2 Sproc Activity
- SSIS packages can be executed via SSMS
- SSIS packages can be executed via CLI
  - Run dtexec.exe from the command prompt (TBD)
- SSIS packages can be executed via PSH/custom code using SSIS MOM .NET SDK/API
  - Microsoft.SqlServer.Management.IntegrationServices.dll is installed in .NET GAC with SQL Server/SSMS installation
- SSIS packages can be executed via T-SQL scripts executing SSISDB sprocs
  - Execute SSISDB sprocs [catalog].[create_execution] + [catalog].[set_execution_parameter_value] + [catalog].[start_execution]

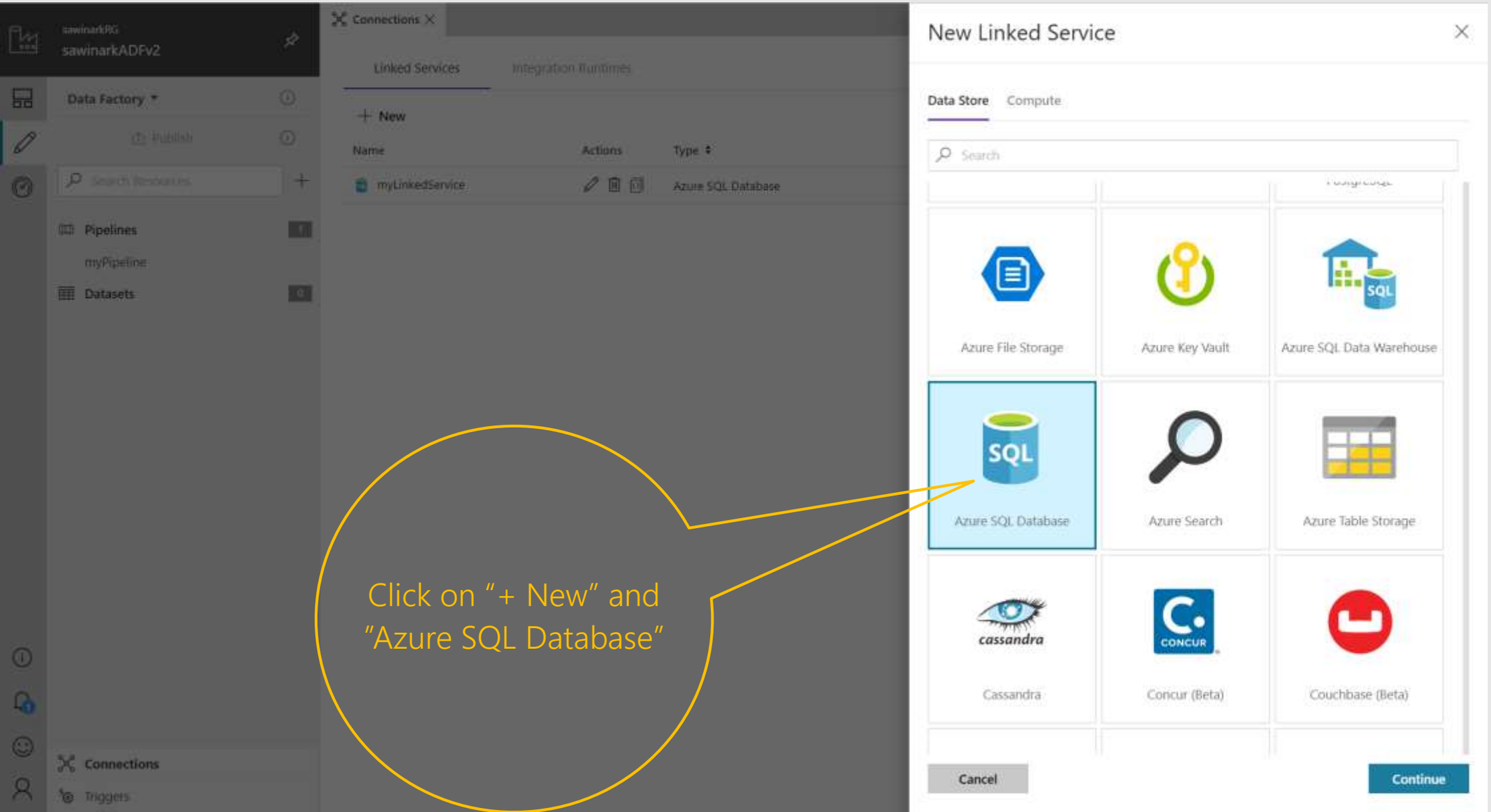
# Execution via ADFv2 Sproc Activity

- Create a linked service for Azure SQL DB/MI server hosting SSISDB
- Create a pipeline with SqlServerStoredProcedure activity
- Trigger/execute the pipeline on demand

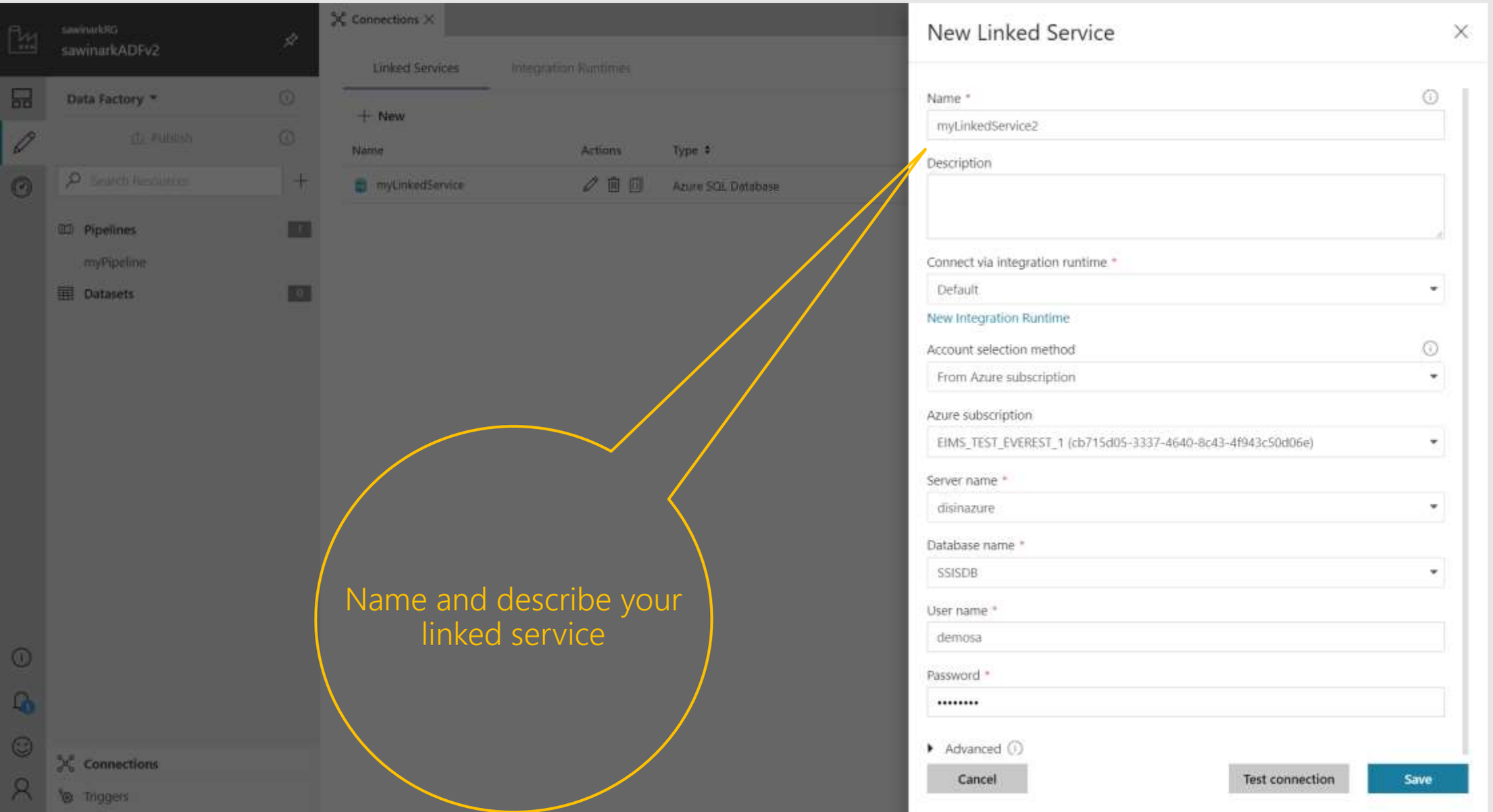
# Execution via ADFv2 Sproc Activity



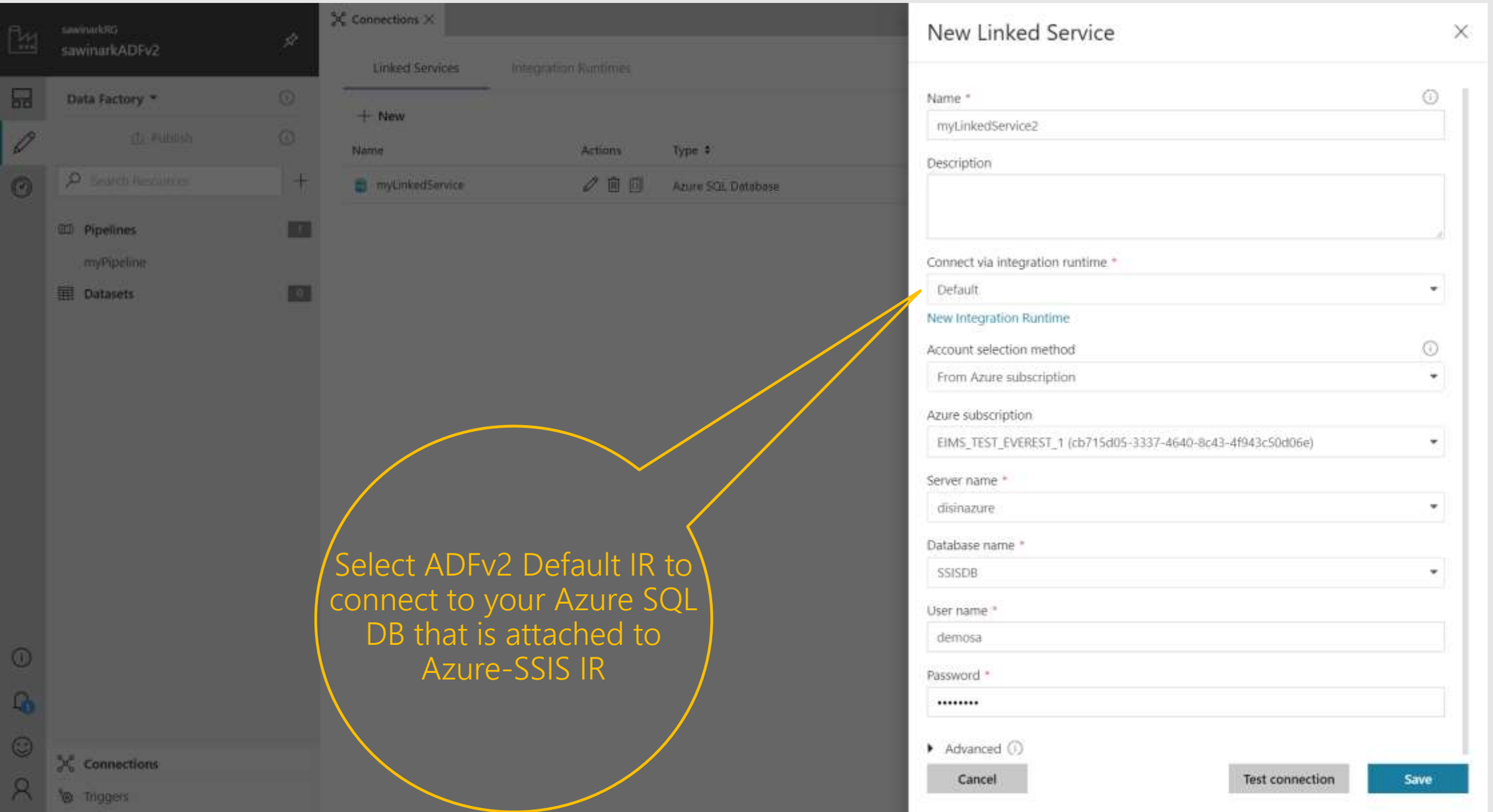
# Execution via ADFv2 Sproc Activity



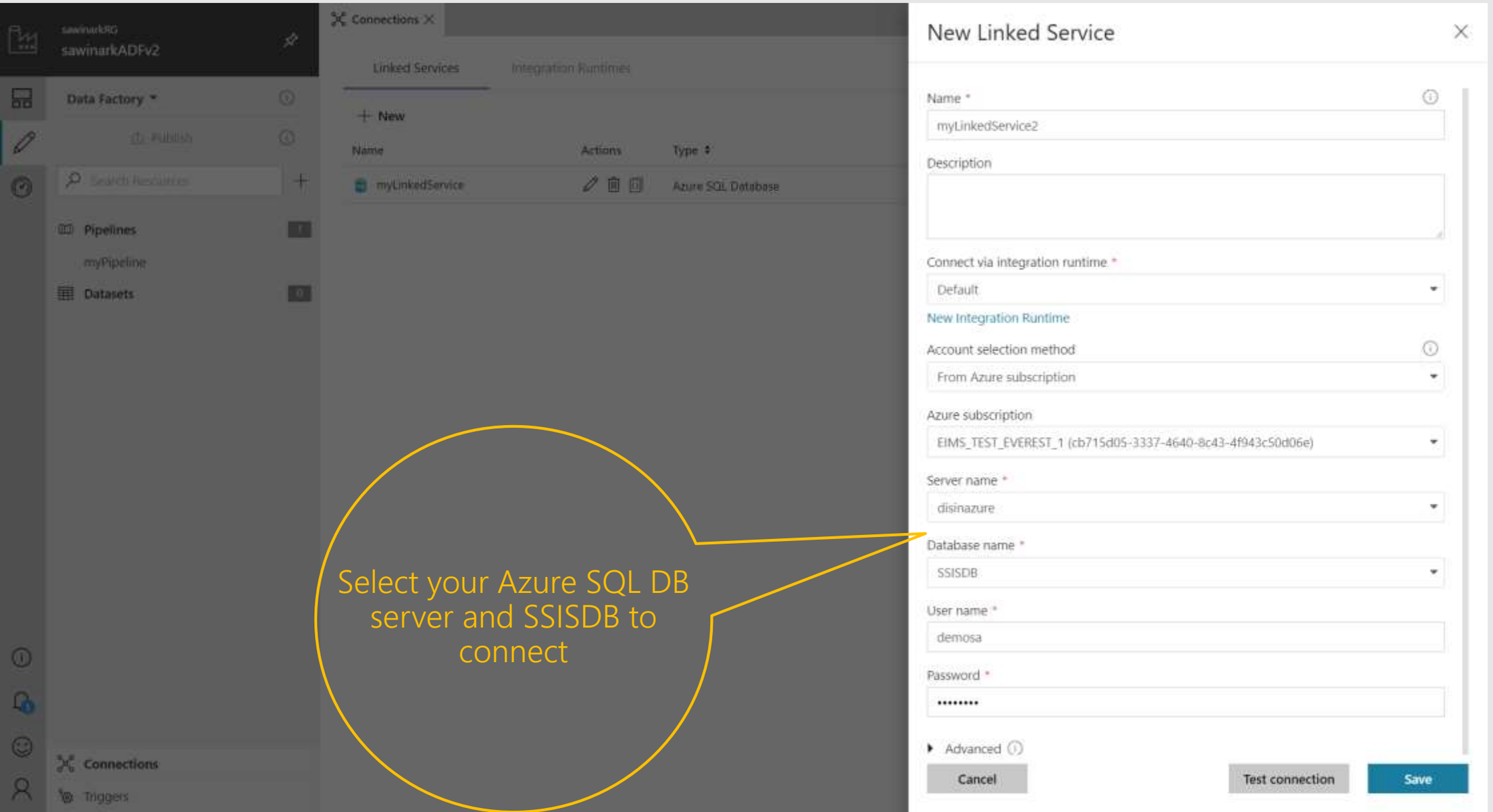
# Execution via ADFv2 Sproc Activity



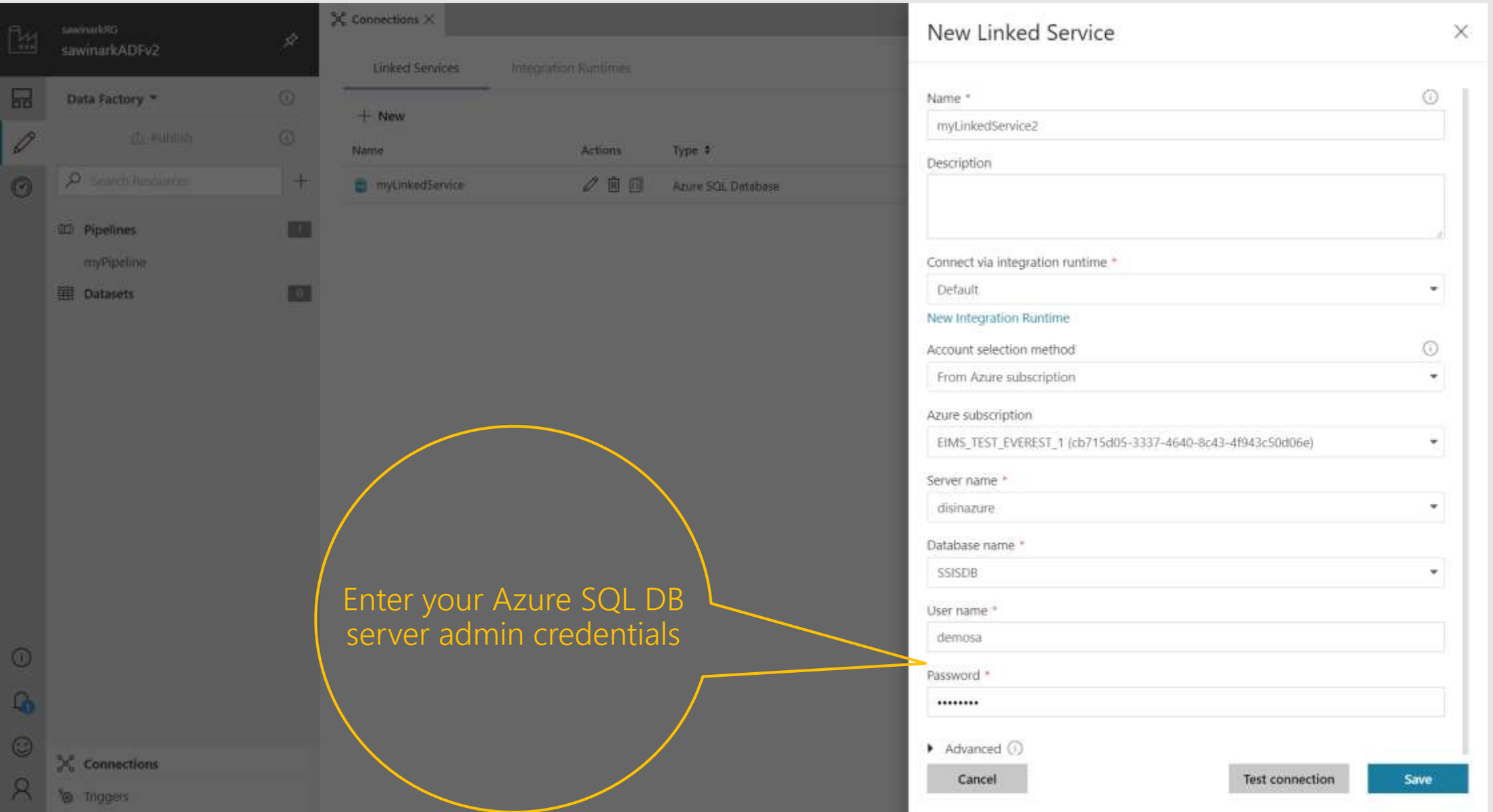
# Execution via ADFv2 Sproc Activity



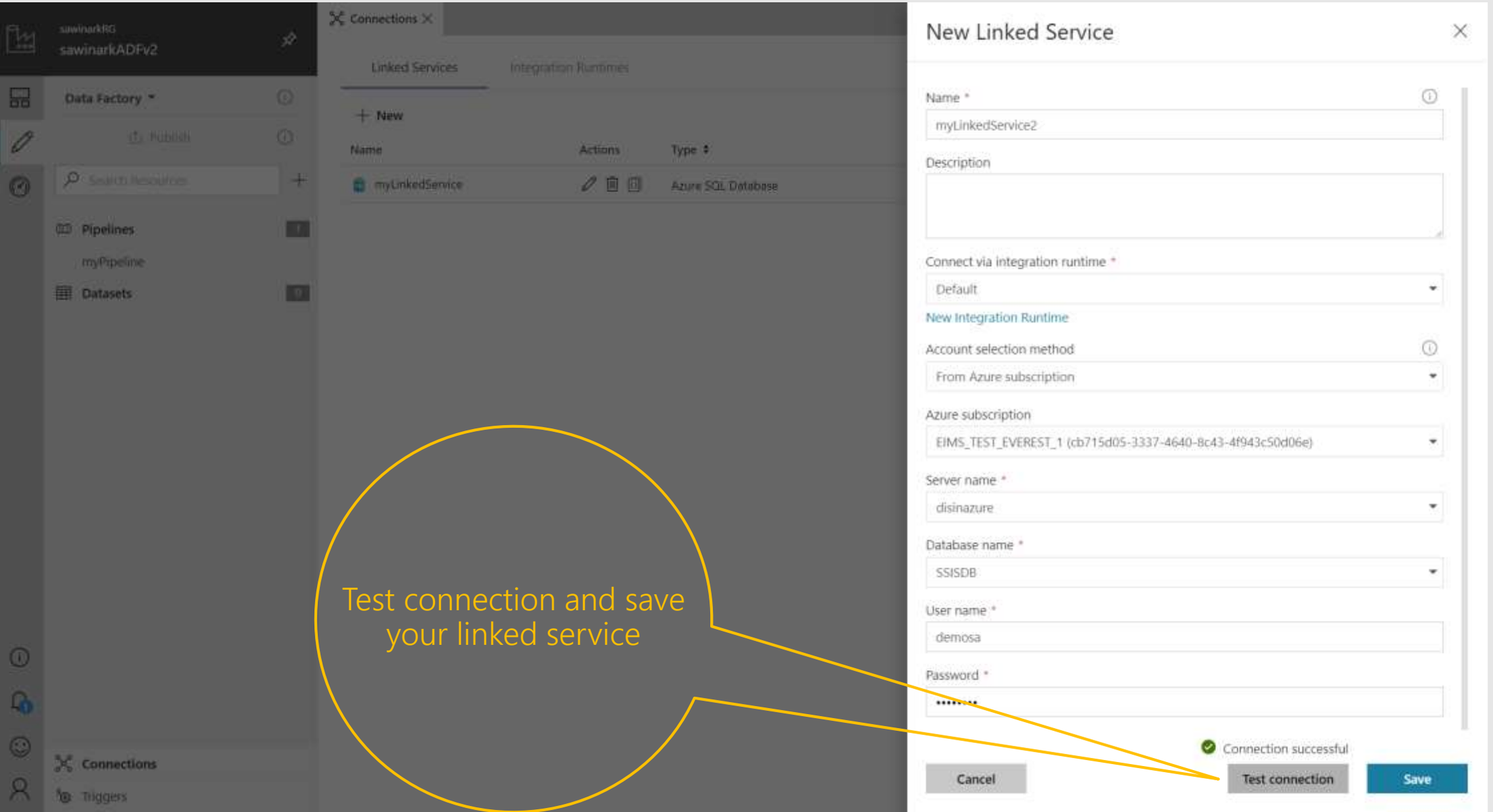
# Execution via ADFv2 Sproc Activity



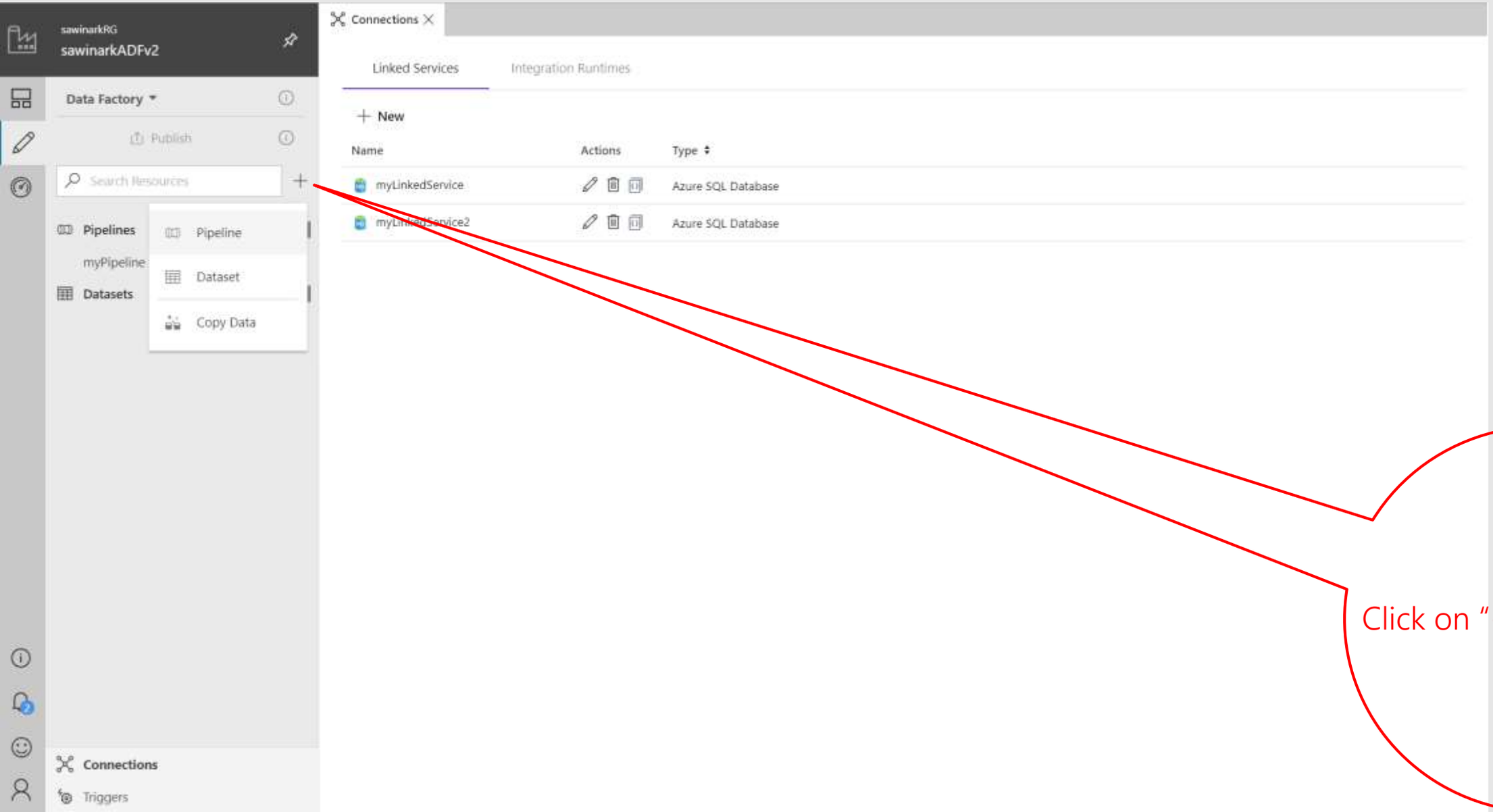
# Execution via ADFv2 Sproc Activity



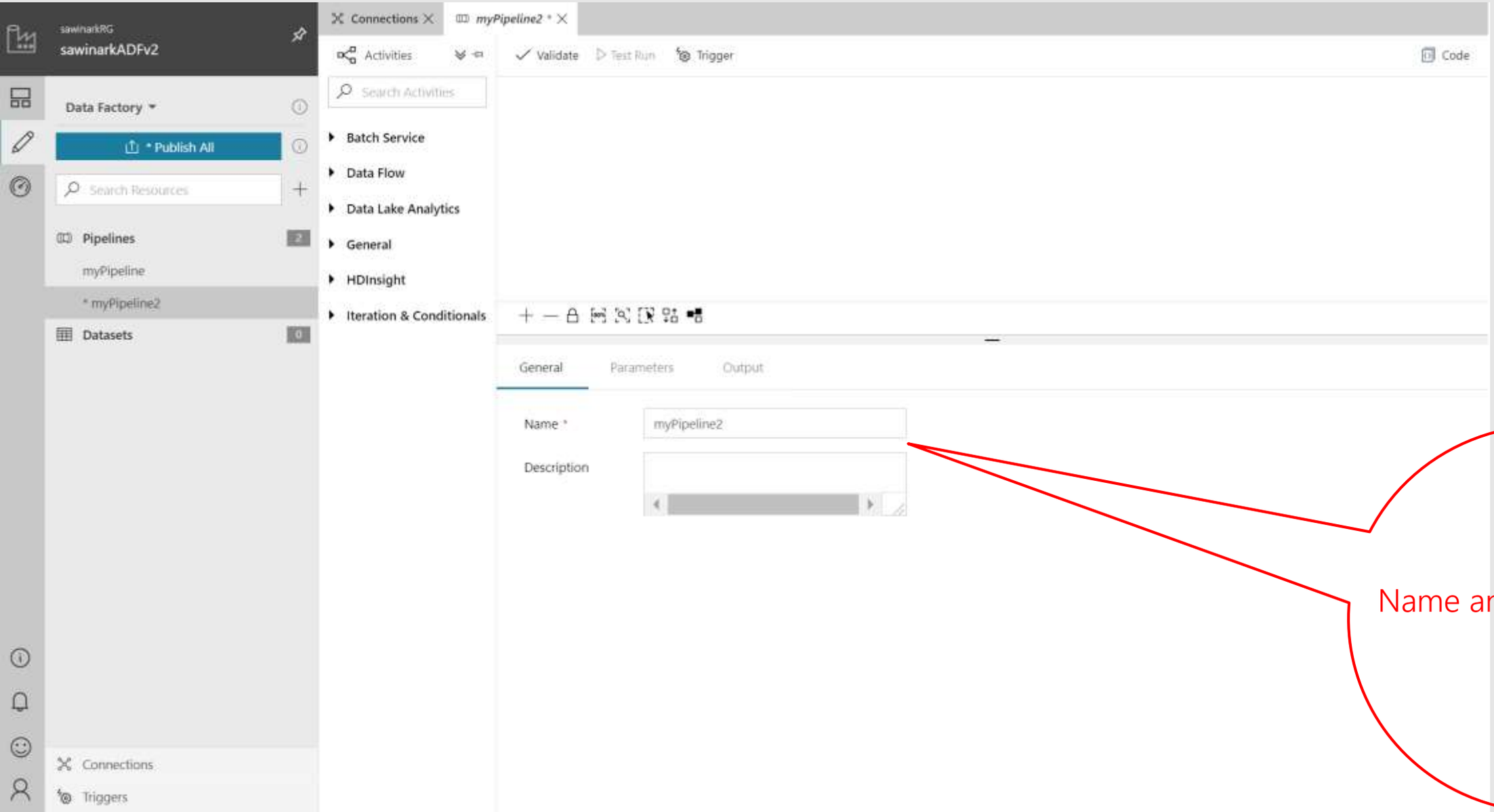
# Execution via ADFv2 Sproc Activity



# Execution via ADFv2 Sproc Activity

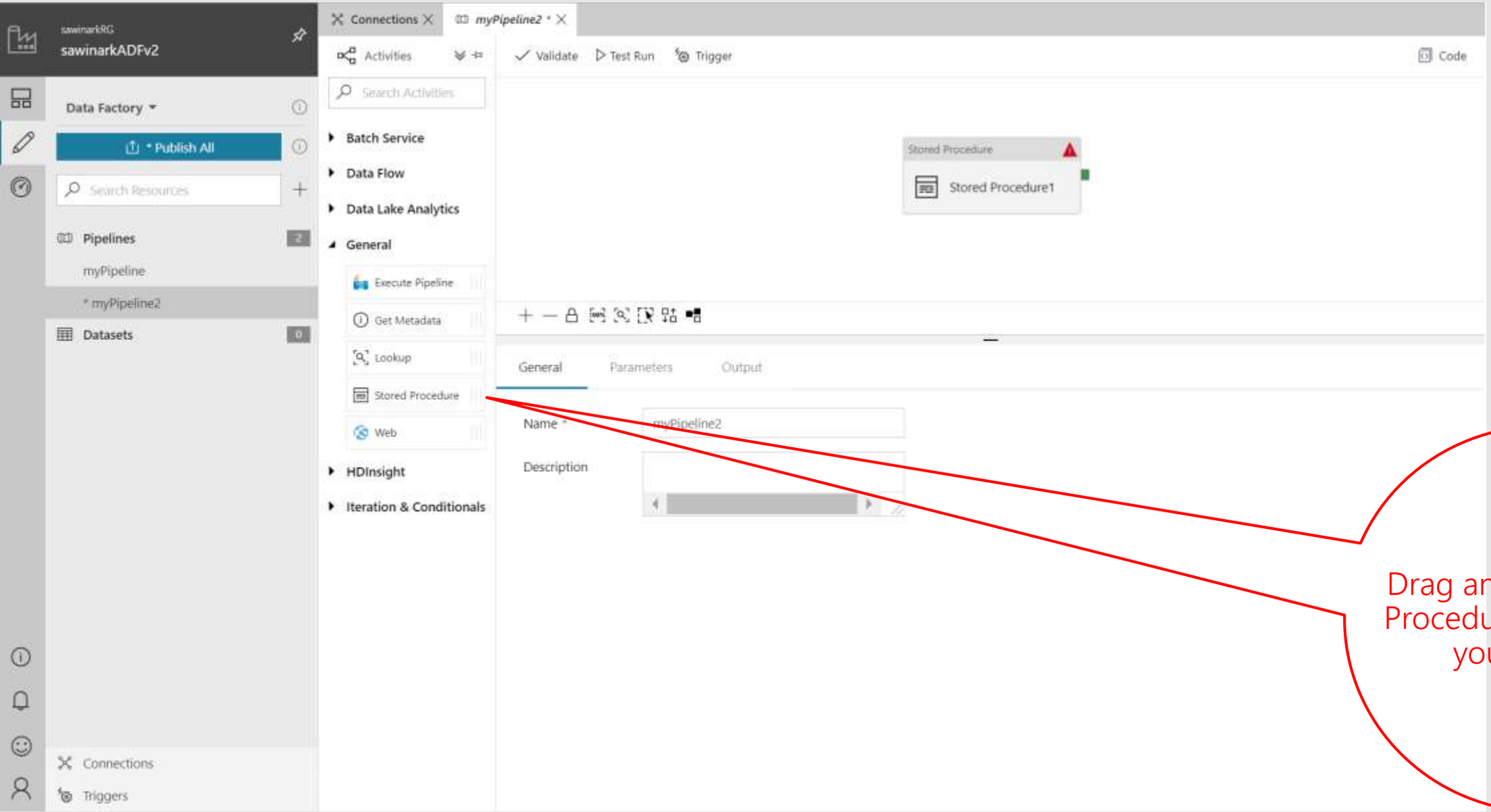


# Execution via ADFv2 Sproc Activity



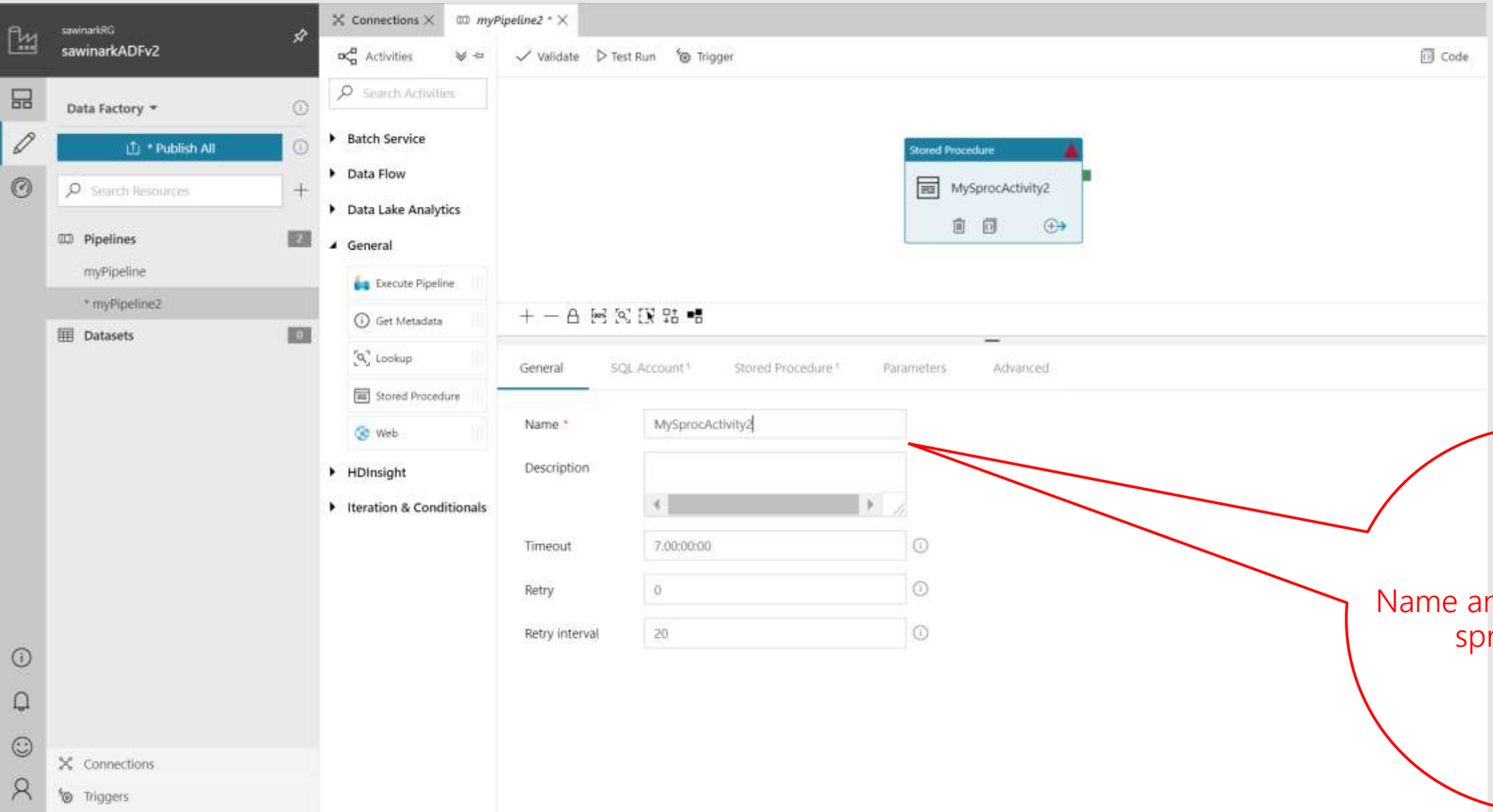

Name and describe your pipeline

# Execution via ADFv2 Sproc Activity



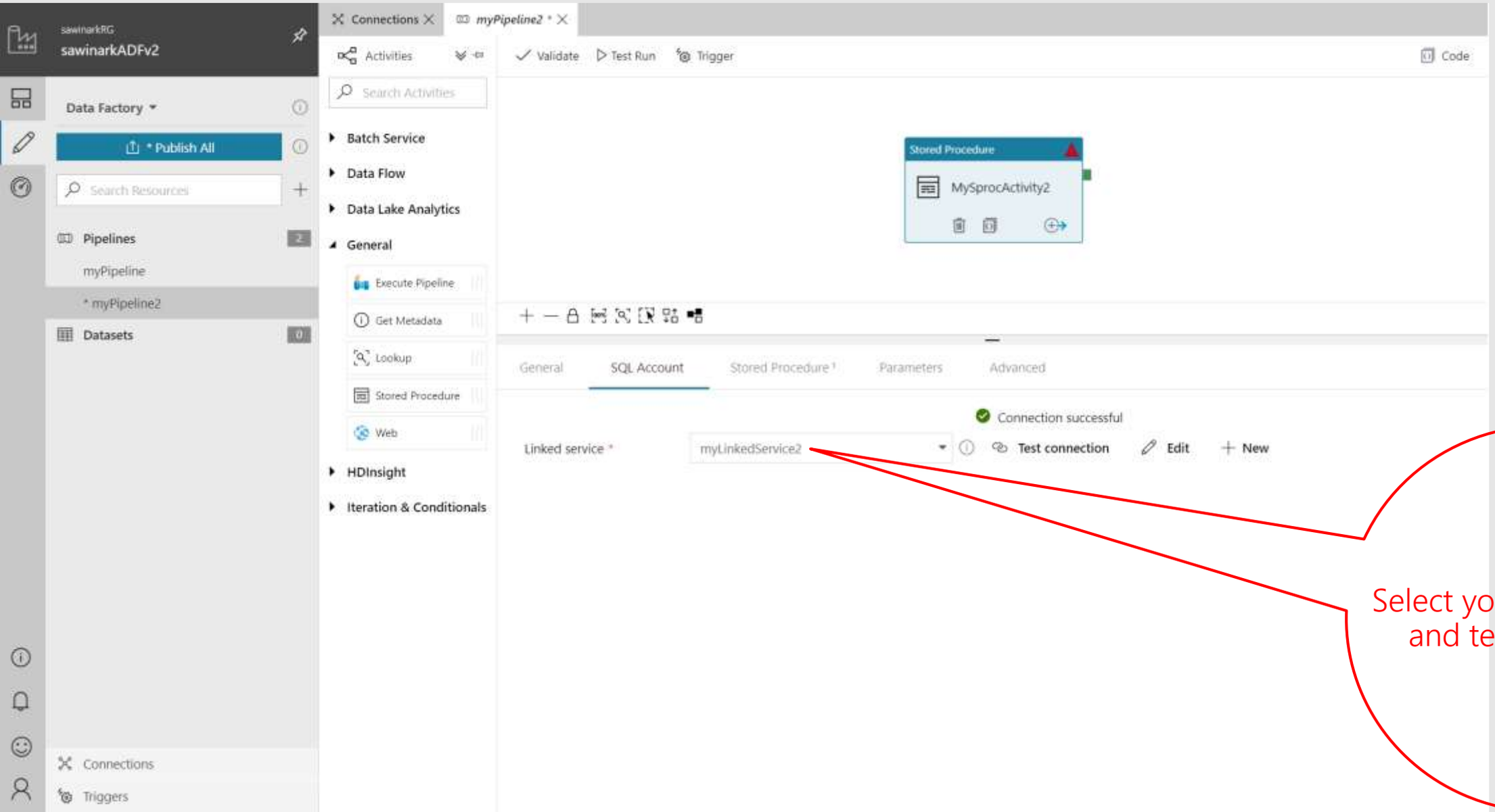

Drag and drop "Stored Procedure" activity into your pipeline

# Execution via ADFv2 Sproc Activity



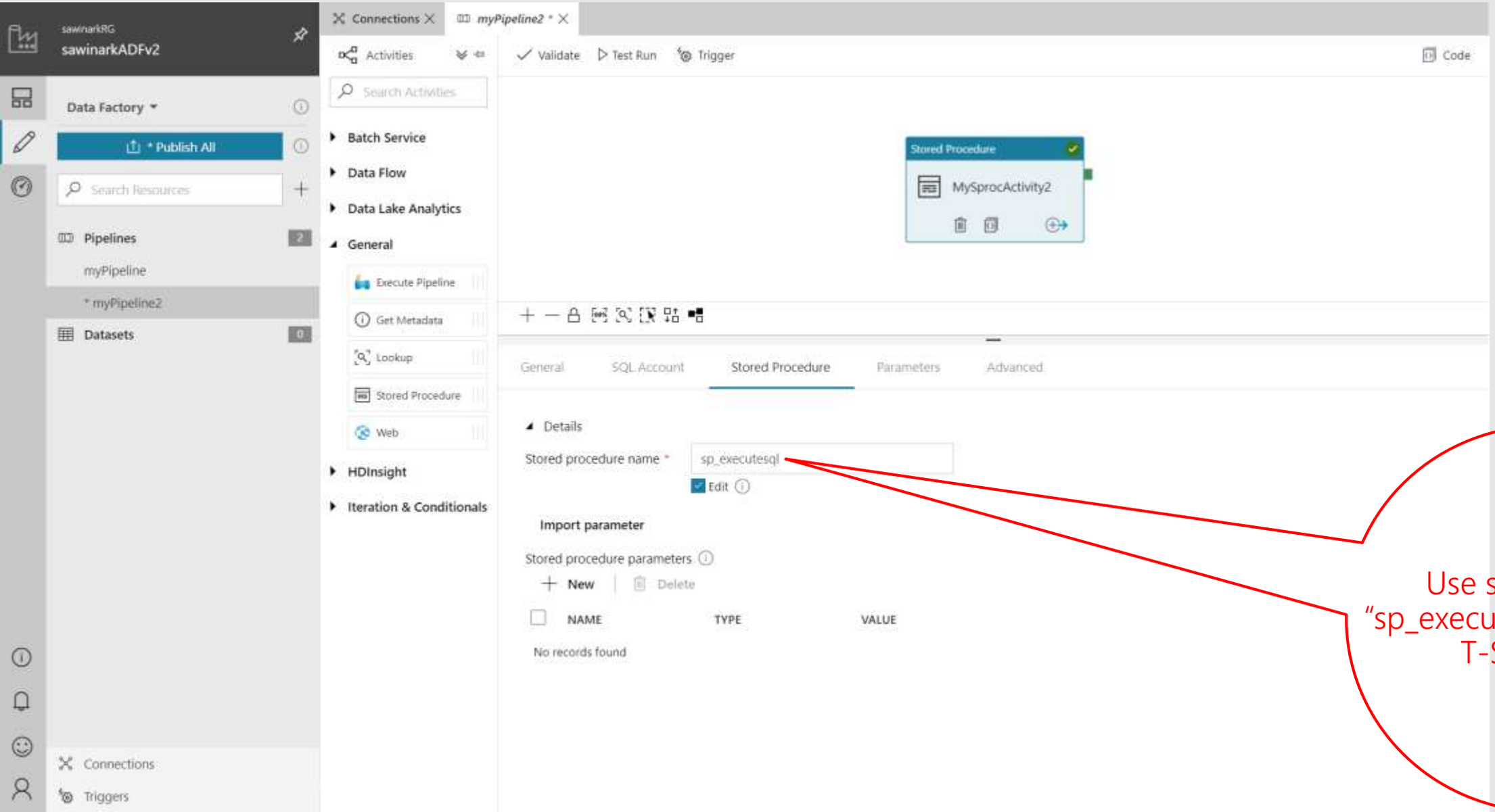

Name and describe your sproc activity

# Execution via ADFv2 Sproc Activity



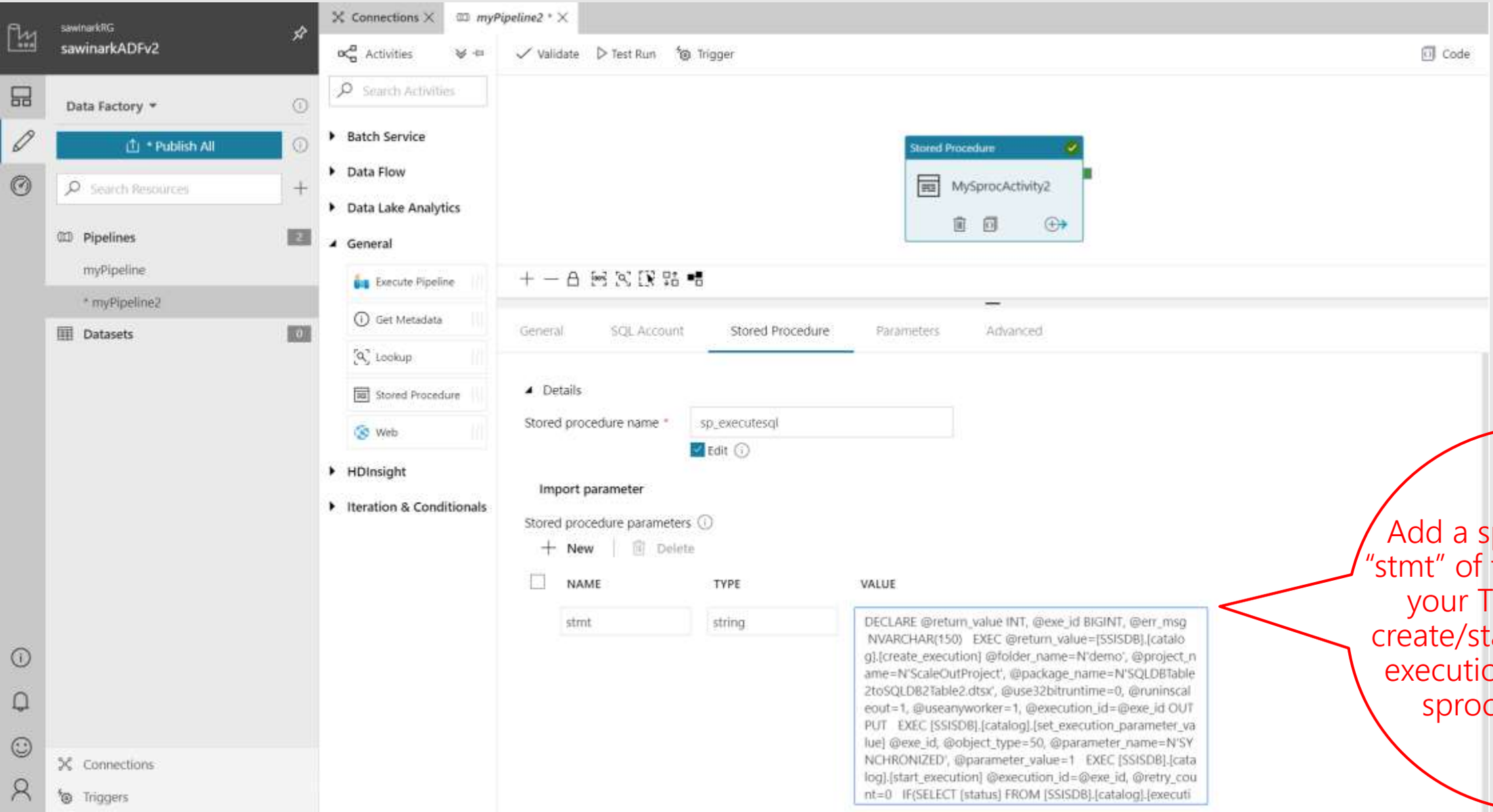

Select your linked service and test connection

# Execution via ADFv2 Sproc Activity



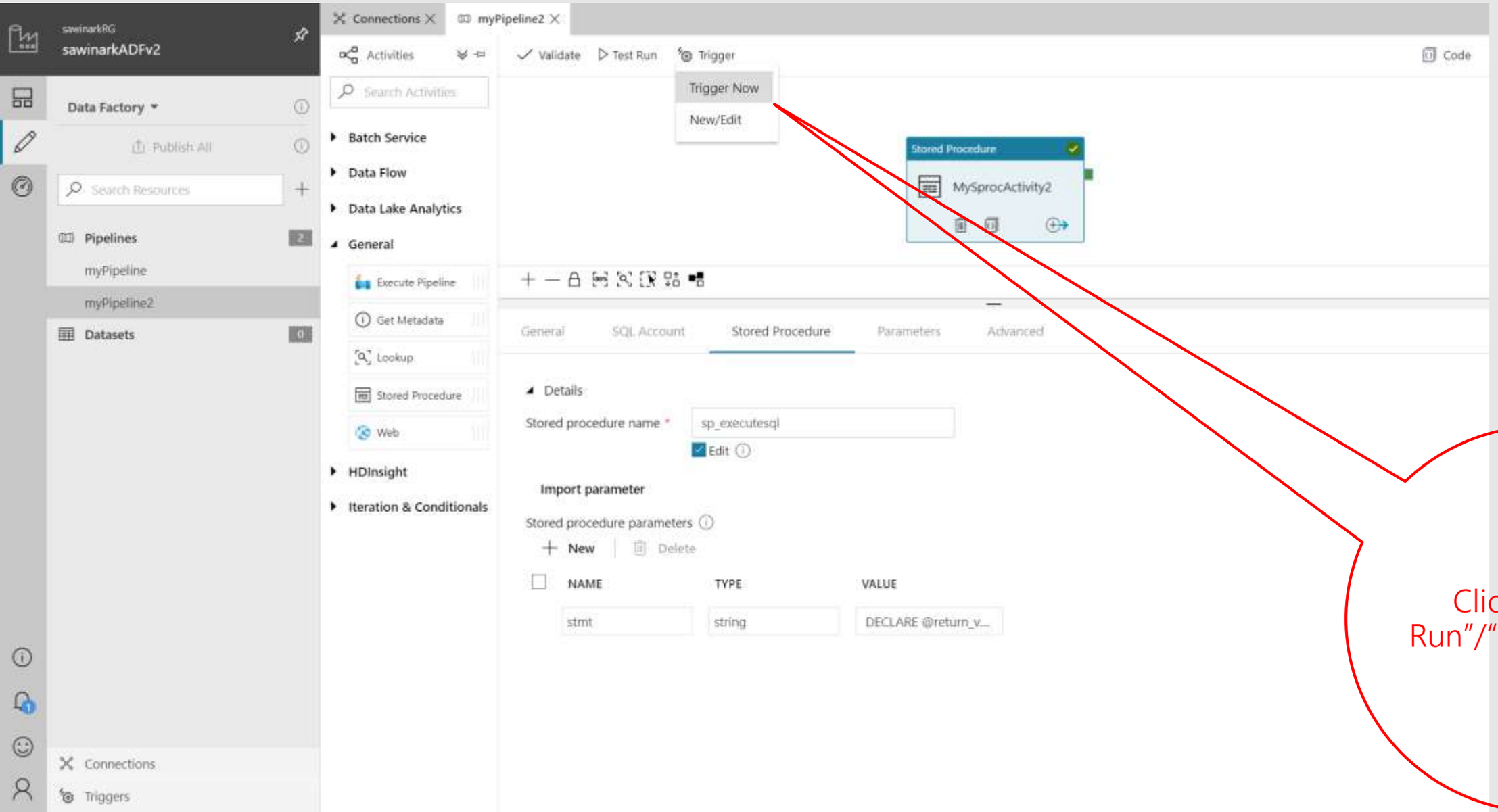

Use system sproc "sp_executesql" to execute T-SQL script

# Execution via ADFv2 Sproc Activity




Add a sproc parameter "stmt" of type "string" with your T-SQL script to create/start SSIS package execution using SSISDB sprocs as its value

# Execution via ADFv2 Sproc Activity




Click on “Test Run”/“Trigger Now”

# Execution via ADFv2 Sproc Activity

```
##### PSH script to invoke/trigger SSIS package executions in ADFv2 #####
# Create a linked service for your Azure SQL Database/Managed Instance server hosting SSISDB
Set-AzureRmDataFactoryV2LinkedService -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -Name "myLinkedService" -File "C:\ADF\SSIS\myLinkedService.json"

# Create a pipeline with sproc activity to execute your SSIS package(s)
Set-AzureRmDataFactoryV2Pipeline -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -Name "myPipeline" -DefinitionFile "C:\ADF\SSIS\myPipeline.json"

# Run your pipeline on demand
$myPipelineRun = Invoke-AzureRmDataFactoryV2Pipeline -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -PipelineName "myPipeline"
```

# Execution via ADFv2 Sproc Activity

```
// JSON script to create a linked service for Azure SQL DB/MI server hosting SSISDB
{
	"name": "myLinkedService",
	"properties": {
		"type": "AzureSqlDatabase",
		"typeProperties": {
			"connectionString": {
				"type": "SecureString",
				"value": "Server=tcp:YourAzureSQLDBServer.database.windows.net/YourAzureSQLMIServerEndpoint,1433;Database=SSISDB;User ID=YourUsername;Password=YourPassword;Trusted_Connection=False;Encrypt=True;Connection Timeout=30"
			}
		}
	}
}
```

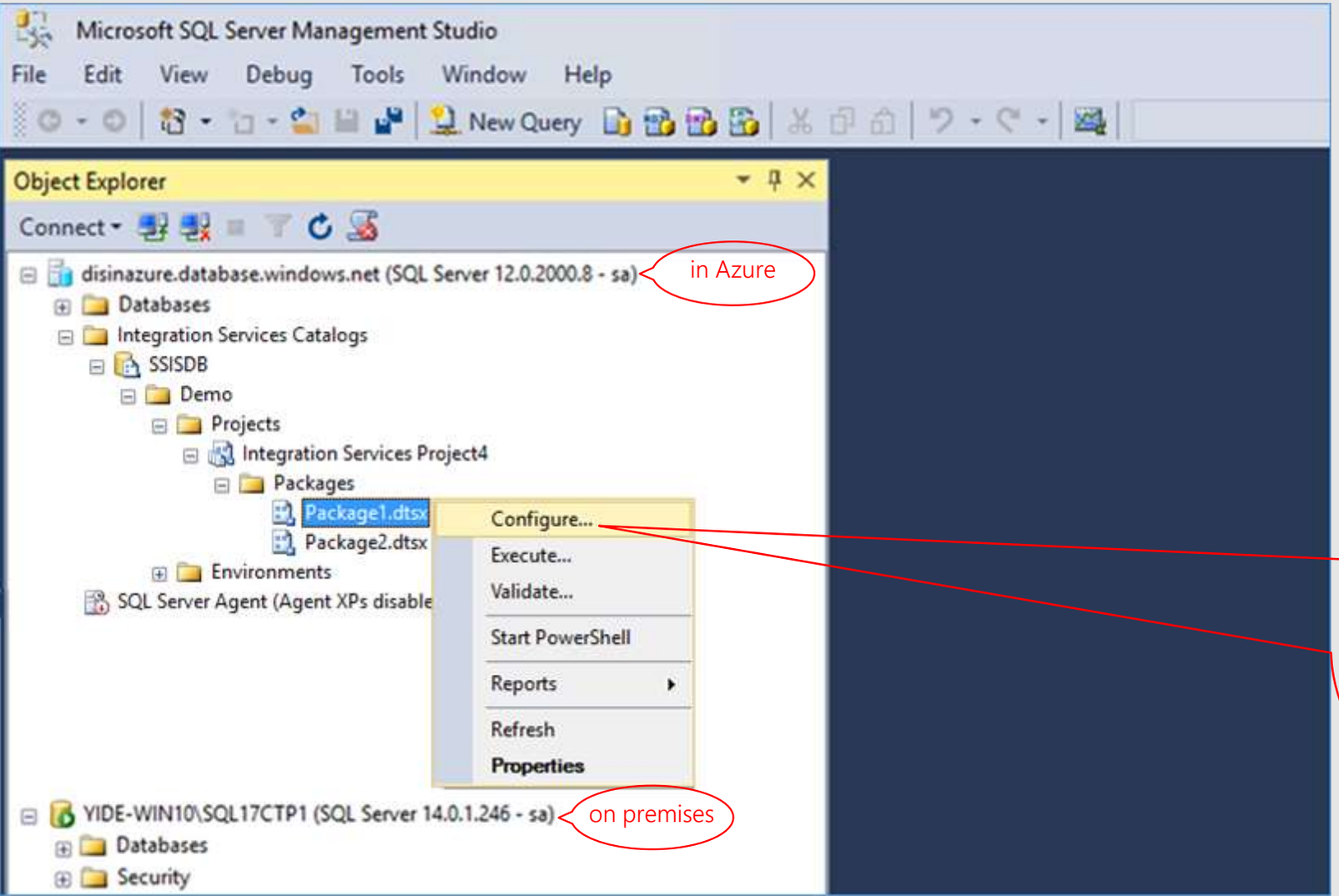
# Execution via ADFv2 Sproc Activity

```
// JSON script to create a pipeline with SqlServerStoredProcedure activity
{
        "name": "myPipeline",
        "properties": {
                "activities": [ {
                        "name": "mySProcActivity",
                        "description":"Sproc Activity to execute SSIS package(s)",
                        "type": "SqlServerStoredProcedure",
                        "linkedServiceName": {
                                "referenceName": "myLinkedService",
                                "type": "LinkedServiceReference"
                        },
                        "typeProperties": {
                                "storedProcedureName": "sp_executesql",
                                "storedProcedureParameters": {
                                        "stmt": {
                                                "value": "DECLARE @return_value INT, @exe_id BIGINT, @err_msg
NVARCHAR(150) EXEC @return_value=[SSISDB].[catalog].[create_execution] @folder_name=N'YourFolder', @project_name=N'YourProject',
@package_name=N'YourPackage', @use32bitruntime=0, @runinscaleout=1, @useanyworker=1, @execution_id=@exe_id OUTPUT EXEC
[SSISDB].[catalog].[set_execution_parameter_value] @exe_id, @object_type=50, @parameter_name=N'SYNCHRONIZED', @parameter_value=1 EXEC
[SSISDB].[catalog].[start_execution] @execution_id=@exe_id, @retry_count=0   IF(SELECT [status] FROM [SSISDB].[catalog].[executions]
WHERE execution_id=@exe_id)<>7 BEGIN SET @err_msg=N'Your package execution did not succeed for execution ID: ' + CAST(@exe_id AS
NVARCHAR(20)) RAISERROR(@err_msg,15,1) END"
                                        }
                                }
                        }
                } ]
        }
}
```

# Execution via ADFv2 Sproc Activity

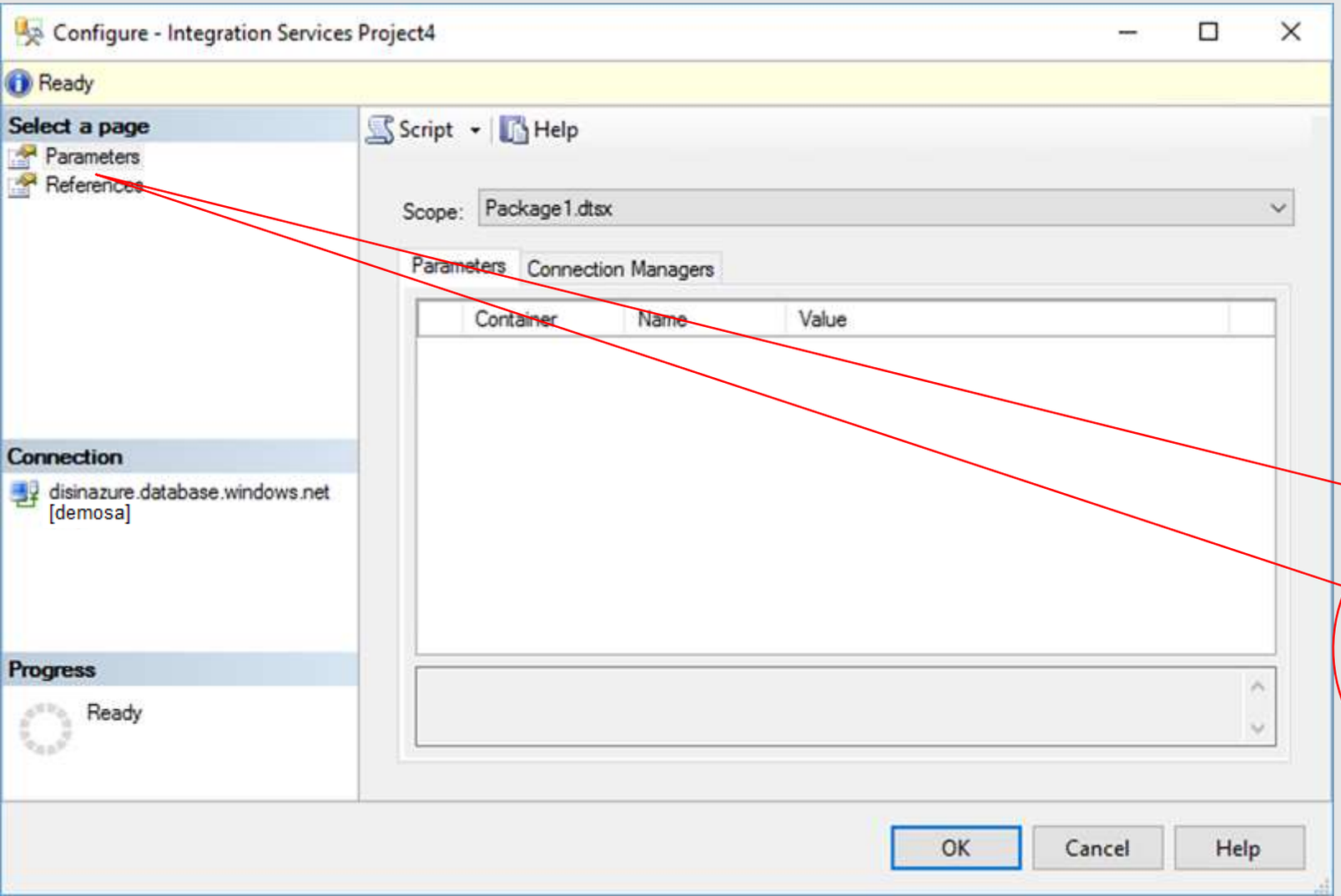
```
-- T-SQL script to create/start SSIS package execution using SSISDB sprocs
DECLARE   @return_value int, @exe_id bigint, @err_msg nvarchar(150)

EXEC @return_value = [SSISDB].[catalog].[create_execution] @folder_name=N'YourFolder', @project_name=N'YourProject',
        @package_name=N'YourPackage', @use32bitruntime=0, @runincluster=1, @useanyworker=1,
        @execution_id=@exe_id OUTPUT

-- To synchronize SSIS package execution, set SYNCHRONIZED execution parameter
EXEC [SSISDB].[catalog].[set_execution_parameter_value] @exe_id, @object_type=50, @parameter_name=N'SYNCHRONIZED',
@parameter_value=1

EXEC [SSISDB].[catalog].[start_execution] @execution_id = @exe_id, @retry_count = 0

-- Raise an error for unsuccessful package execution, check package execution status = created (1)/running
(2)/canceled (3)/failed (4)/
-- pending (5)/ended unexpectedly (6)/succeeded (7)/stopping (8)/completed (9)
IF (SELECT [status] FROM [SSISDB].[catalog].[executions] WHERE execution_id = @exe_id)<>7
BEGIN
        SET @err_msg=N'Your package execution did not succeed for execution ID: '+ CAST(@execution_id as
nvarchar(20))
        RAISERROR(@err_msg, 15, 1)
END
```

# Execution via SSMS



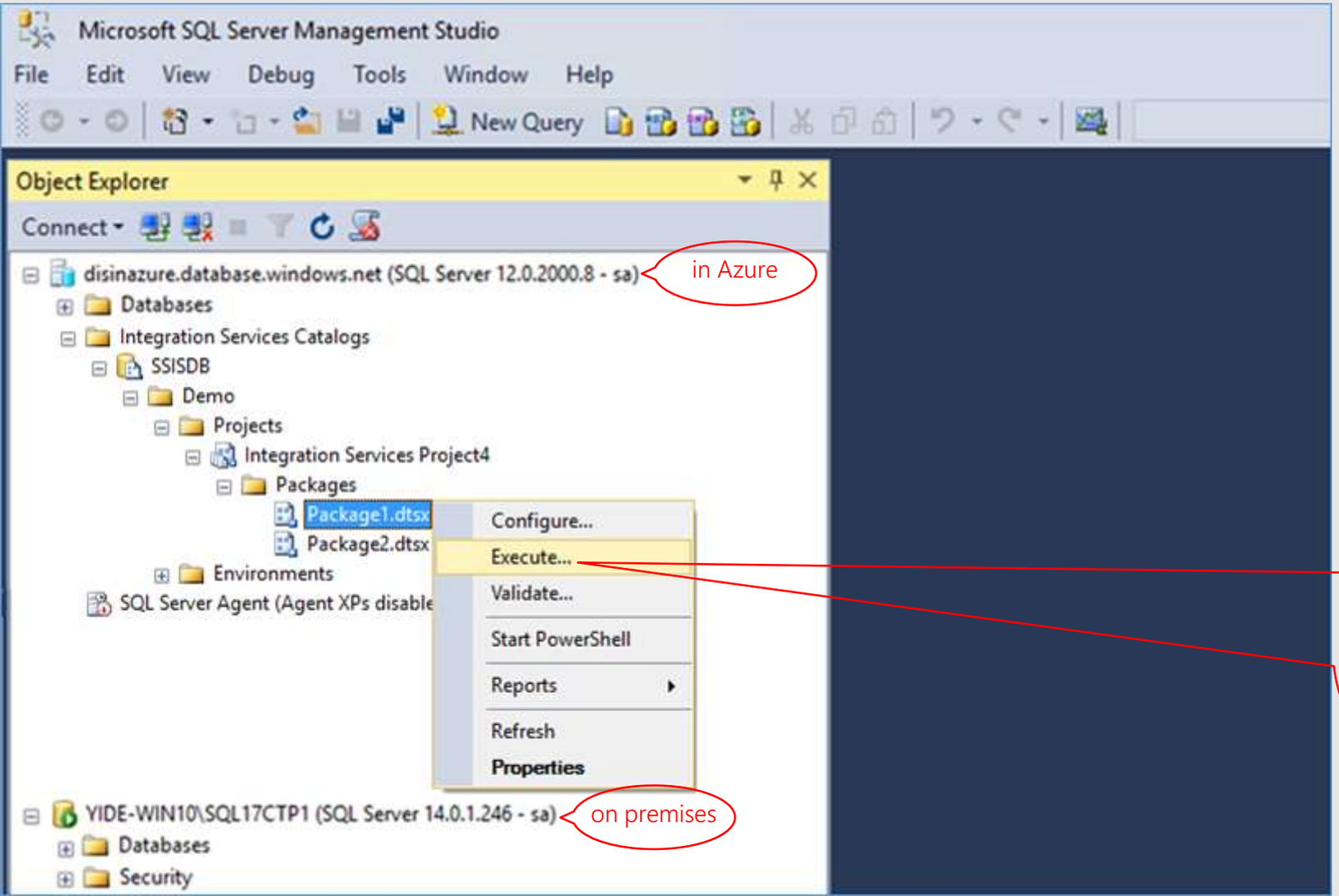

Once deployed, you can configure packages for execution on SSIS PaaS

# Execution via SSMS



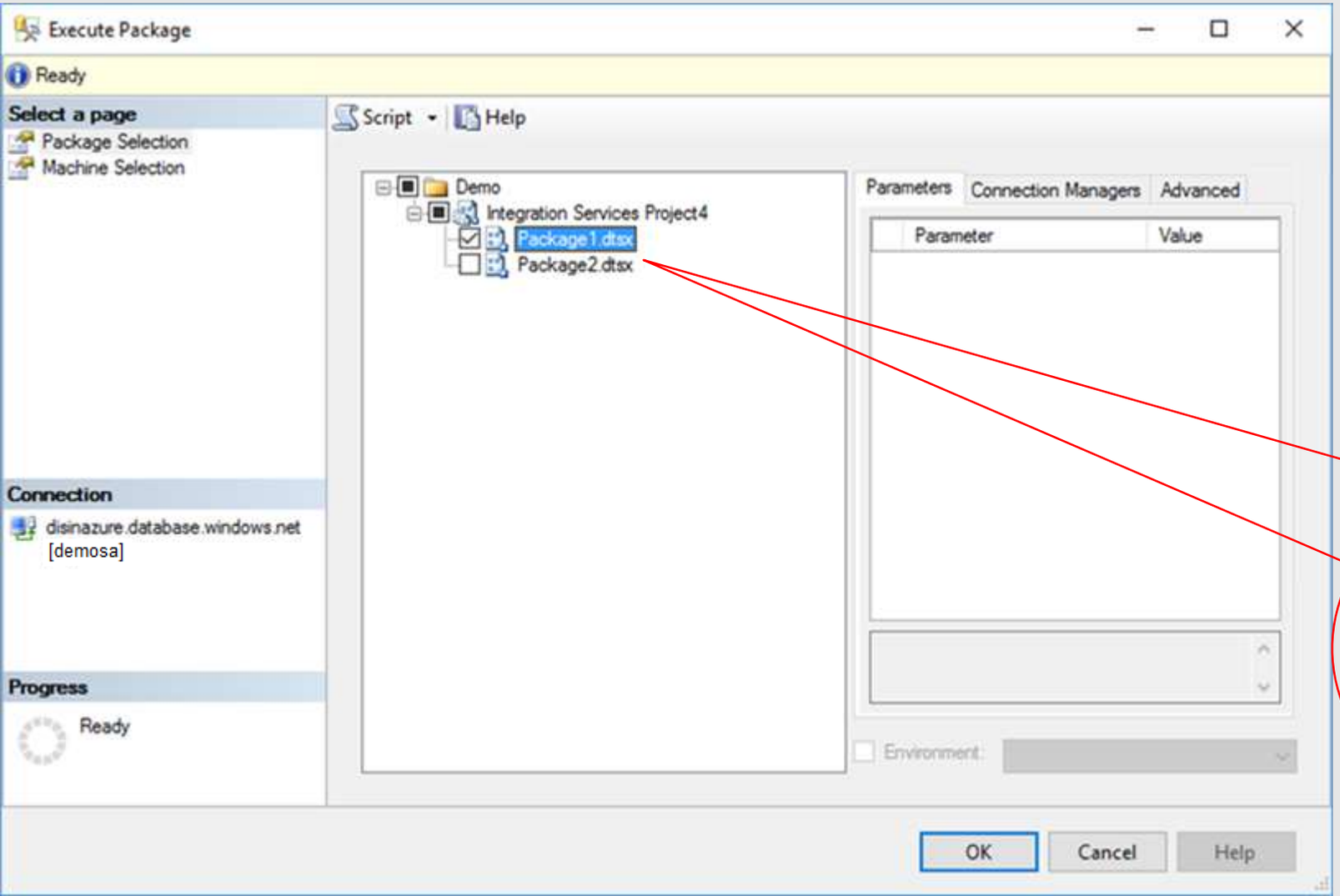

You can set package run-time parameters/environment references

# Execution via SSMS




Once configured, you can execute packages on SSIS PaaS

# Execution via SSMS




You can select some packages to execute on SSIS PaaS

# Scheduling Methods

# Scheduling Methods

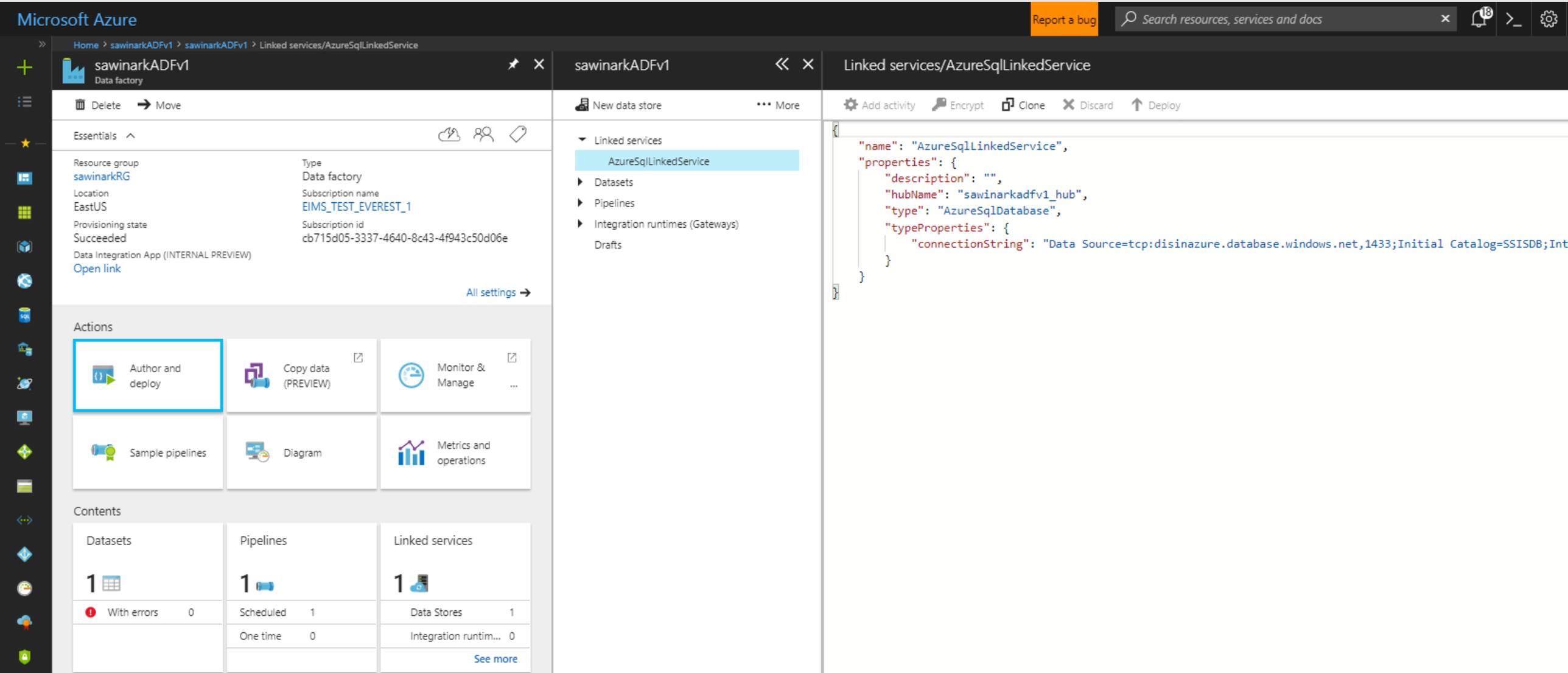
- SSIS package executions can be directly scheduled as first-class SSIS activities in ADFv2 pipelines (Work in Progress)
  - For now, SSIS package executions can be indirectly scheduled via ADFv1/v2 Sproc Activity
- If you use Azure SQL MI server to host SSISDB
  - SSIS package executions can also be scheduled via Azure SQL MI Agent (Extended Private Preview)
- If you use Azure SQL DB server to host SSISDB
  - SSIS package executions can also be scheduled via Elastic Jobs (Private Preview)
- If you keep on-prem SQL Server
  - SSIS package executions can also be scheduled via on-prem SQL Server Agent

# Scheduling via ADFv1 Sproc Activity

- Create a linked service for Azure SQL DB/MI server hosting SSISDB
- Create an output dataset that drives scheduling
- Create a pipeline with SqlServerStoredProcedure activity

# Scheduling via ADFv1 Sproc Activity

- Create a linked service for Azure SQL DB/MI server hosting SSISDB

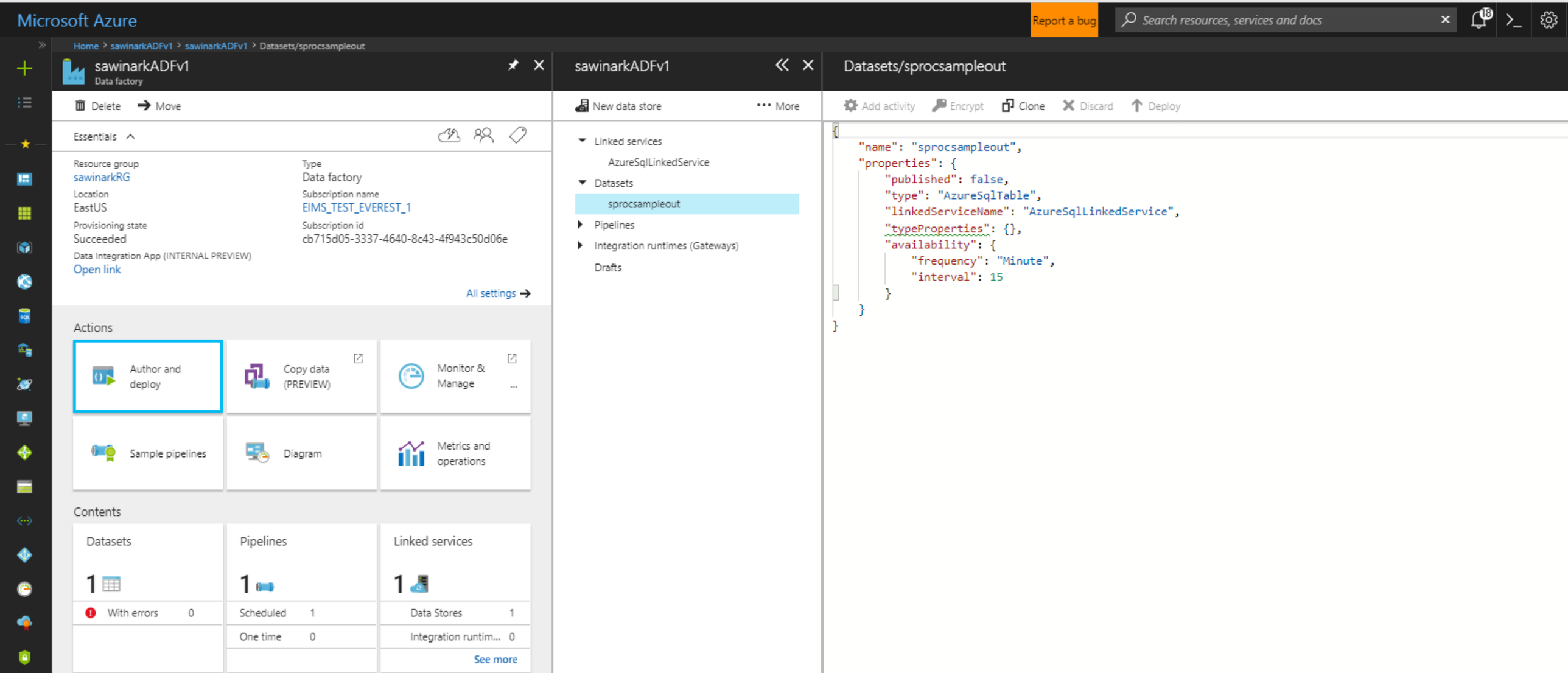


# Scheduling via ADFv1 Sproc Activity

```
// JSON script to create a linked service for Azure SQL DB/MI server hosting SSISDB
{
	"name": "AzureSqlLinkedService",
	"properties": {
		"description": "",
		"type": "AzureSqlDatabase",
		"typeProperties": {
			"connectionString":"Data Source=tcp:YourAzureSQLDBServer.database.windows.net/YourAzureSQLMIServerEndpoint,1433;Initial Catalog=SSISDB;User ID=YourUsername;Password=YourPassword;Integrated Security=False;Encrypt=True;Connect Timeout=30"
		}
	}
}
```

# Scheduling via ADFv1 Sproc Activity

- Create an output dataset that drives scheduling

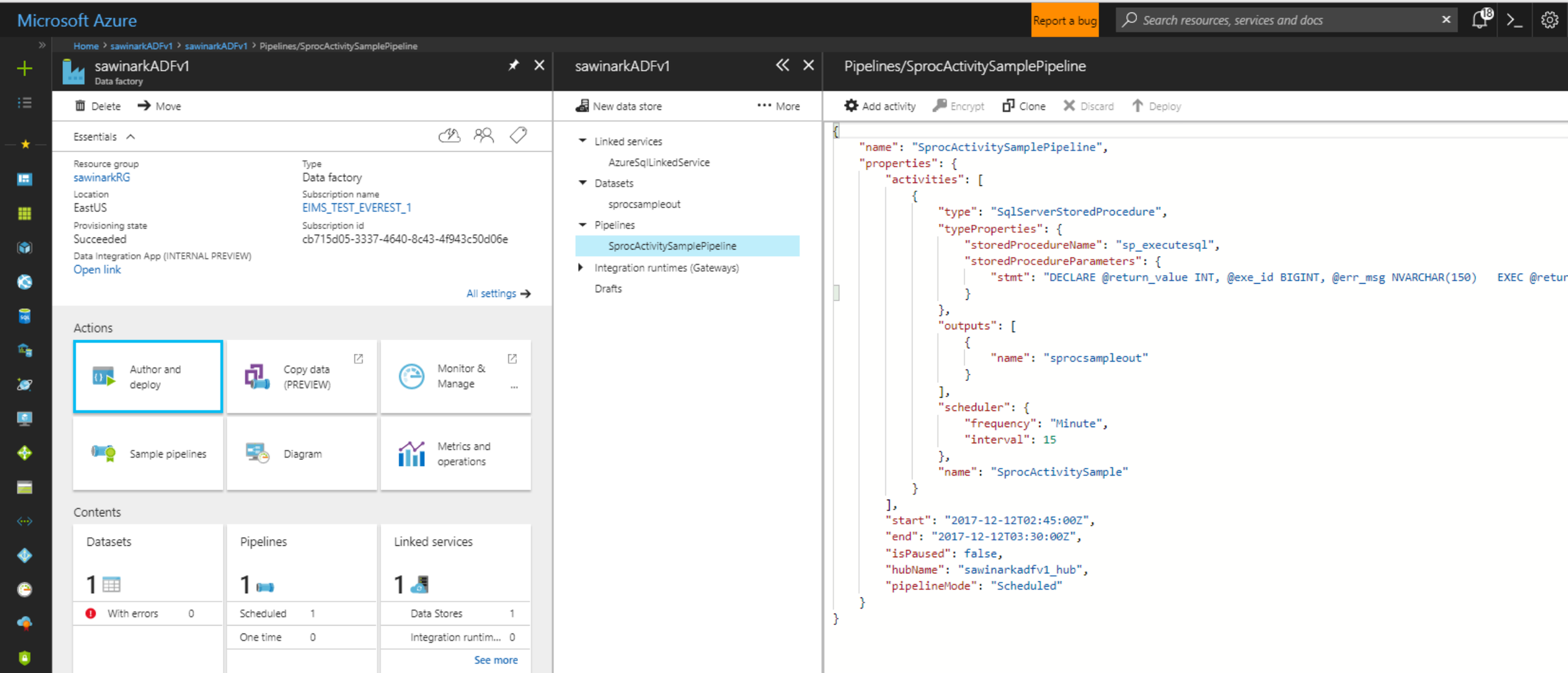


# Scheduling via ADFv1 Sproc Activity

```
// JSON script to create an output dataset that drives scheduling
{
      "name": "sprocsampleout",
      "properties": {
            "type": "AzureSqlTable",
            "linkedServiceName": "AzureSqlLinkedService",
            "typeProperties": {},
            "availability": {
                  "frequency": "Hour",
                  "interval": 1
            }
      }
}
```

# Scheduling via ADFv1 Sproc Activity

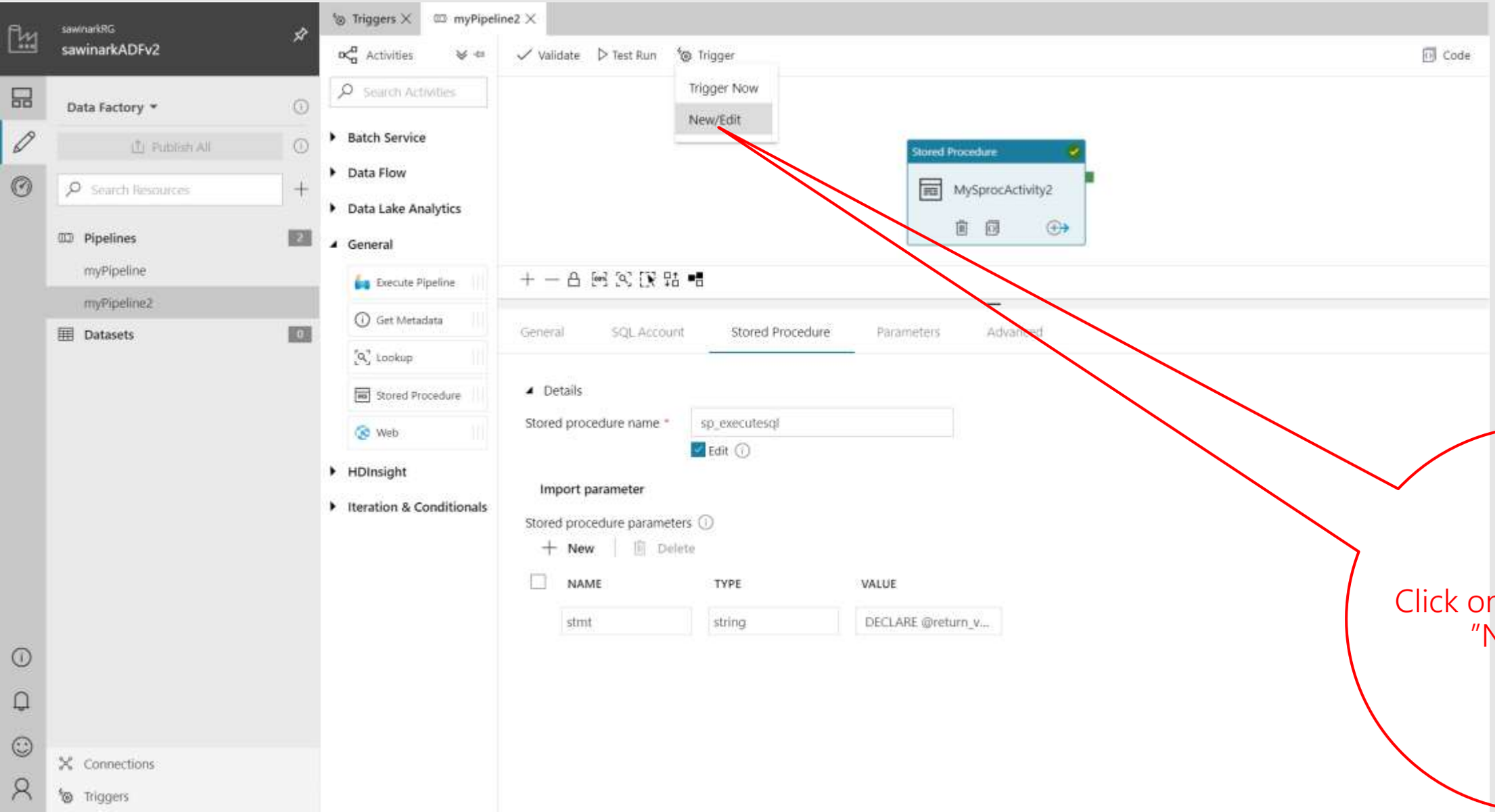
- Create a pipeline with SqlServerStoredProcedure activity



# Scheduling via ADFv1 Sproc Activity

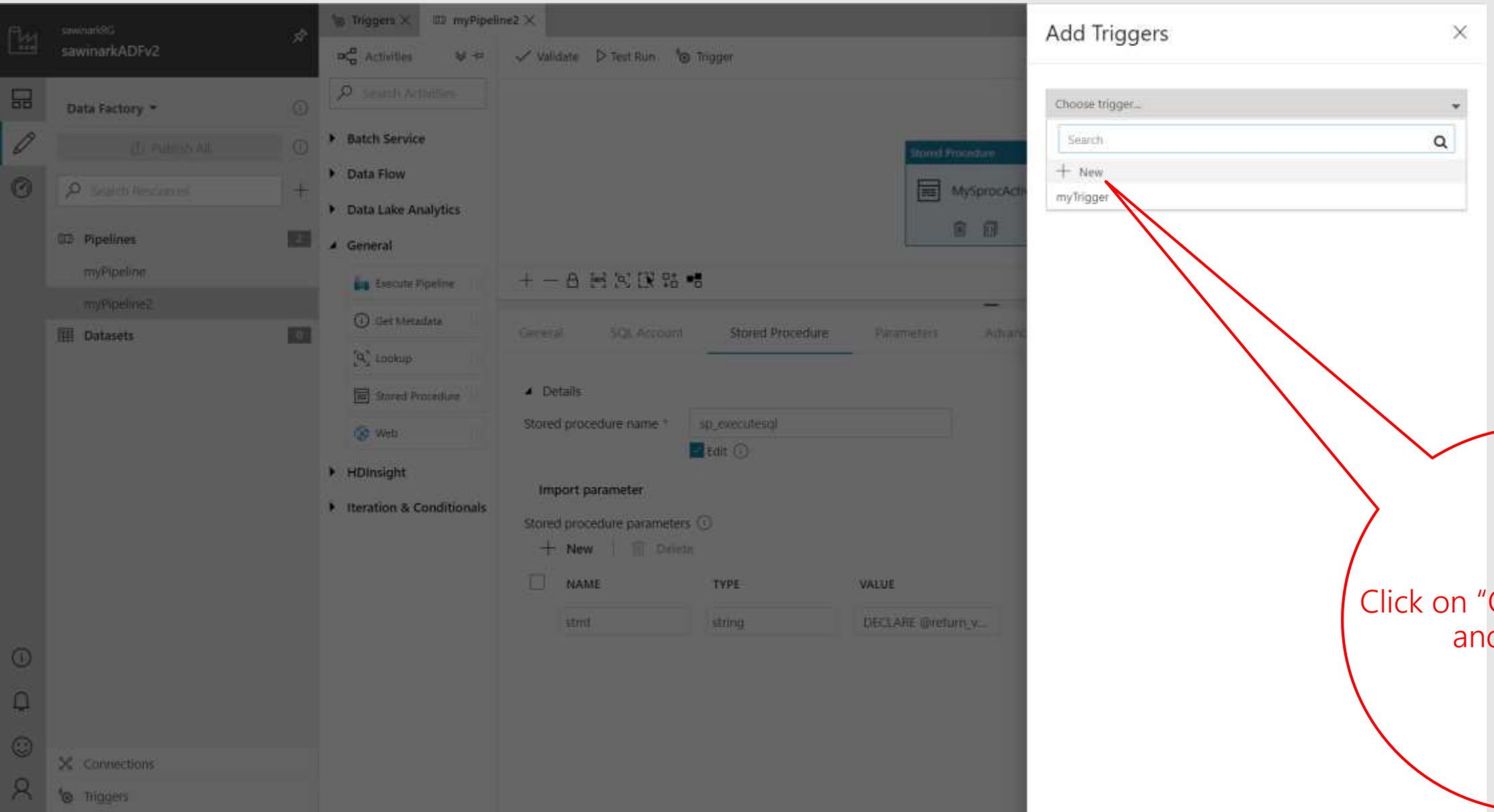
```
// JSON script to create a pipeline with SqlServerStoredProcedure activity
{
            "name": "SprocActivitySamplePipeline",
            "properties": {
                        "activities": [ {
                                    "name": "SprocActivitySample“,
                                    "type": "SqlServerStoredProcedure",
                                    "typeProperties": {
                                                "storedProcedureName": "sp_executesql",
                                                "storedProcedureParameters": {
                                                            "stmt": "DECLARE @return_value INT, @exe_id BIGINT, @err_msg NVARCHAR(150) EXEC
@return_value=[SSISDB].[catalog].[create_execution] @folder_name=N'YourFolder', @project_name=N'YourProject', @package_name=N'YourPackage',
@use32bitruntime=0, @runinscaleout=1, @useanyworker=1, @execution_id=@exe_id OUTPUT EXEC [SSISDB].[catalog].[set_execution_parameter_value] @exe_id,
@object_type=50, @parameter_name=N'SYNCHRONIZED', @parameter_value=1 EXEC [SSISDB].[catalog].[start_execution] @execution_id=@exe_id, @retry_count=0
IF(SELECT [status] FROM [SSISDB].[catalog].[executions] WHERE execution_id=@exe_id)<>7 BEGIN SET @err_msg=N'Your package execution did not succeed for
execution ID: ' + CAST(@exe_id AS NVARCHAR(20)) RAISERROR(@err_msg,15,1) END"
                                                }
                                    },
                                    "outputs": [ {
                                                "name": "sprocsampleout"
                                    } ],
                                    "scheduler": {
                                                "frequency": "Hour",
                                                "interval": 1
                                    }
                        } ],
                        "start": "2017-04-02T00:00:00Z",
                        "end": "2017-04-02T05:00:00Z",
                        "isPaused": false
            }
}
```

# Scheduling via ADFv2 Sproc Activity

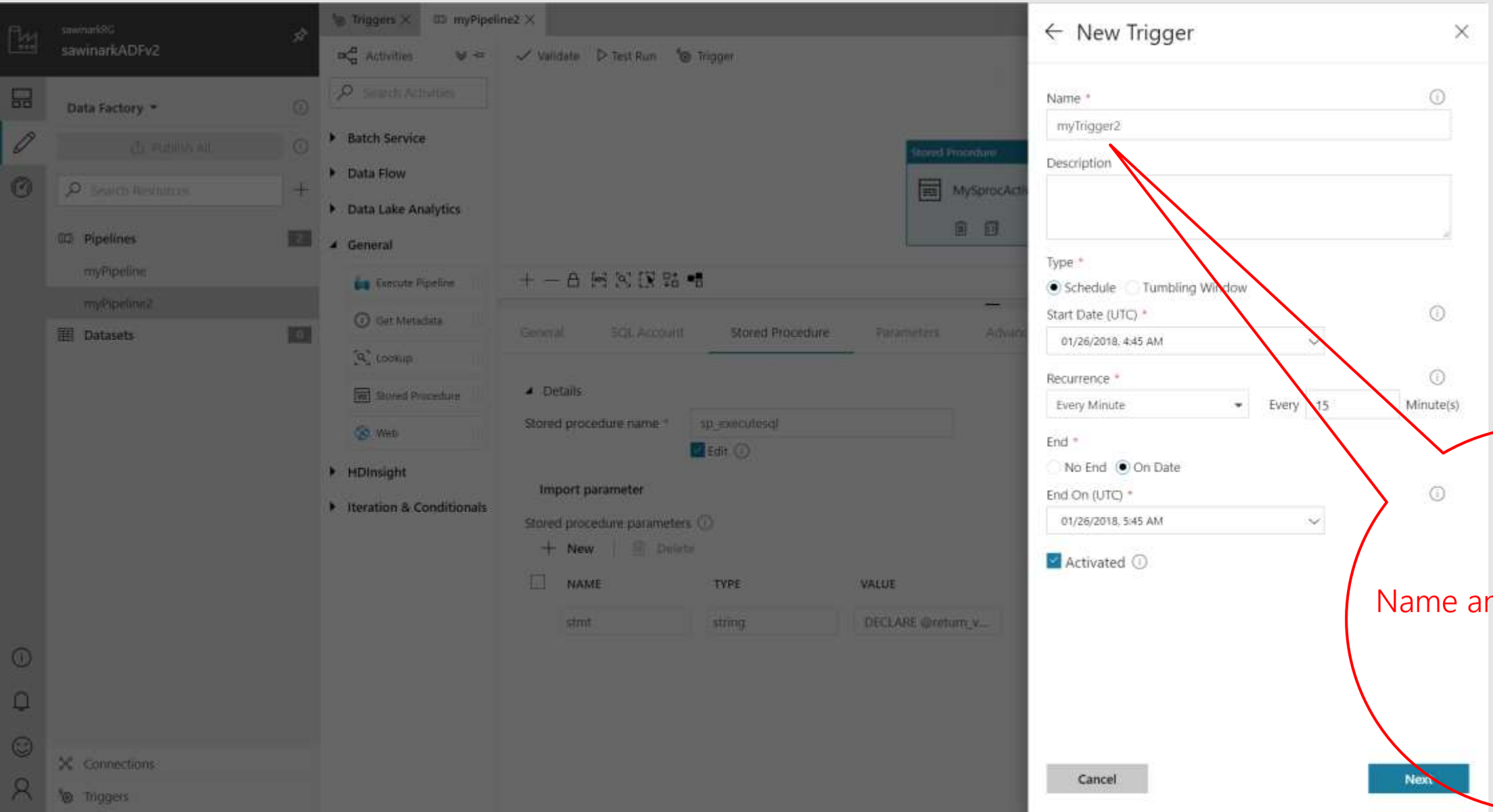



Click on "Trigger" and "New/Edit"

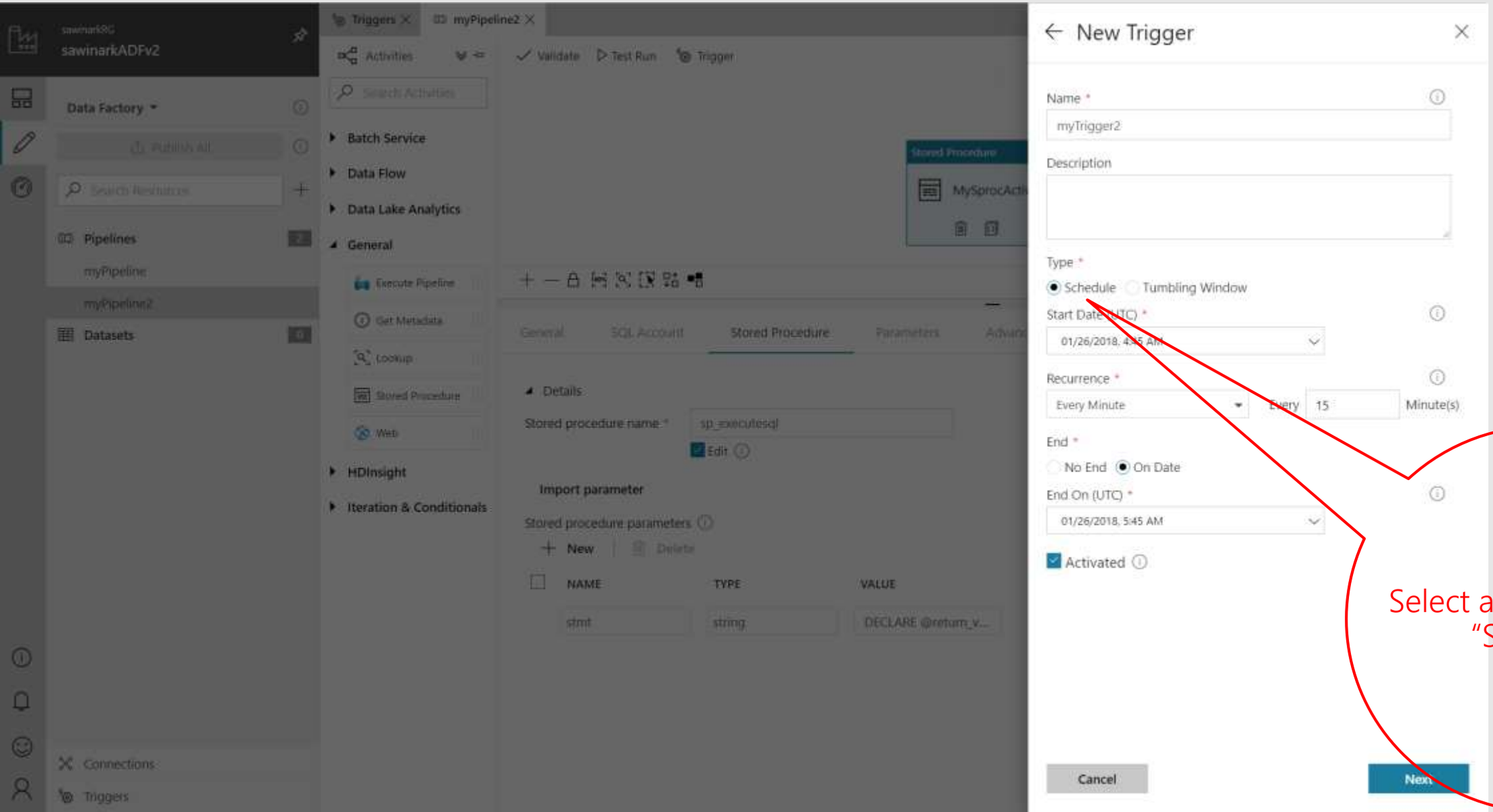
# Scheduling via ADFv2 Sproc Activity



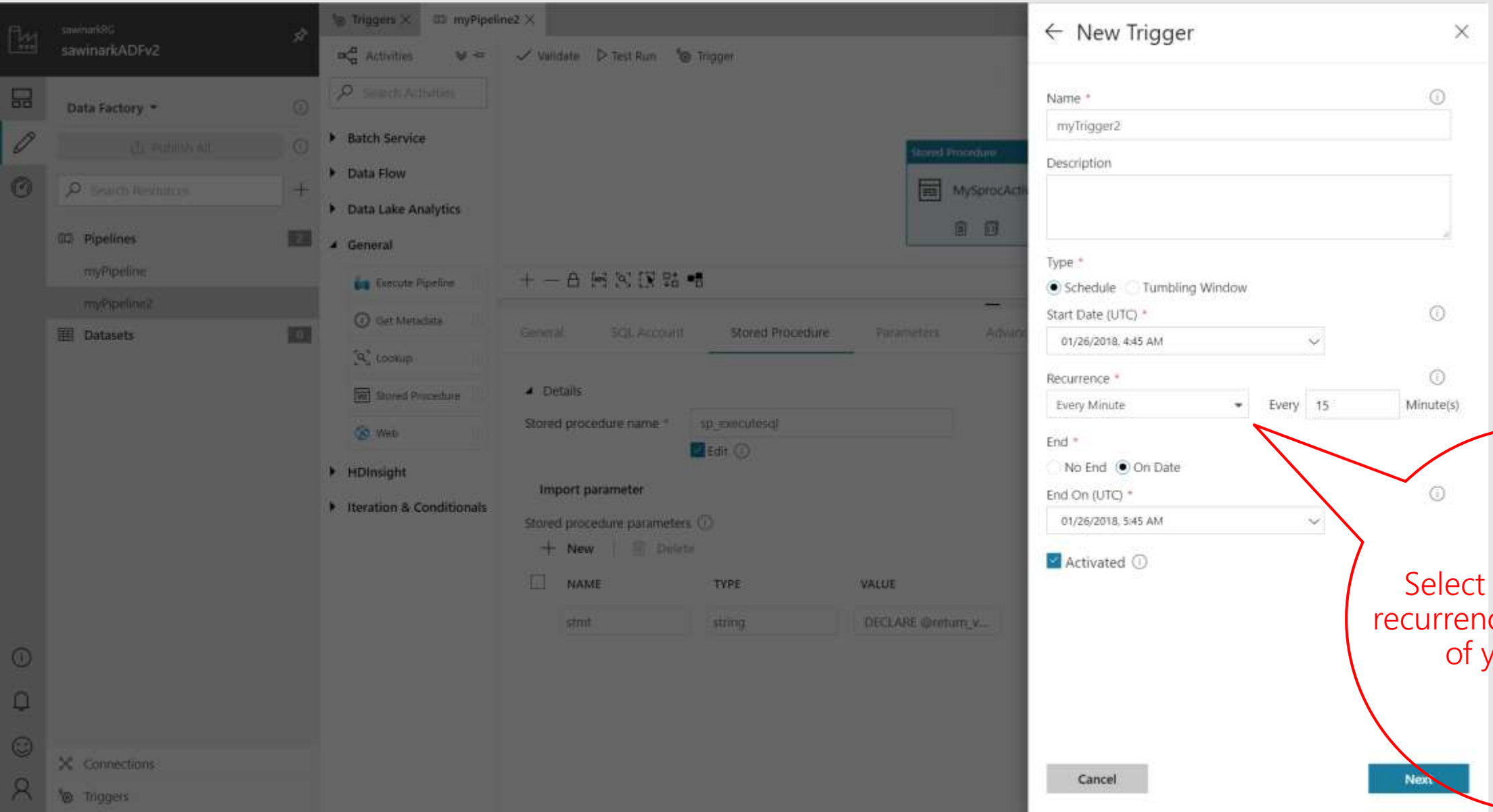
# Scheduling via ADFv2 Sproc Activity



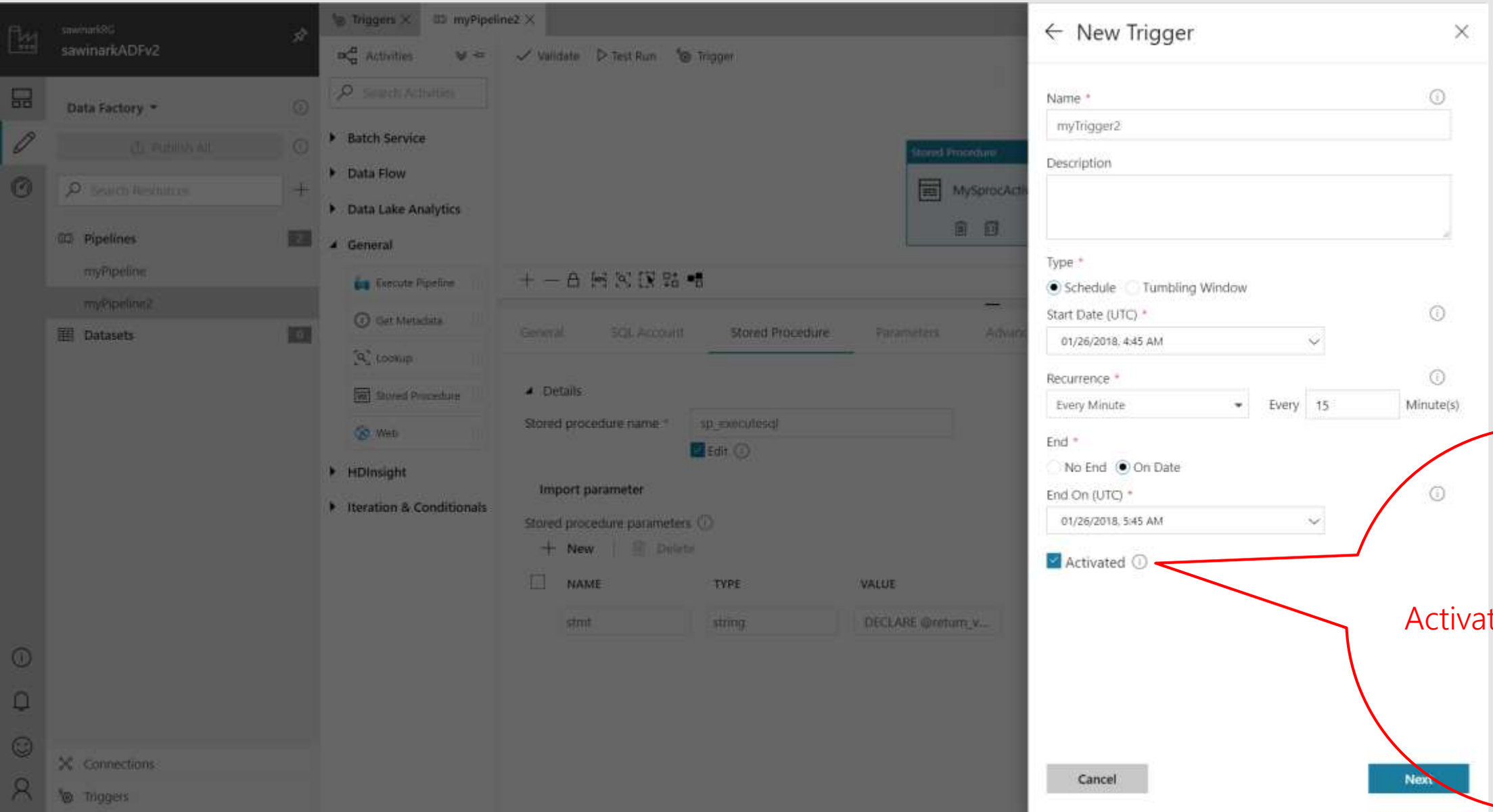
# Scheduling via ADFv2 Sproc Activity



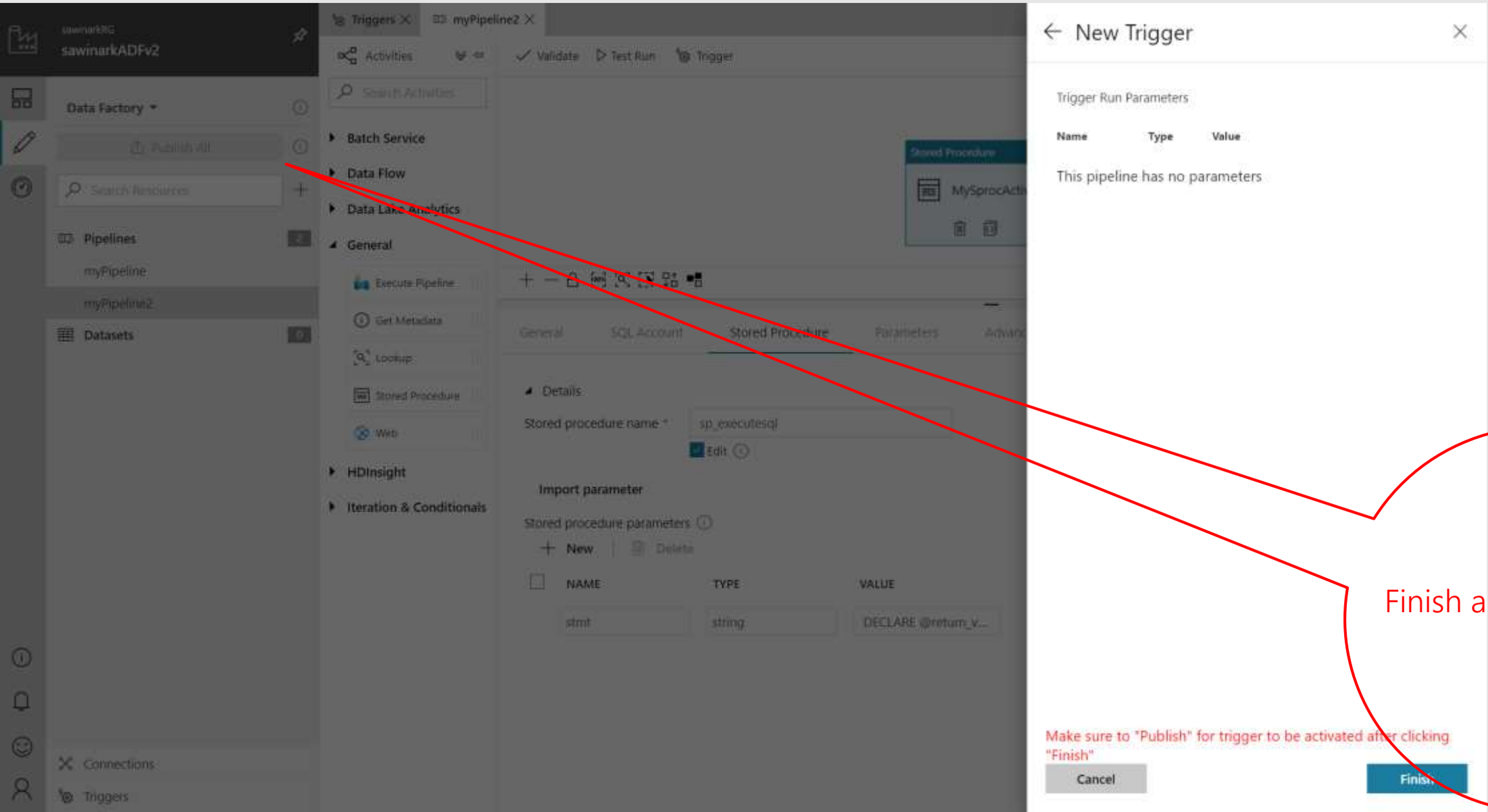
# Scheduling via ADFv2 Sproc Activity



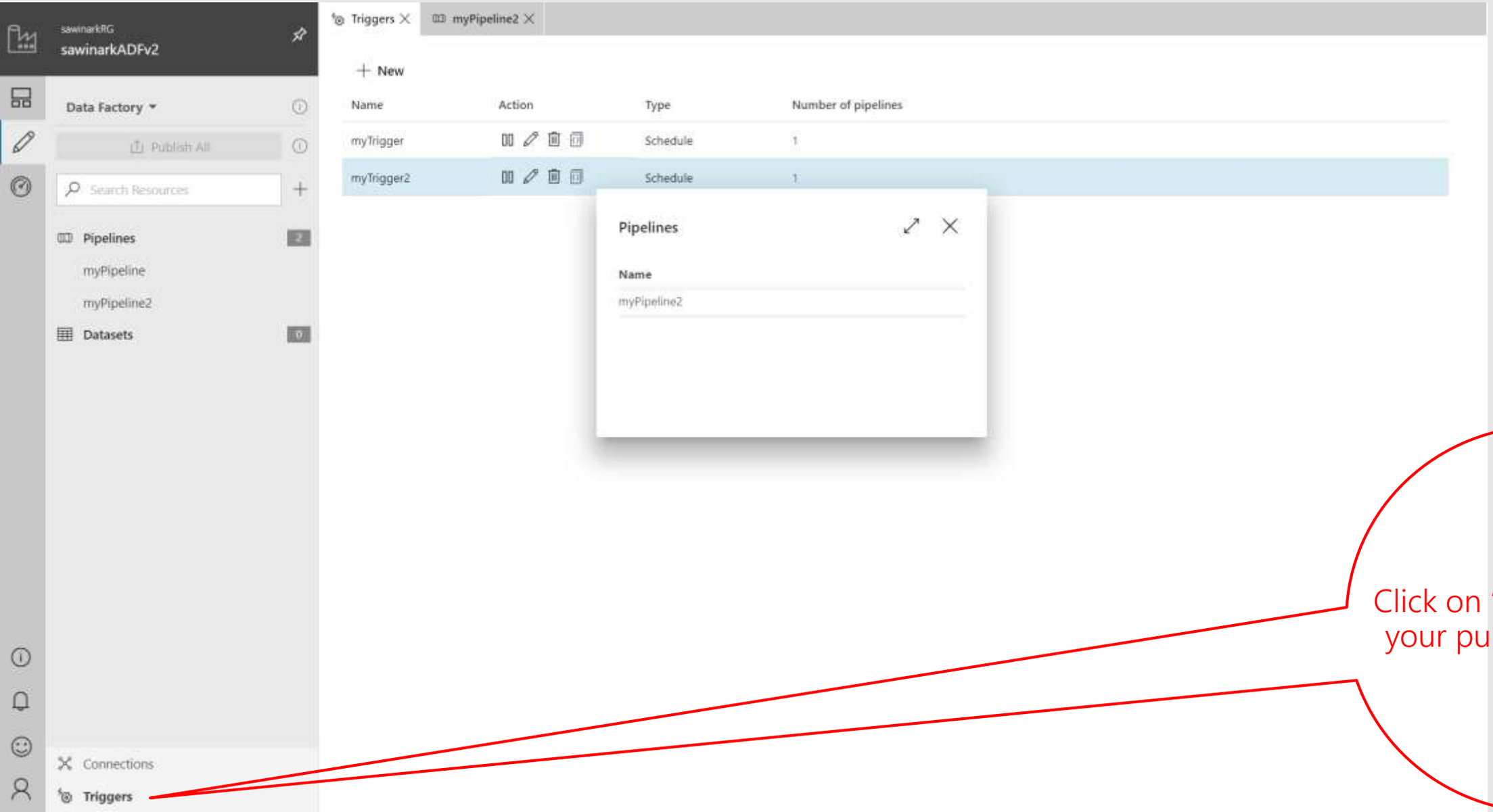
# Scheduling via ADFv2 Sproc Activity



# Scheduling via ADFv2 Sproc Activity



# Scheduling via ADFv2 Sproc Activity



# Scheduling via ADFv2 Sproc Activity

```
##### PSH script to schedule SSIS package executions in ADFv2 #####

# Create a trigger to schedule your pipeline runs
Set-AzureRmDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -Name "myTrigger" -DefinitionFile "C:\ADF\SSIS\myTrigger.json"

# Start your trigger
Start-AzureRmDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -Name "myTrigger"
```

# Scheduling via ADFv2 Sproc Activity

```
// JSON script to create a trigger for scheduling your pipeline runs
{
        "properties": {
                "name": “myTrigger",
                "type": "ScheduleTrigger",
                "typeProperties": {
                        "recurrence": {
                                "frequency": "Minute",
                                "interval": 15,
                                "startTime": "2017-12-22T04:30:00Z",
                                "endTime": "2017-12-22T05:00:00Z"
                        }
                },
                "pipelines": [{
                        "pipelineReference": {
                                "type": "PipelineReference",
                                "referenceName": "myPipeline"
                        },
                        "parameters": {}
                }]
        }
}
```

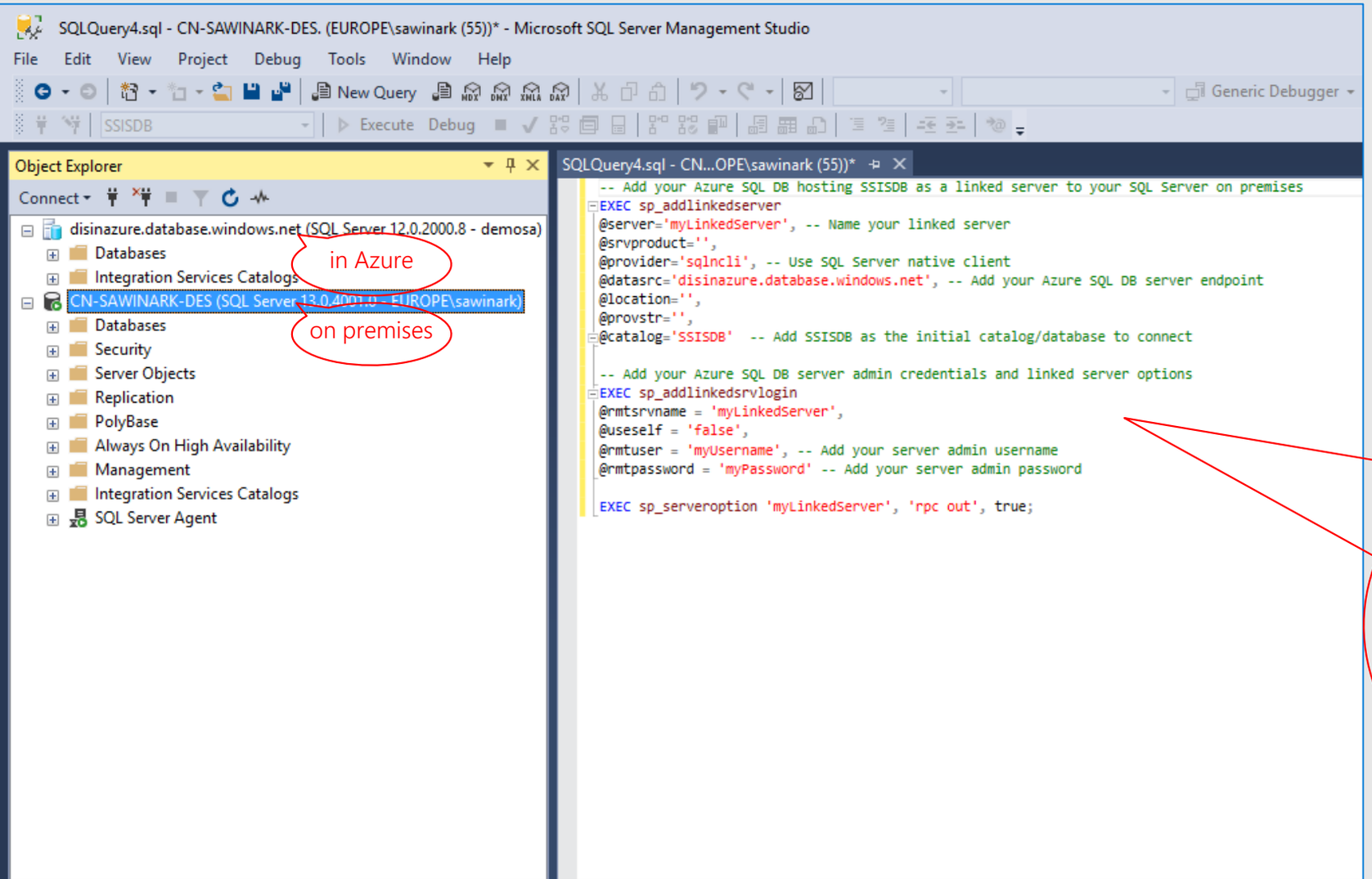
# Scheduling via Elastic Jobs

- Create Elastic Jobs DB
- Create Elastic Jobs account
- Create DB-scoped credentials
- Add Elastic Jobs target groups
- Add Elastic Jobs target group members
- Add jobs to schedule SSIS package executions
- Add job steps to create and start SSIS package executions
- Enable job schedules

# Scheduling via Elastic Jobs

```
-- T-SQL script to set up Elastic Jobs for scheduling SSIS package execution
-- Create Elastic Jobs target group
EXEC jobs.sp_add_target_group 'TargetGroup'

-- Add Elastic Jobs target group member
EXEC jobs.sp_add_target_group_member @target_group_name='TargetGroup', @target_type='SqlDatabase',
        @server_name='YourAzureSQLDBServer.database.windows.net',@database_name='SSISDB'

-- Add a job to schedule SSIS package execution
EXEC jobs.sp_add_job @job_name='ExecutePackageJob', @description='Description', @schedule_interval_type='Minutes',
        @schedule_interval_count=60

-- Add a job step to create/start SSIS package execution using SSISDB sprocs
EXEC jobs.sp_add_jobstep @job_name='ExecutePackageJob', @command=N'DECLARE @exe_id bigint
        EXEC [SSISDB].[catalog].[create_execution] @folder_name=N''YourFolder'', @project_name=N''YourProject'',
                @package_name=N''YourPackage'', @use32bitruntime=0, @runincluster=1, @useanyworker=1,
        @execution_id=@exe_id OUTPUT
        EXEC [SSISDB].[catalog].[start_execution] @exe_id, @retry_count=0', @credential_name='YourDBScopedCredentials',
                @target_group_name='TargetGroup'

-- Enable the job schedule
EXEC jobs.sp_update_job @job_name='ExecutePackageJob', @enabled=1, @schedule_interval_type='Minutes',
        @schedule_interval_count=60
```

# Scheduling via On-Prem SQL Server Agent

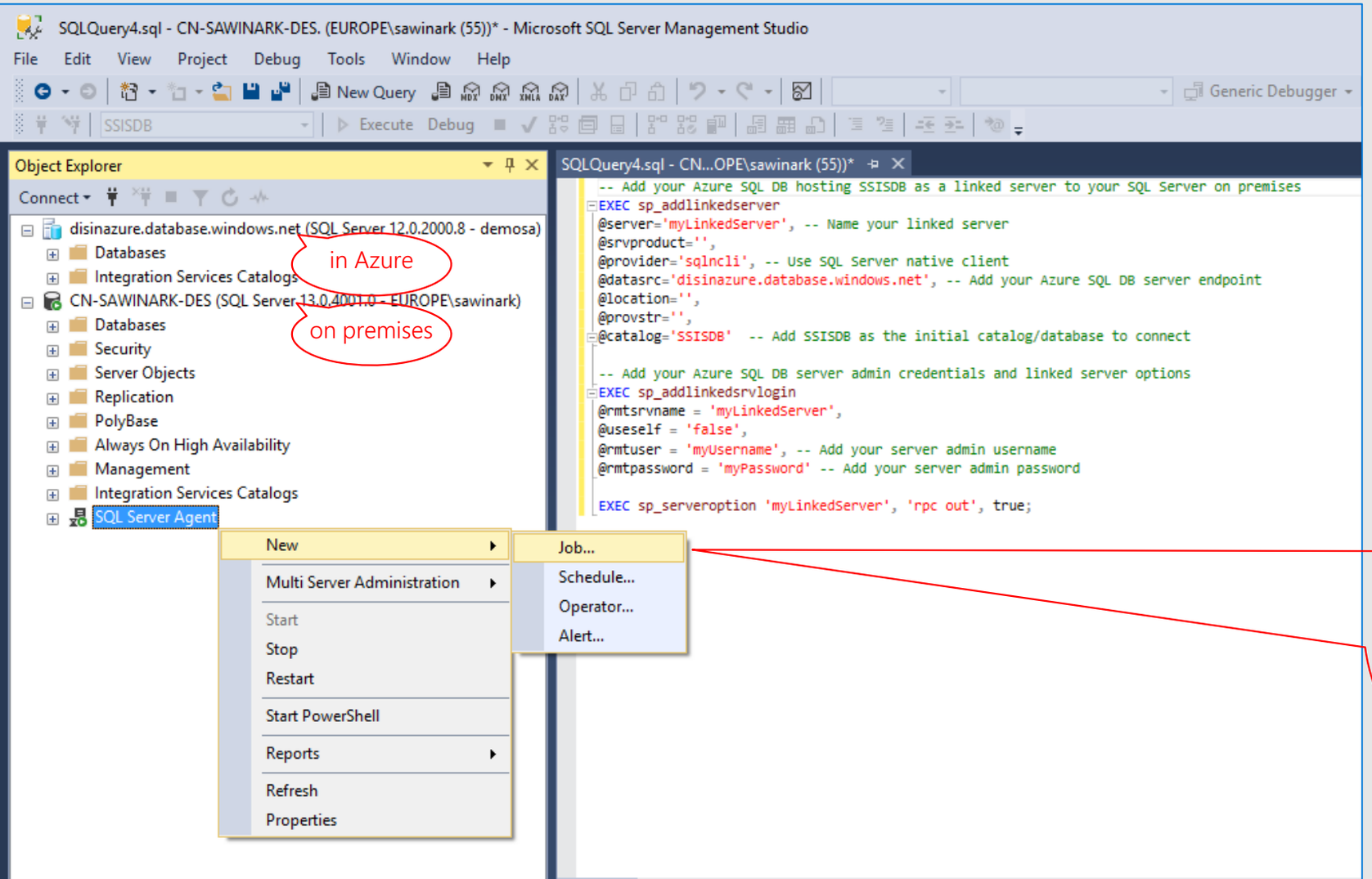



Add your Azure SQL DB hosting SSISDB as a linked server to your SQL Server on premises

# Scheduling via On-Prem SQL Server Agent

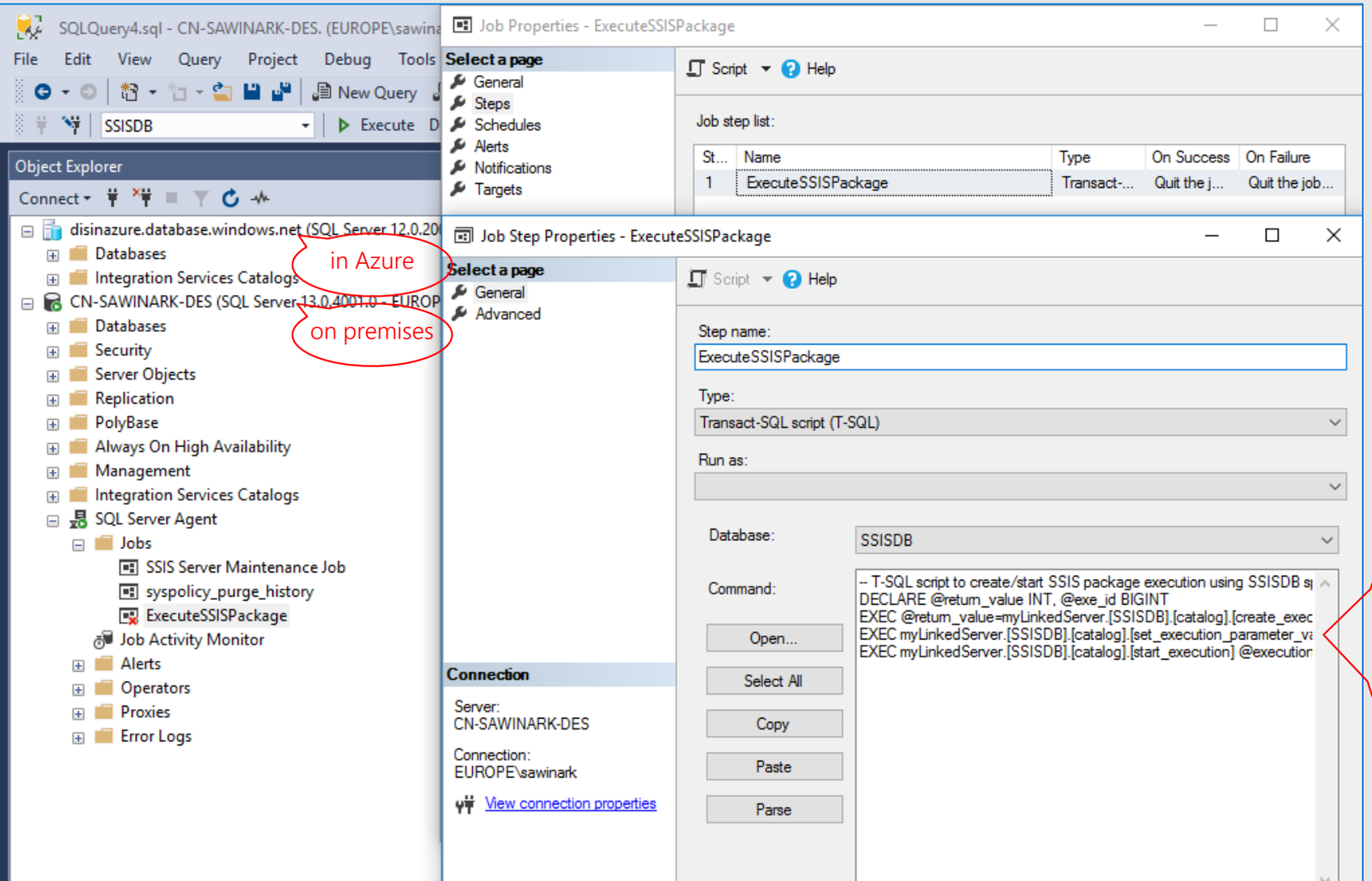
```
-- Add your Azure SQL DB hosting SSISDB as a linked server to your SQL Server on premises
EXEC sp_addlinkedserver
        @server='myLinkedServer', -- Name your linked server
        @srvproduct='',
        @provider='sqlncli', -- Use SQL Server native client
        @datasrc='disinazure.database.windows.net', -- Add your Azure SQL DB server endpoint
        @location='',
        @provstr='',
        @catalog='SSISDB'  -- Add SSISDB as the initial catalog/database to connect

-- Add your Azure SQL DB server admin credentials and linked server options
EXEC sp_addlinkedsrvlogin
        @rmtsrvname = 'myLinkedServer',
        @useself = 'false',
        @rmtuser = 'myUsername', -- Add your server admin username
        @rmtpassword = 'myPassword' -- Add your server admin password

EXEC sp_serveroption 'myLinkedServer', 'rpc out', true;
```

# Scheduling via On-Prem SQL Server Agent




Add T-SQL jobs to schedule SSIS package executions on SSIS PaaS via on-prem SQL Server Agent

# Scheduling via On-Prem SQL Server Agent

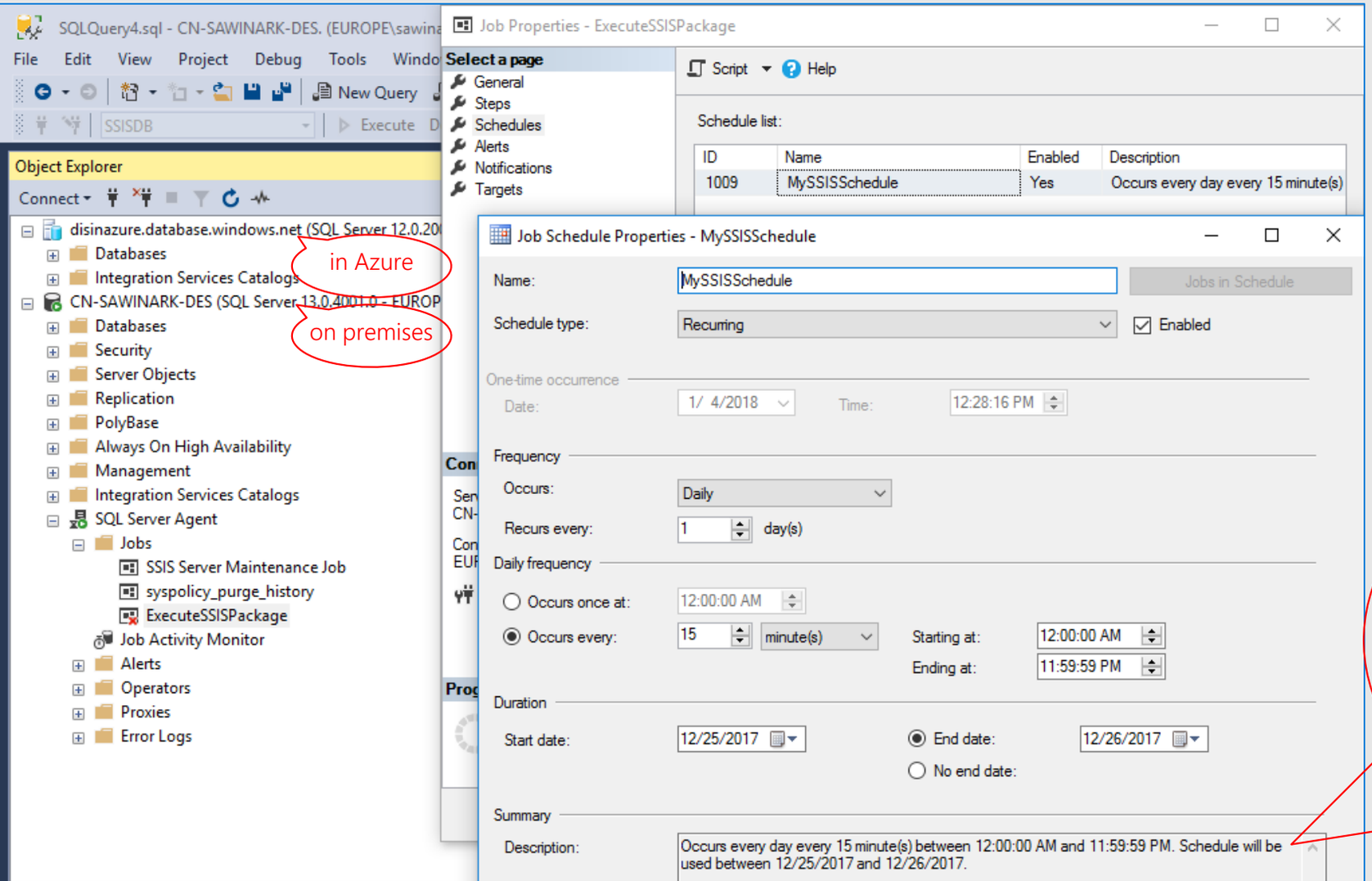



T-SQL jobs can execute SSISDB sprocs in Azure SQL DB that has been added as a linked server to SQL Server on premises

# Scheduling via On-Prem SQL Server Agent

```
-- T-SQL script to create/start SSIS package execution using SSISDB sprocs
DECLARE @return_value int, @exe_id bigint

EXEC @return_value = [YourLinkedServer].[SSISDB].[catalog].[create_execution]
        @folder_name=N'folderName', @project_name=N'projectName',
        @package_name=N'packageName', @use32bitruntime=0, @runincluster=1, @useanyworker=1,
        @execution_id=@exe_id OUTPUT

EXEC [YourLinkedServer].[SSISDB].[catalog].[set_execution_parameter_value] @exe_id,
        @object_type=50, @parameter_name=N'SYNCHRONIZED', @parameter_value=1

EXEC [YourLinkedServer].[SSISDB].[catalog].[start_execution] @execution_id=@exe_id
```

# Scheduling via On-Prem SQL Server Agent



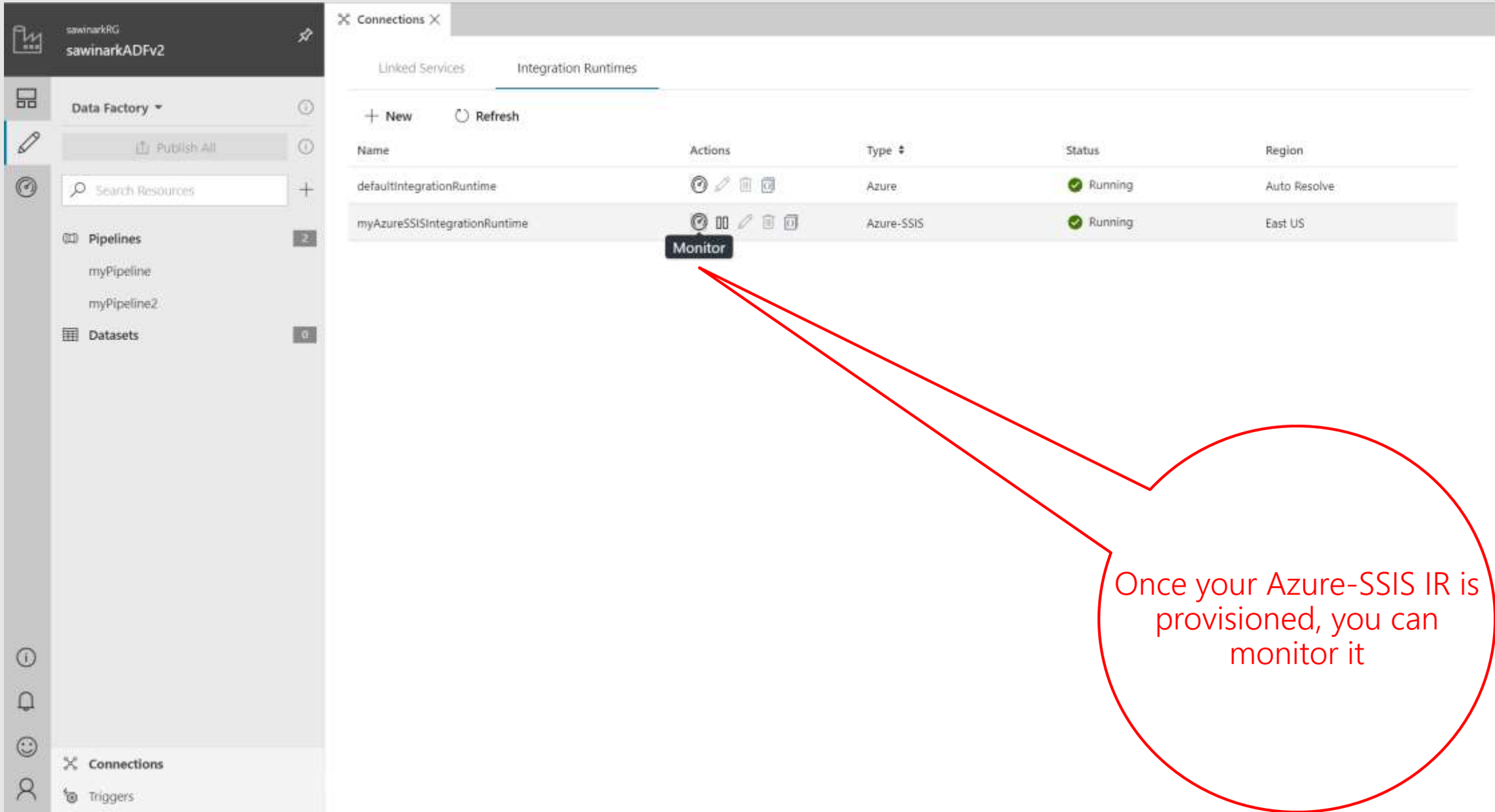

Schedule T-SQL jobs to execute SSISDB sprocs in Azure SQL DB server, triggering SSIS package executions on Azure-SSIS IR

# Monitoring Methods

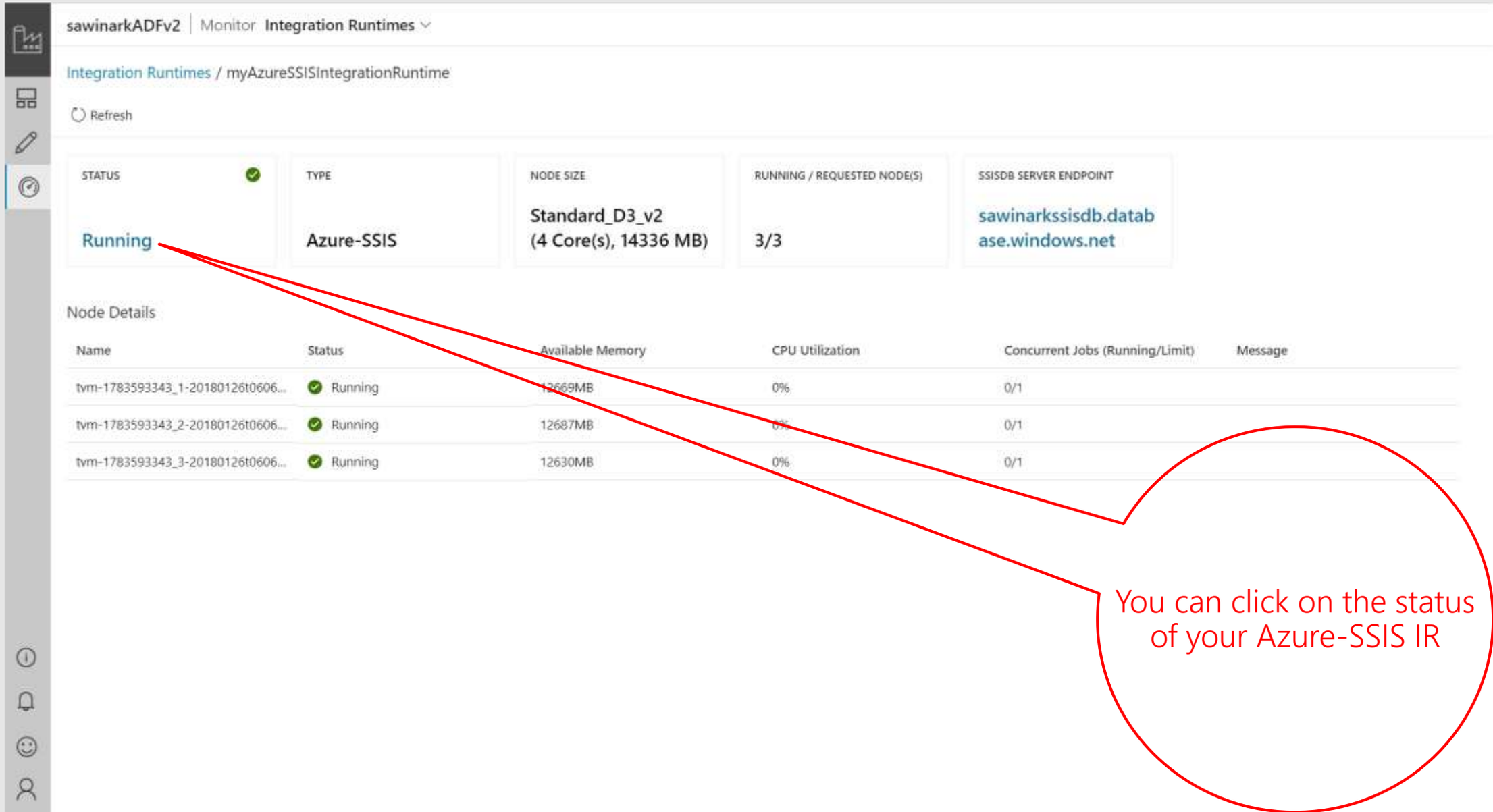
# Monitoring Methods

- Azure-SSIS IR can be monitored via ADFv2 App
- Azure-SSIS IR can be monitored via PSH/custom code using ADFv2 .NET SDK/API
- SSIS package executions can be directly monitored as first-class SSIS activities in ADFv2 pipelines (Work in Progress)
  - For now, SSIS package executions can be indirectly monitored via ADFv2 Sproc Activity
- SSIS package executions can be monitored via SSMS

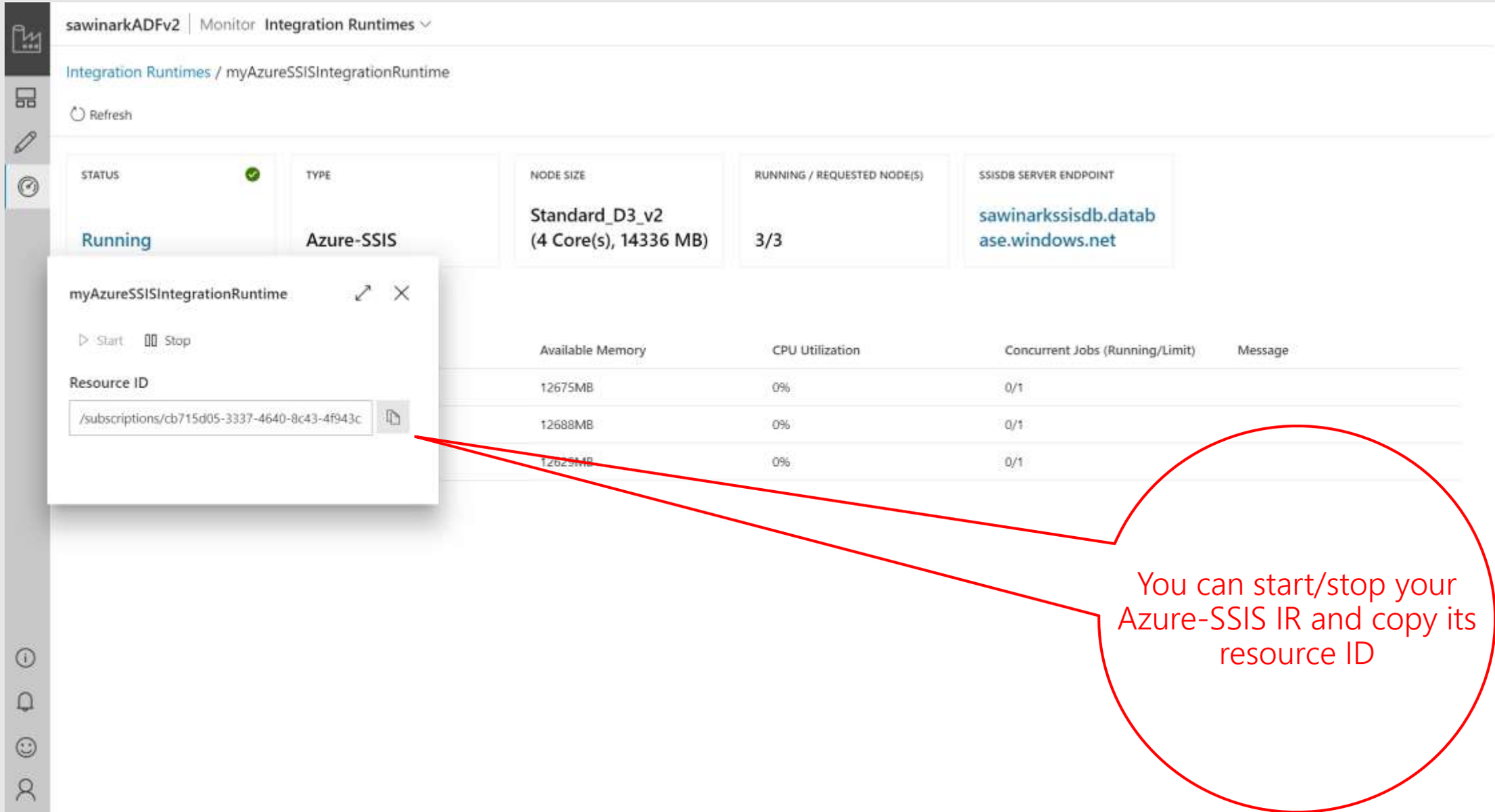
# Monitoring via ADFv2 App



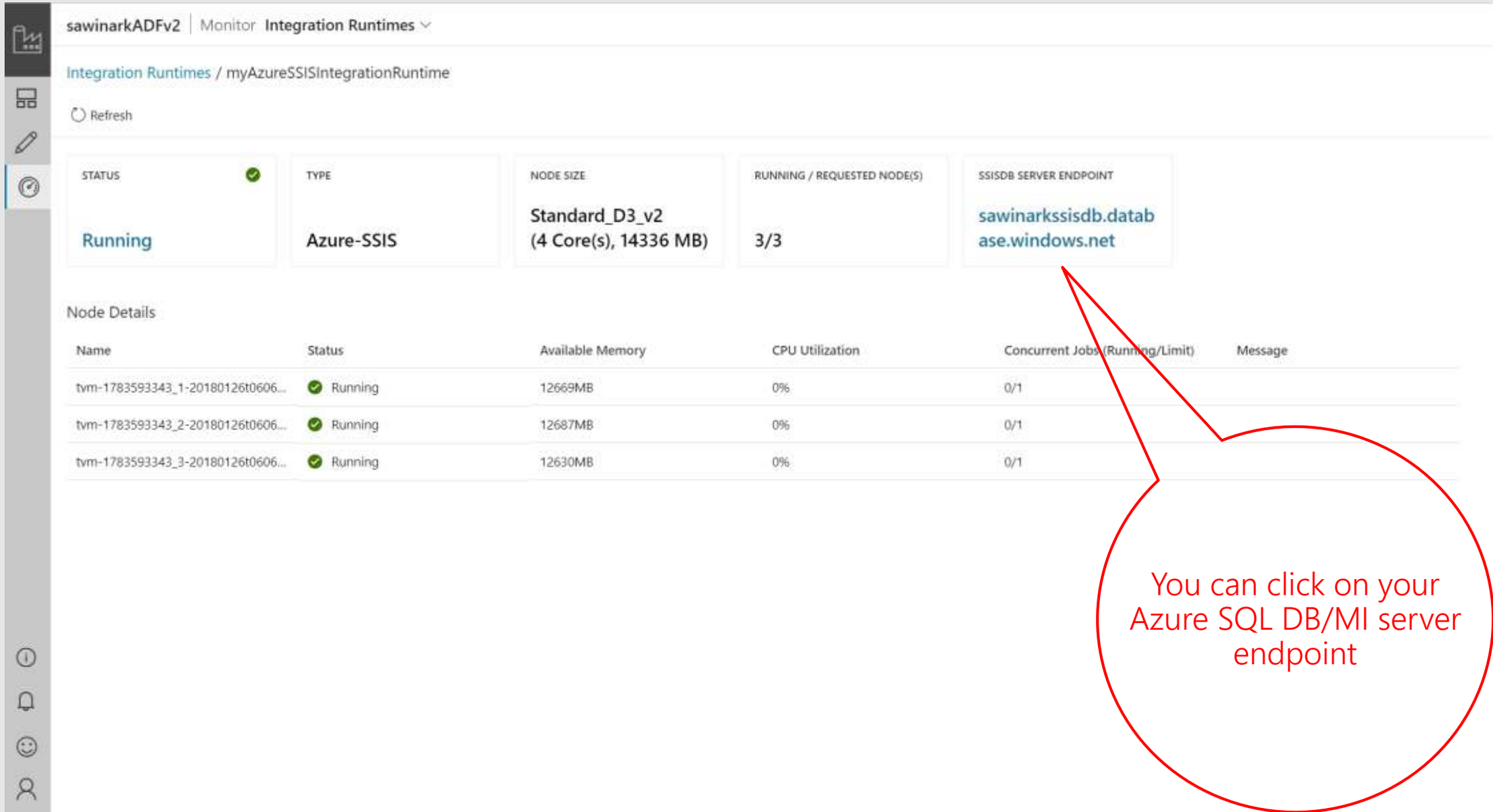
# Monitoring via ADFv2 App



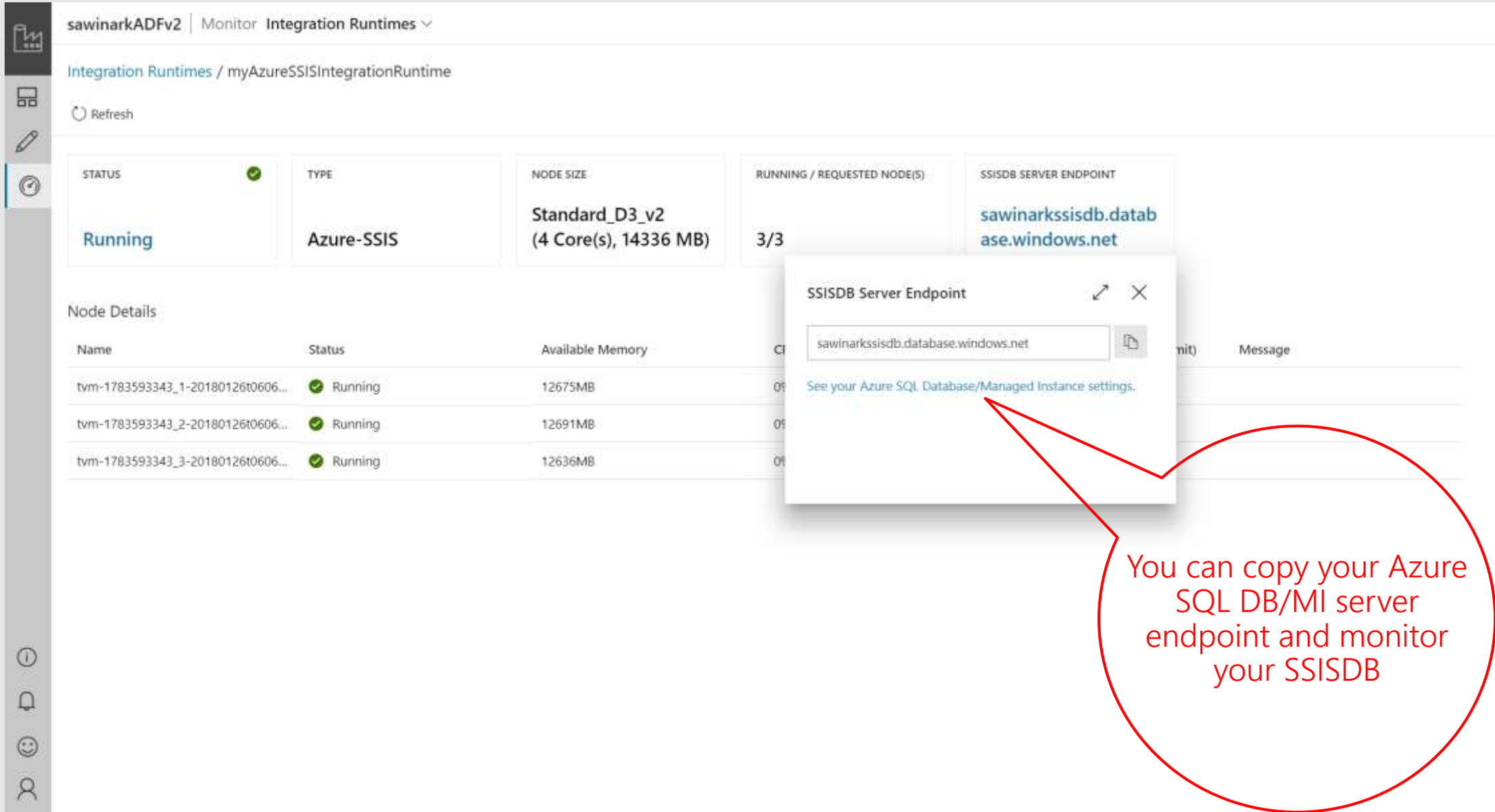
# Monitoring via ADFv2 App



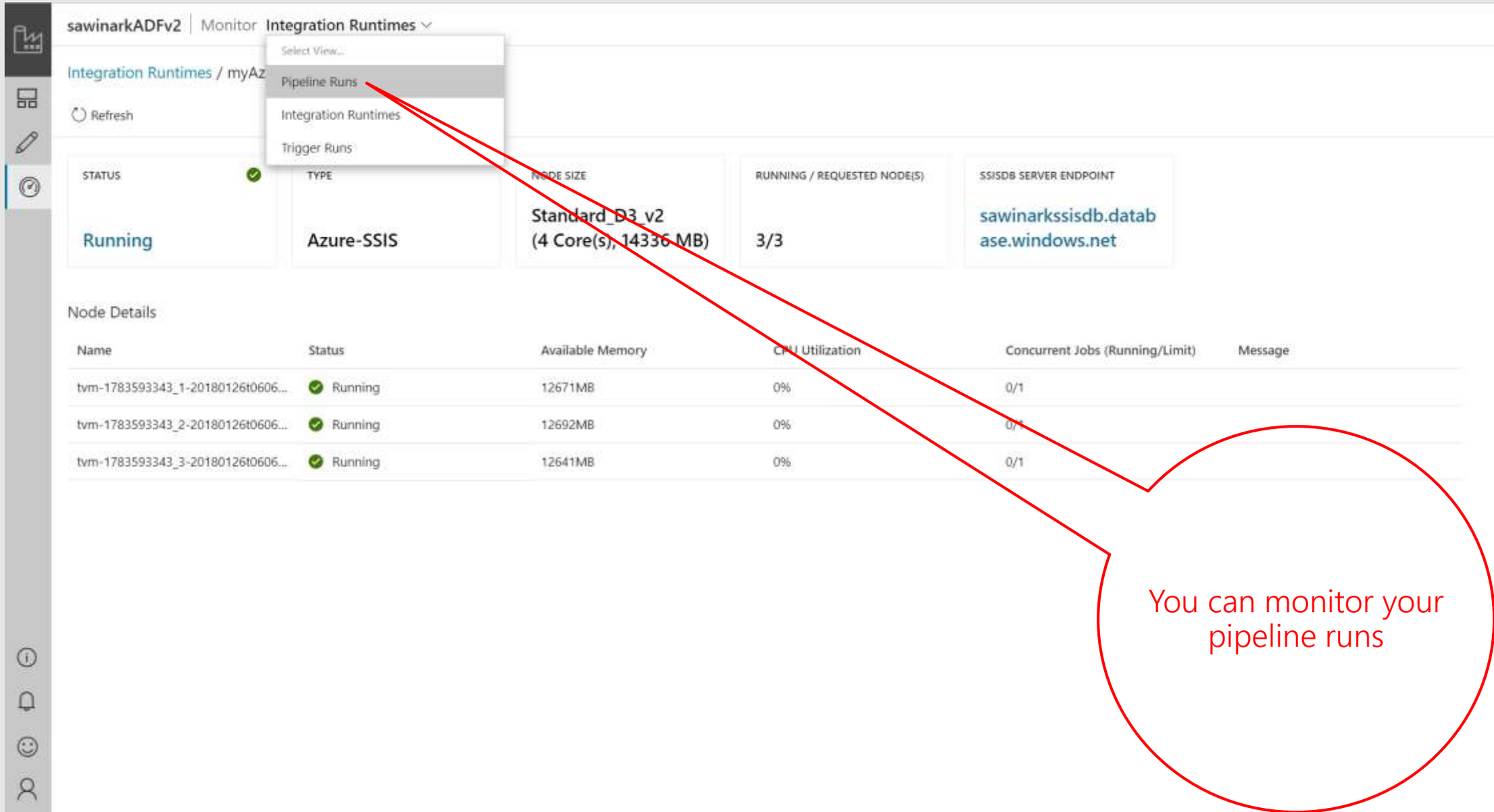
# Monitoring via ADFv2 App



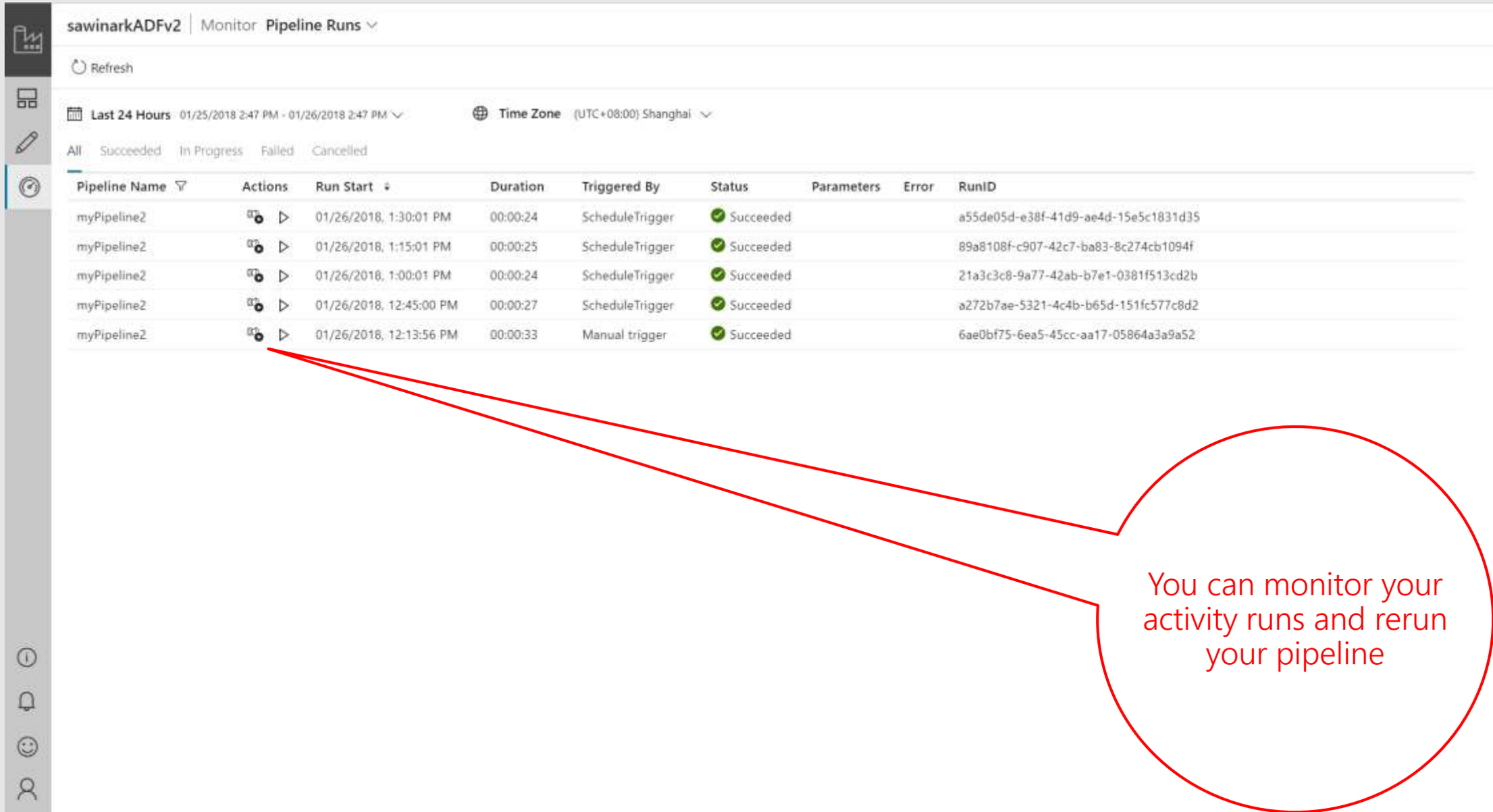
# Monitoring via ADFv2 App



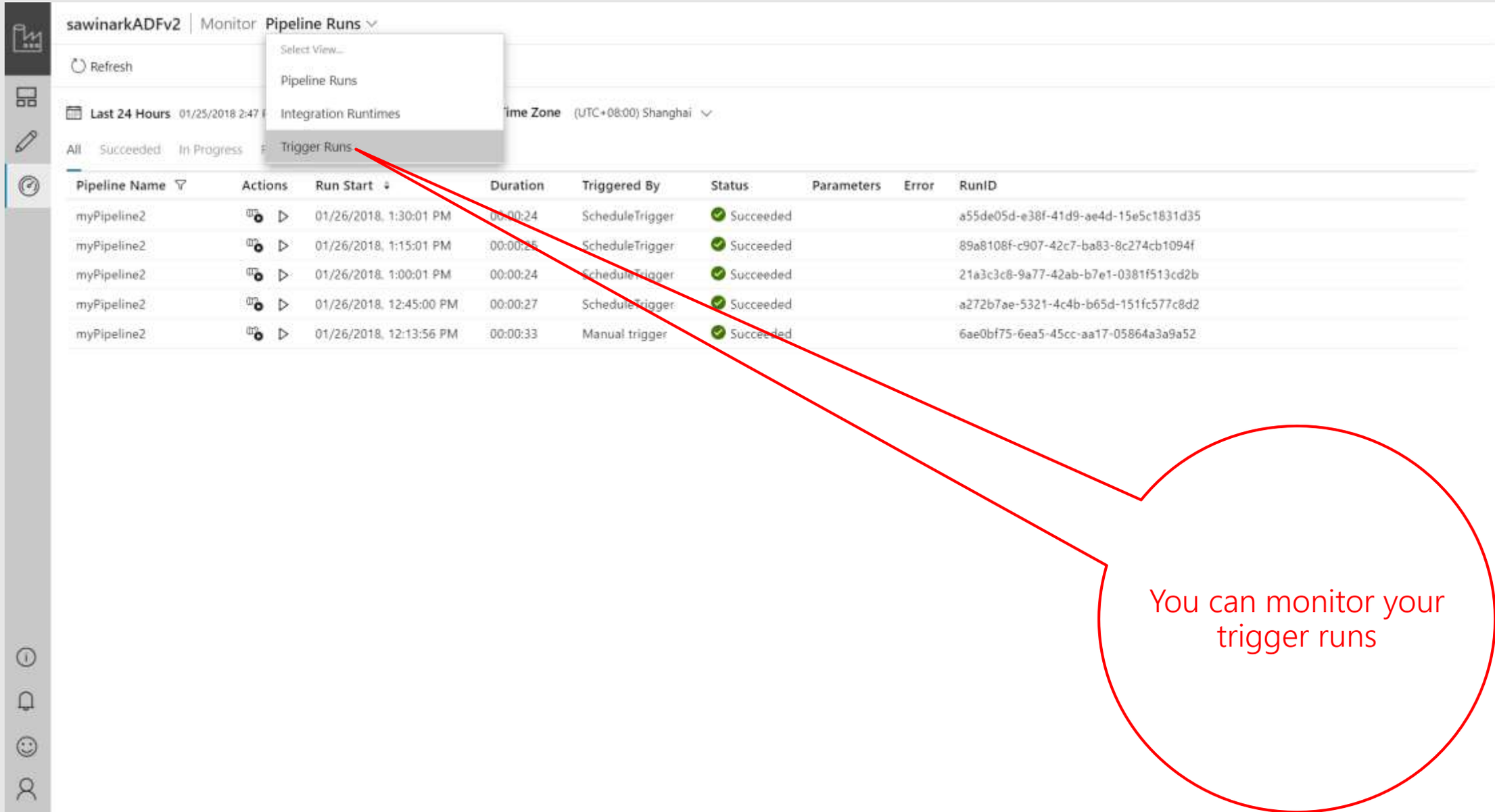
# Monitoring via ADFv2 App



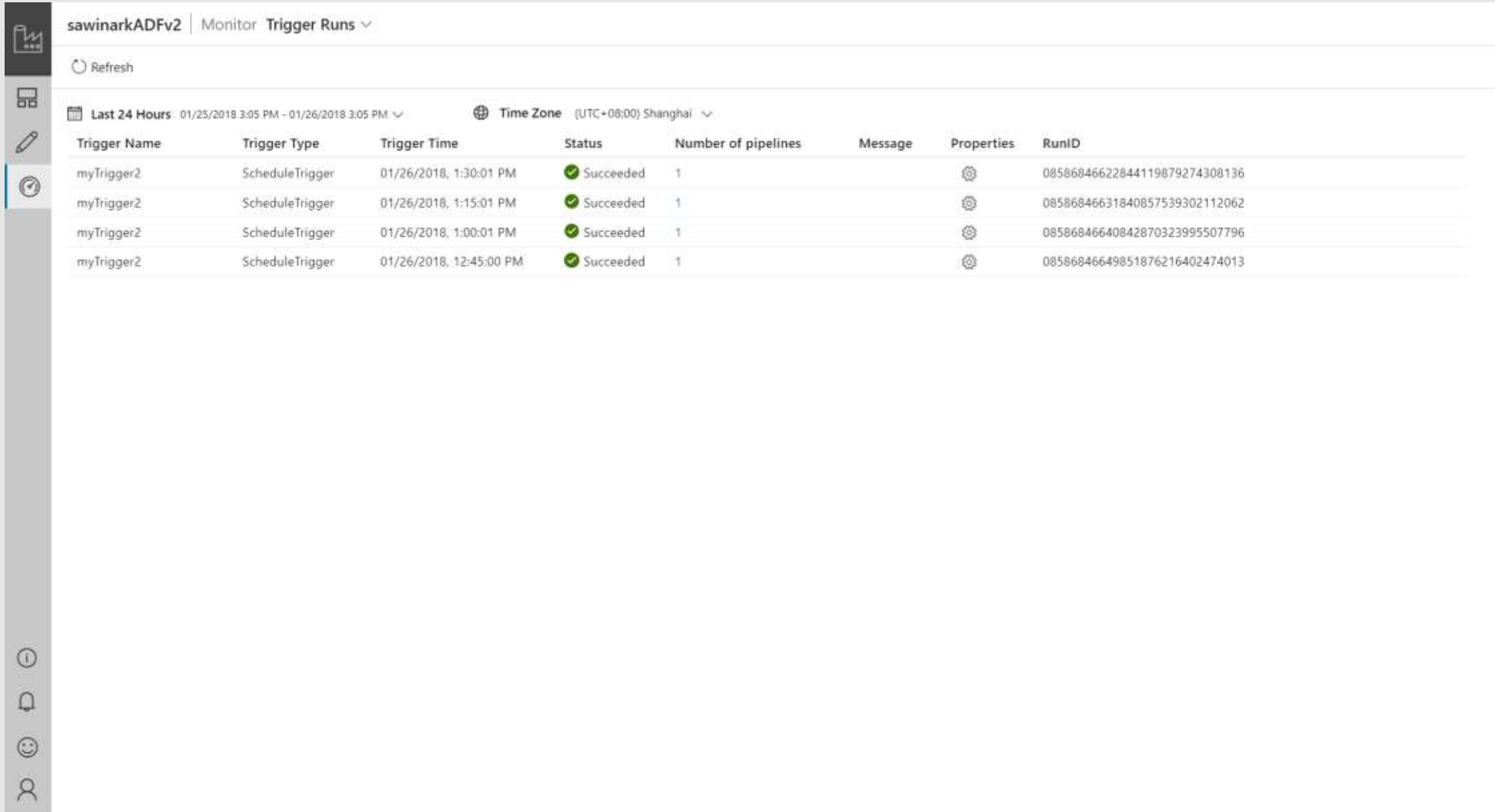
# Monitoring via ADFv2 App



# Monitoring via ADFv2 App



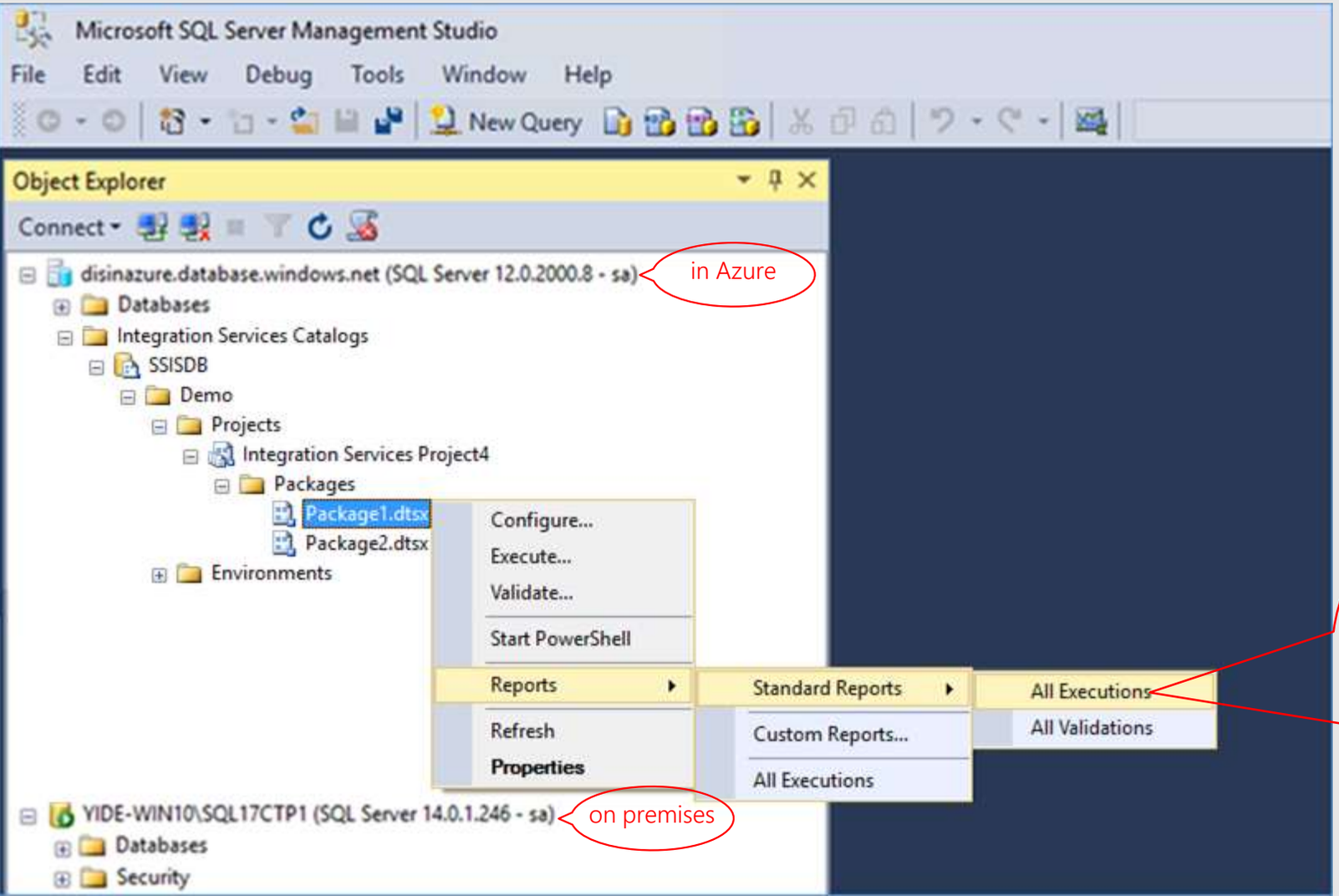
# Monitoring via ADFv2 App



# Monitoring via PSH

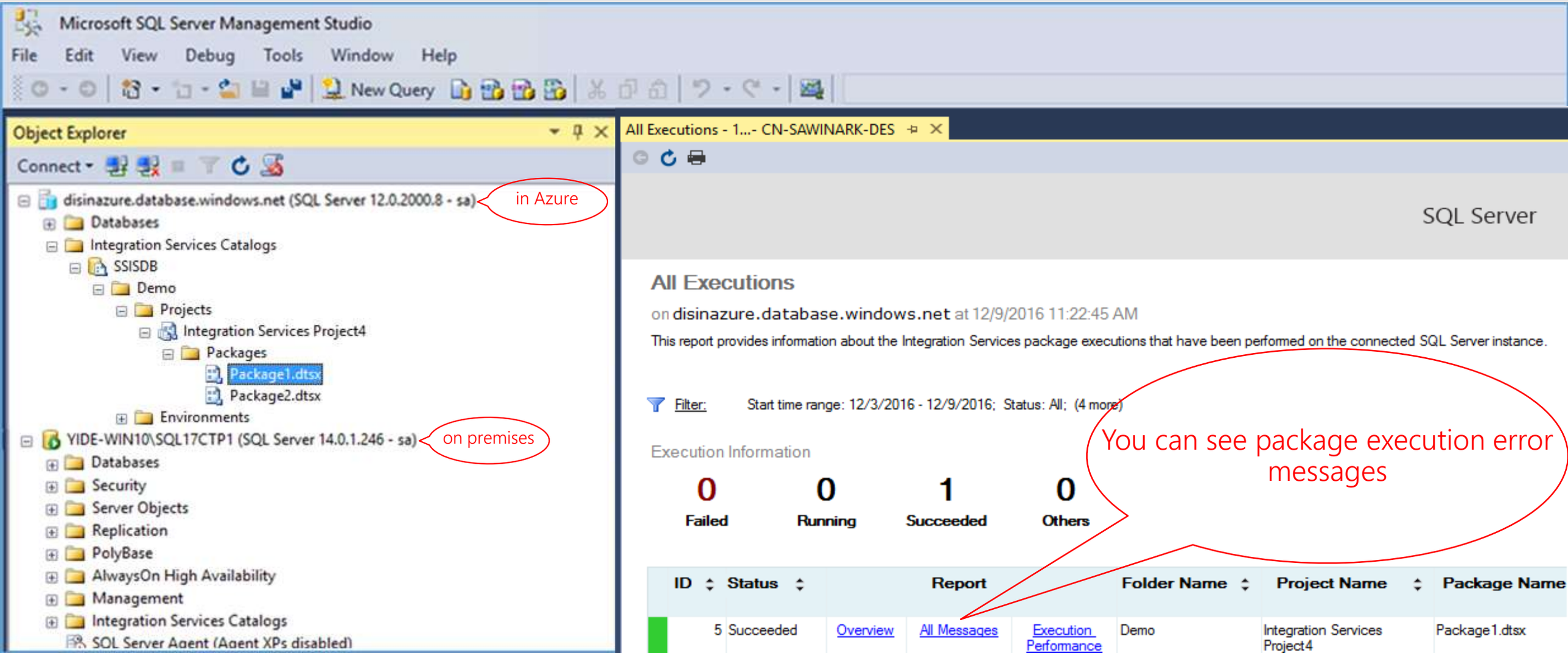
```
##### PSH script to monitor Azure-SSIS Integration Runtime and SSIS package executions in ADFv2 #####

# Query/monitor your Azure SSIS Integration Runtime
Get-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $DataFactoryName -Name $AzureSSISName -ResourceGroupName $ResourceGroupName -Status

# Query/monitor your pipeline run
Get-AzureRmDataFactoryV2PipelineRun -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -PipelineRunId $myPipelineRun

# Query/monitor your trigger
Get-AzureRmDataFactoryV2Trigger -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -Name "myTrigger"

# Query/monitor your trigger runs
Get-AzureRmDataFactoryV2TriggerRun -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -TriggerName "myTrigger" -TriggerRunStartedAfter "2017-12-22" -TriggerRunStartedBefore "2017-12-23"
```

# Monitoring via SSMS




You can see reports of all package executions on SSIS PaaS

# Monitoring via SSMS

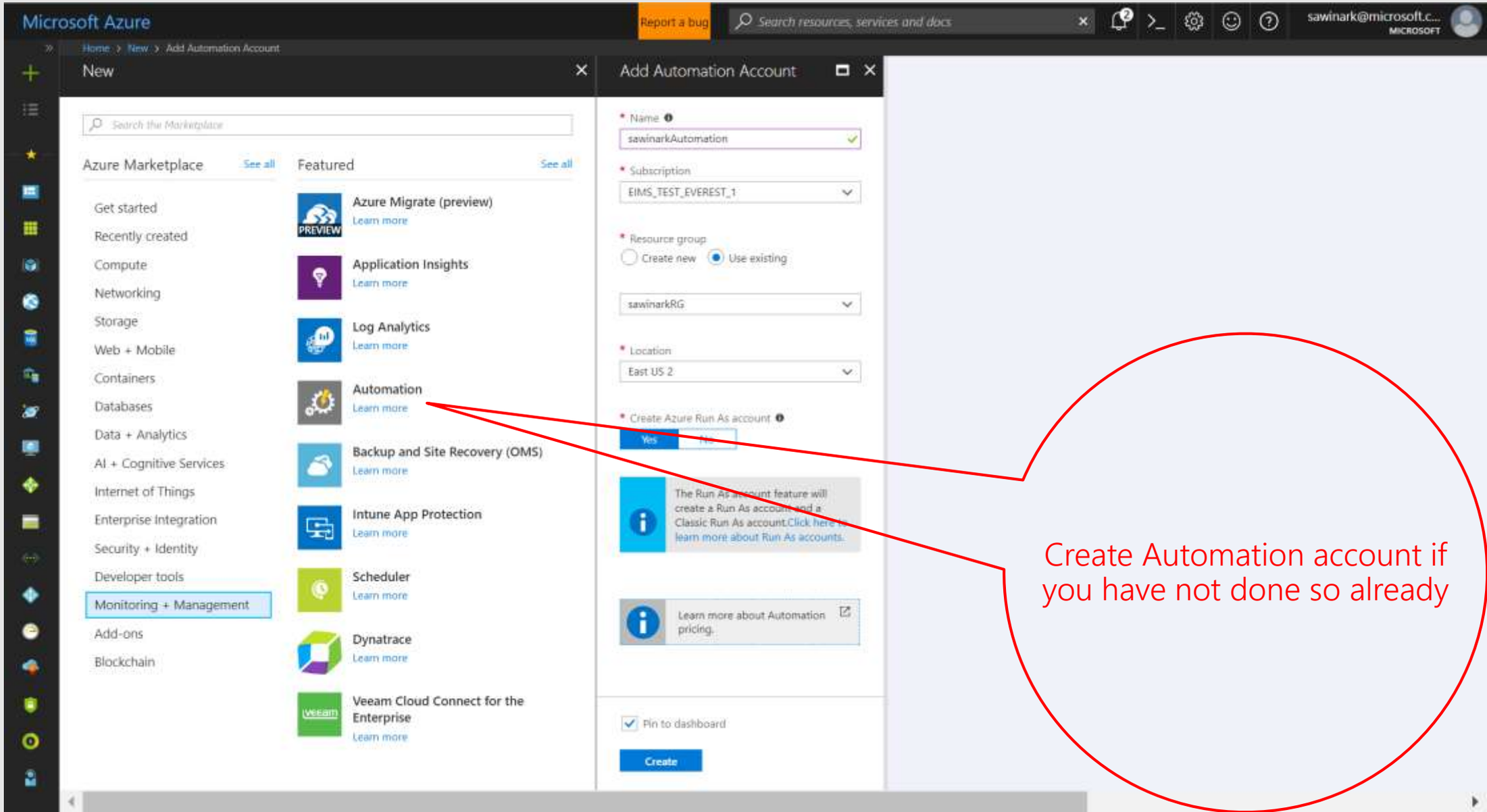


# On-Demand/Just-In-Time Provisioning

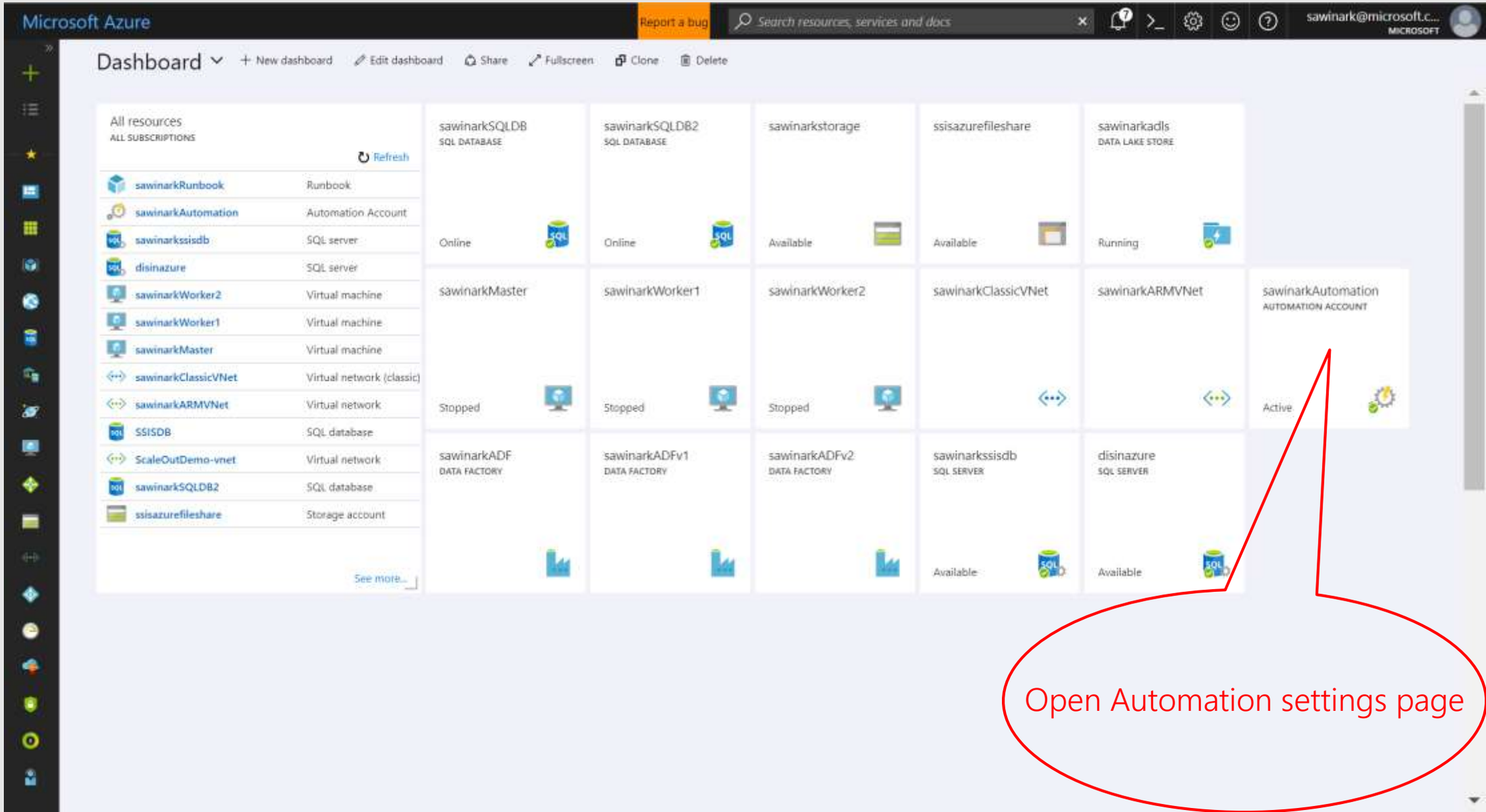
# On-Demand/Just-In-Time Provisioning

- Starting/stopping Azure-SSIS IR can be invoked/scheduled via Azure Automation
- Starting/stopping Azure-SSIS IR can be invoked/scheduled using ADFv2 Web Activity that triggers Azure Automation via webhooks
- Starting/stopping Azure-SSIS IR can be done on demand/just in time before/after executing packages by chaining ADFv2 Web and Sproc Activities

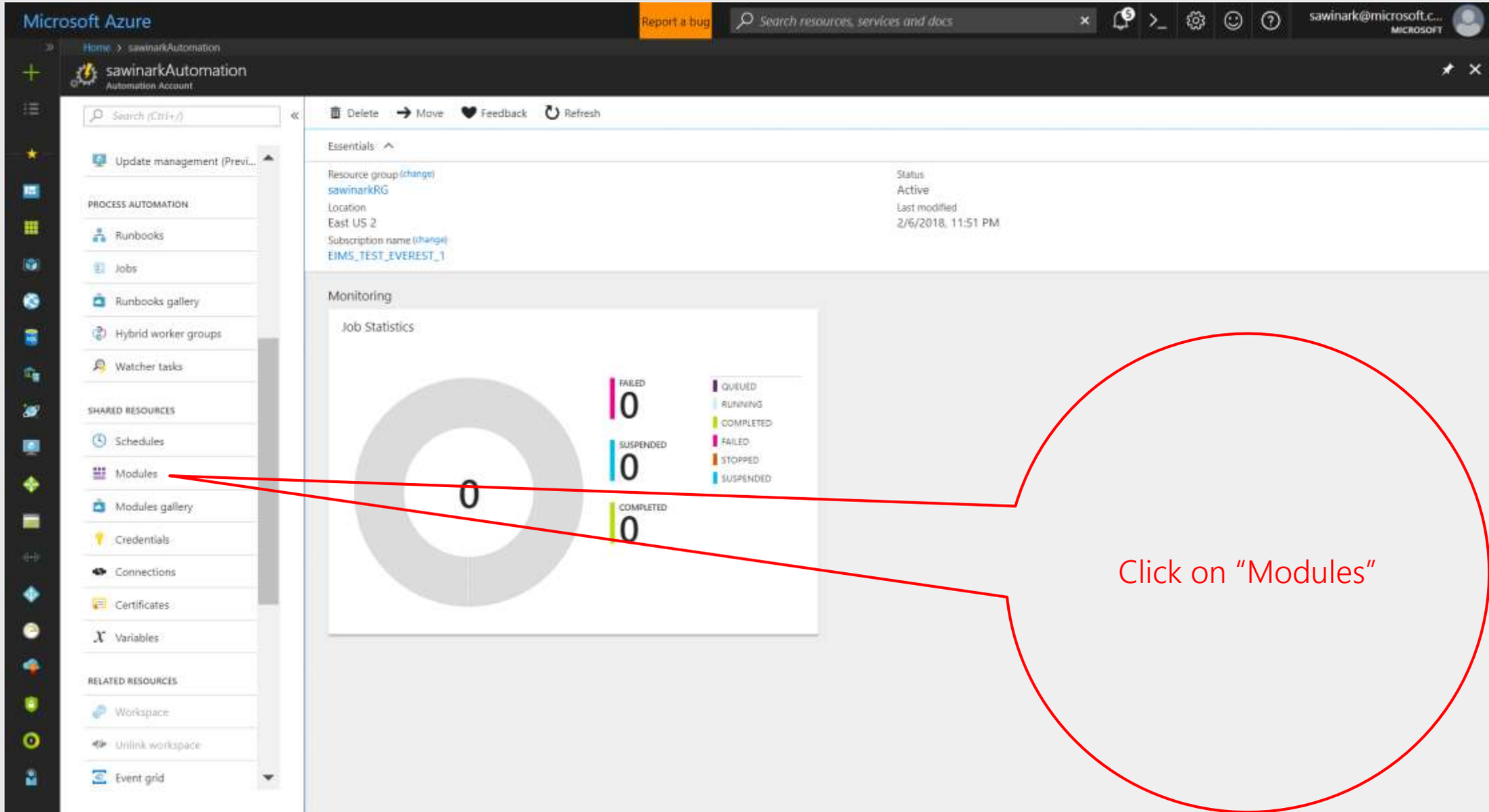
# On-Demand/Just-In-Time Provisioning



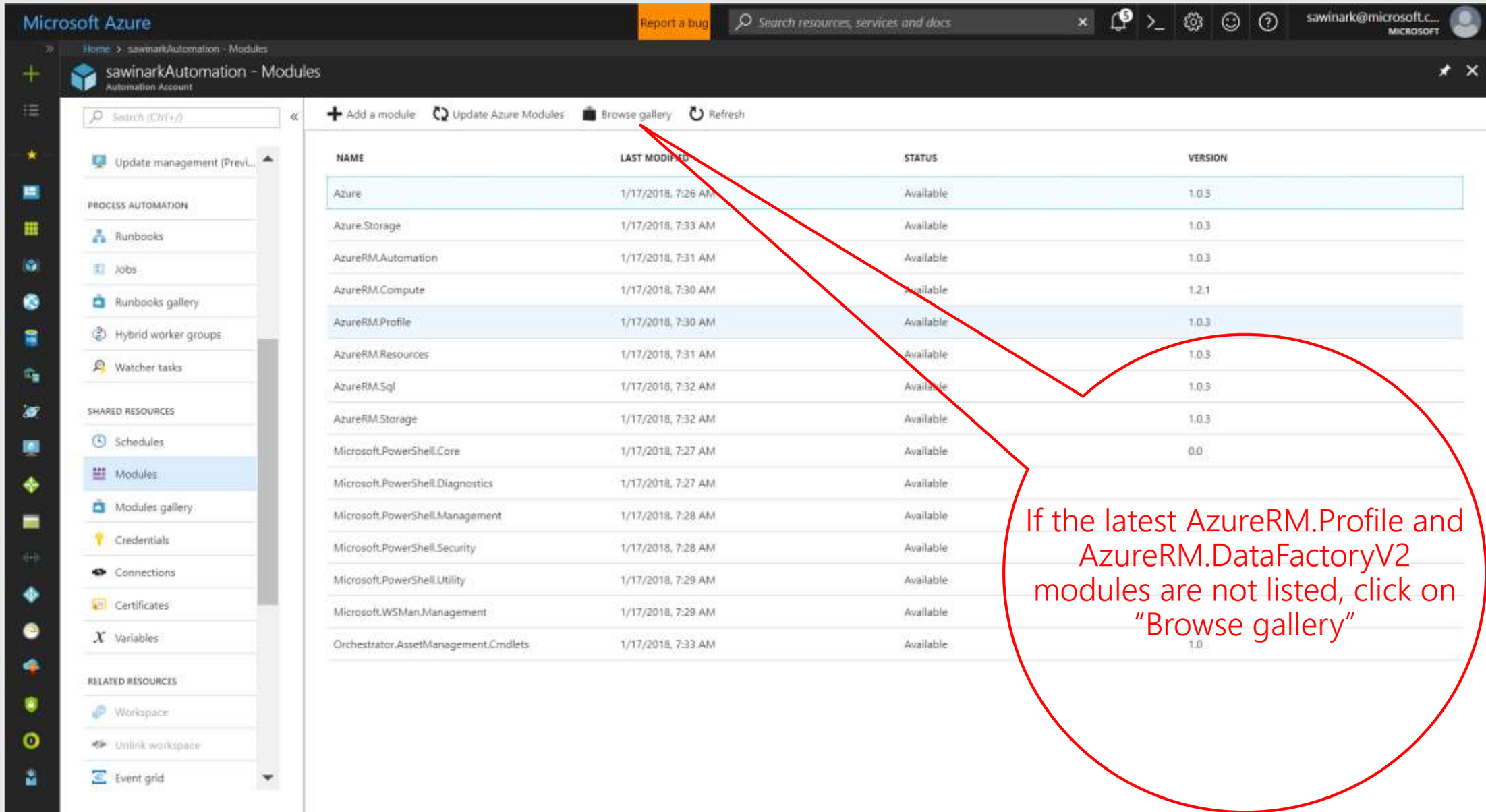
# On-Demand/Just-In-Time Provisioning



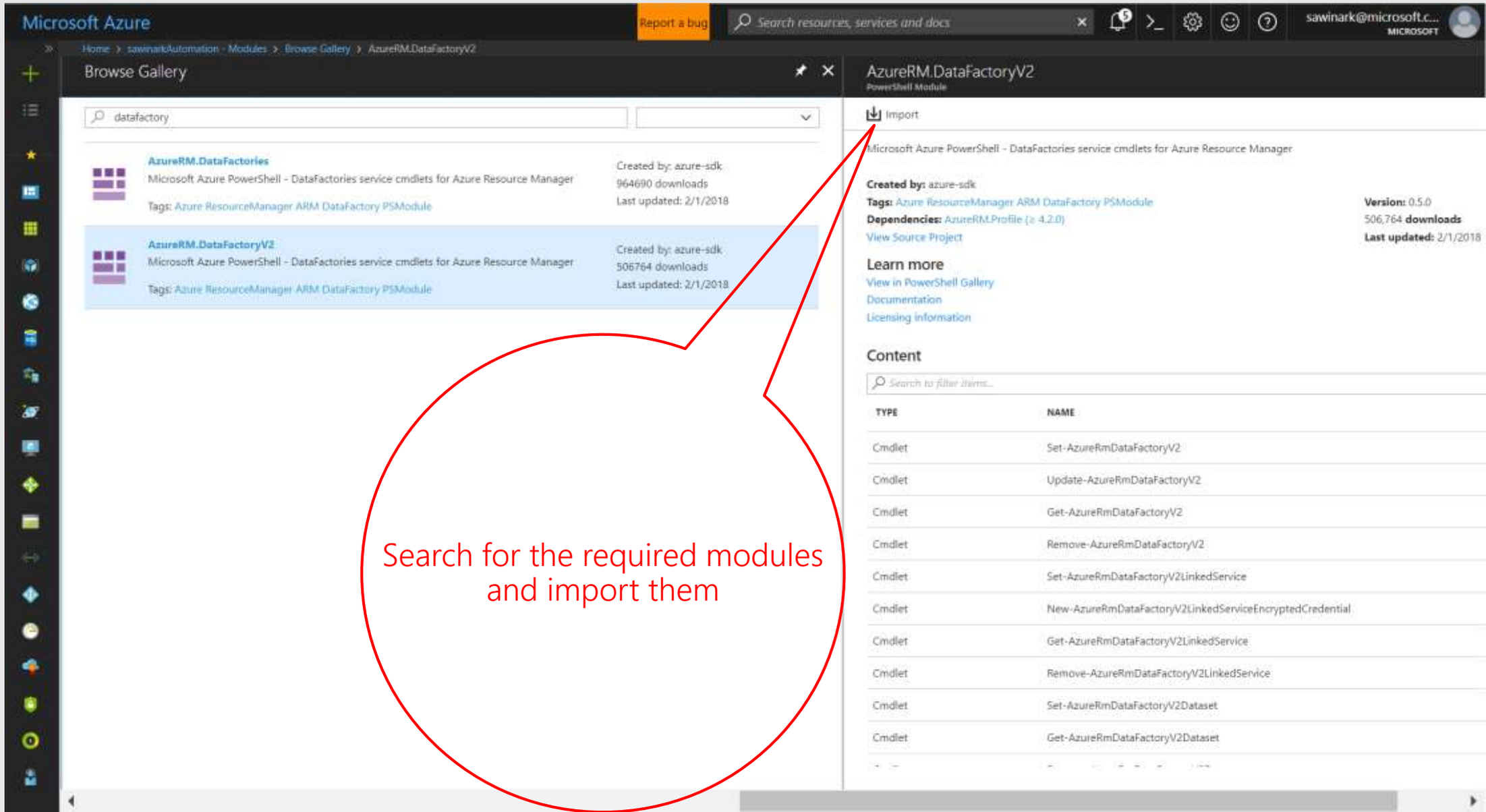
# On-Demand/Just-In-Time Provisioning



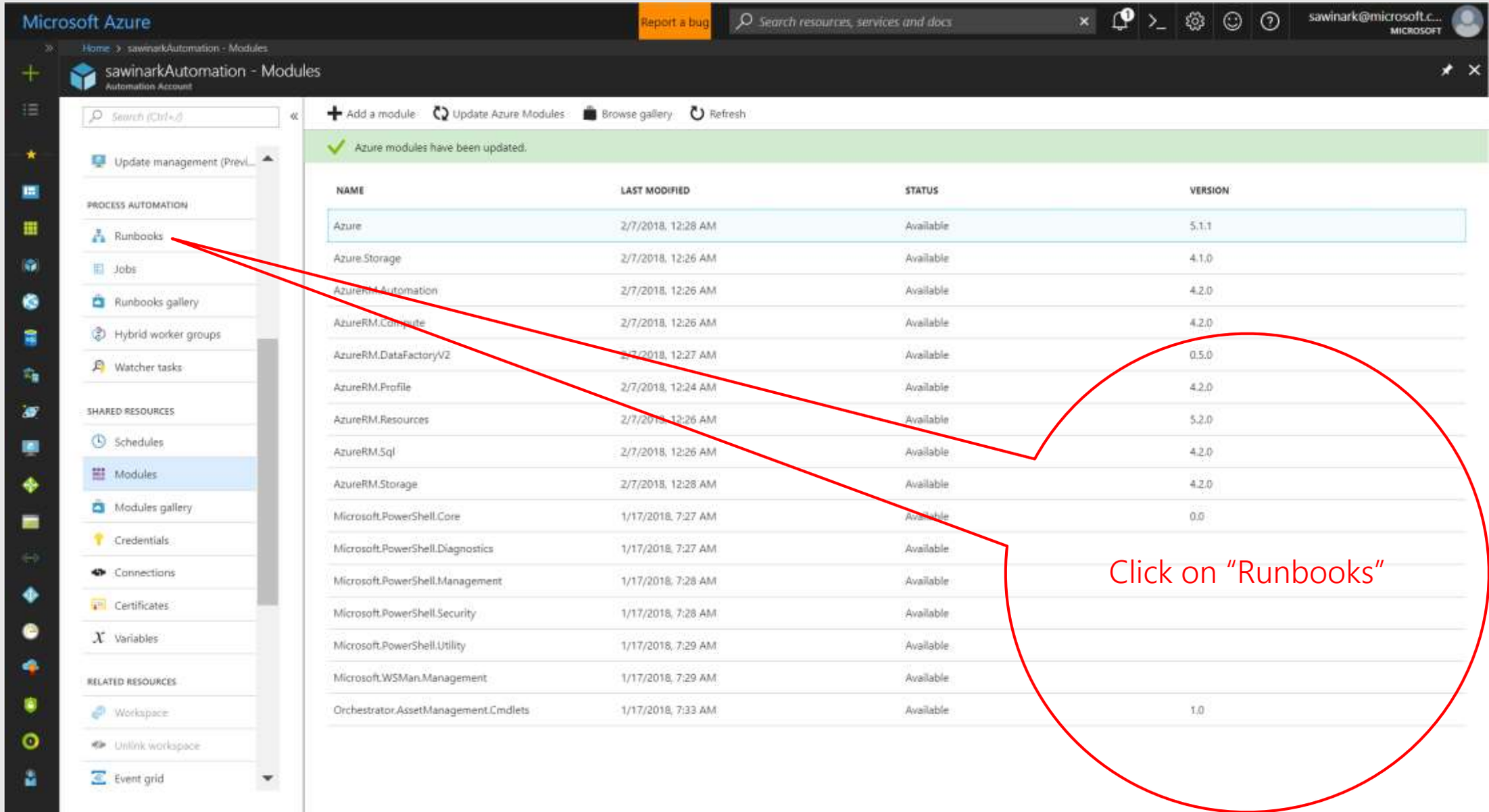
# On-Demand/Just-In-Time Provisioning



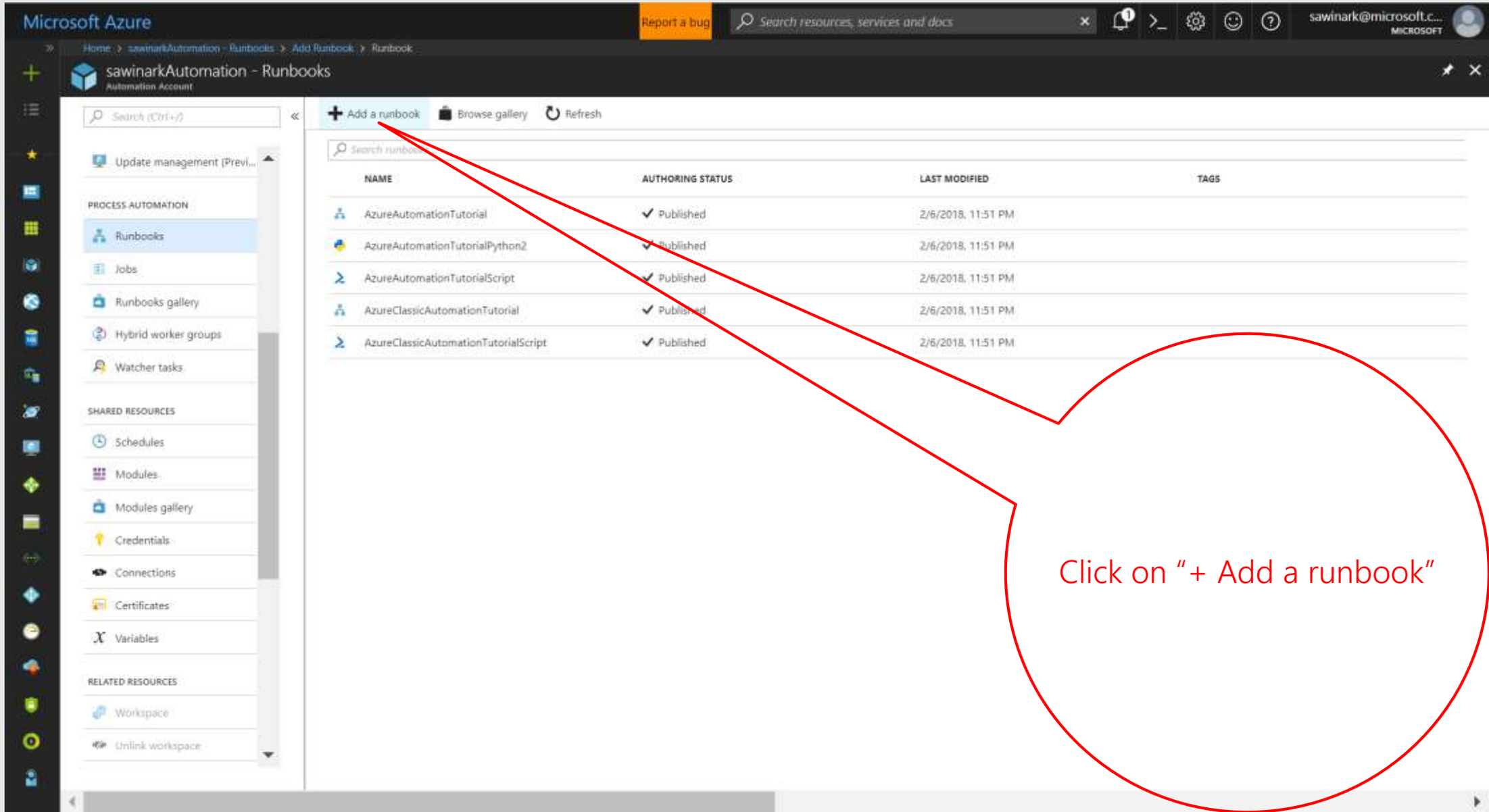
# On-Demand/Just-In-Time Provisioning



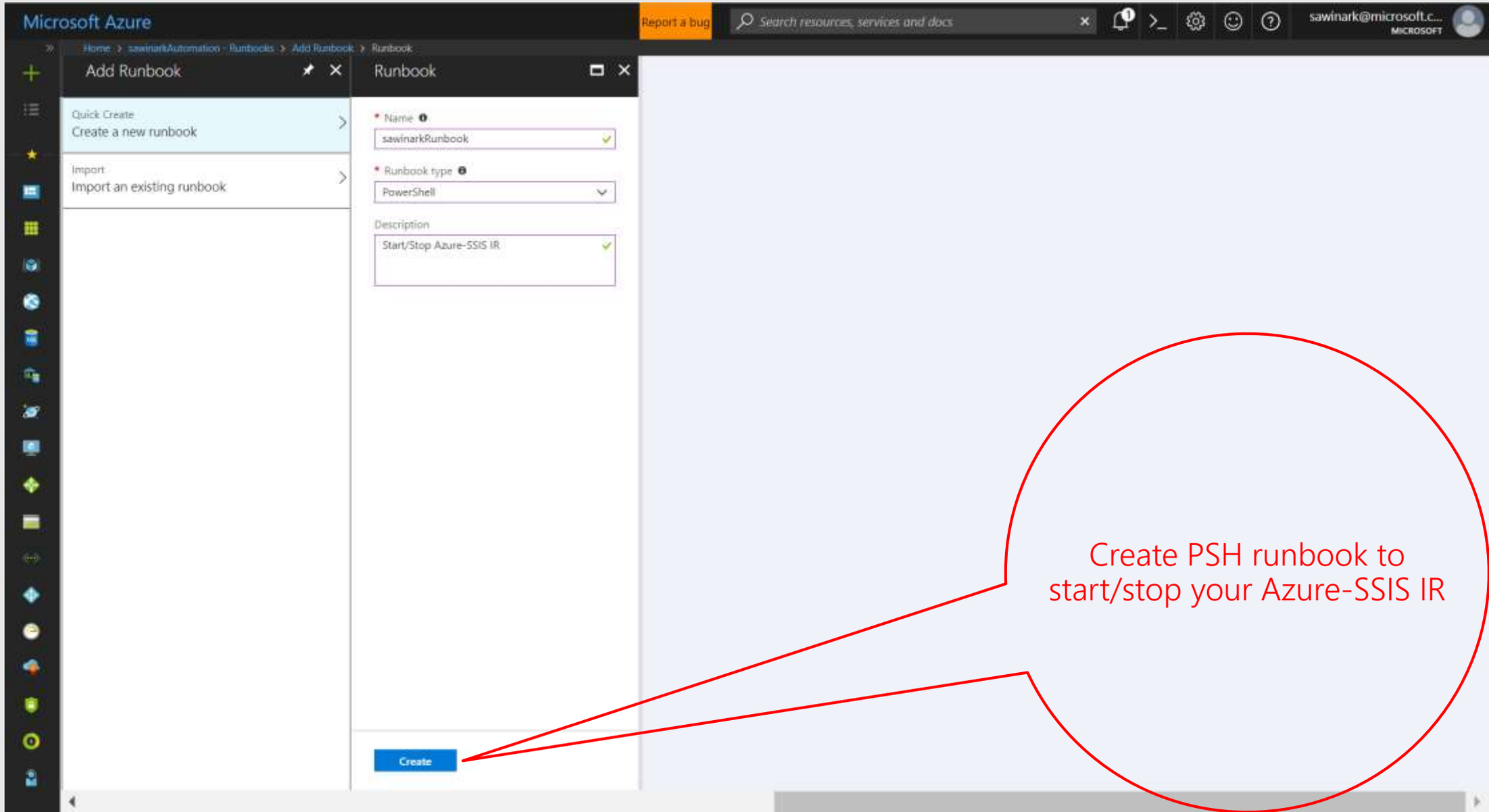
# On-Demand/Just-In-Time Provisioning



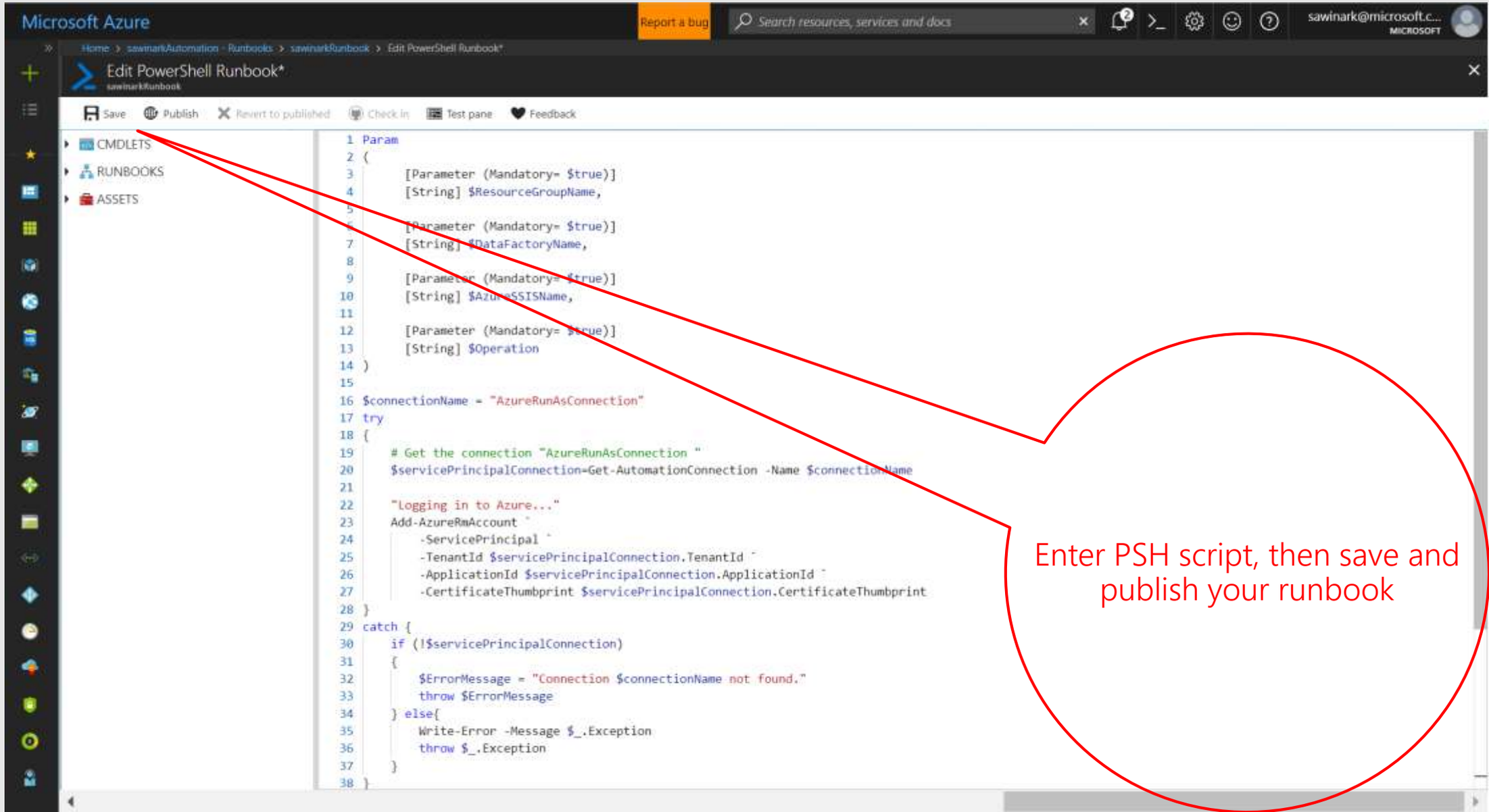
# On-Demand/Just-In-Time Provisioning



# On-Demand/Just-In-Time Provisioning



# On-Demand/Just-In-Time Provisioning



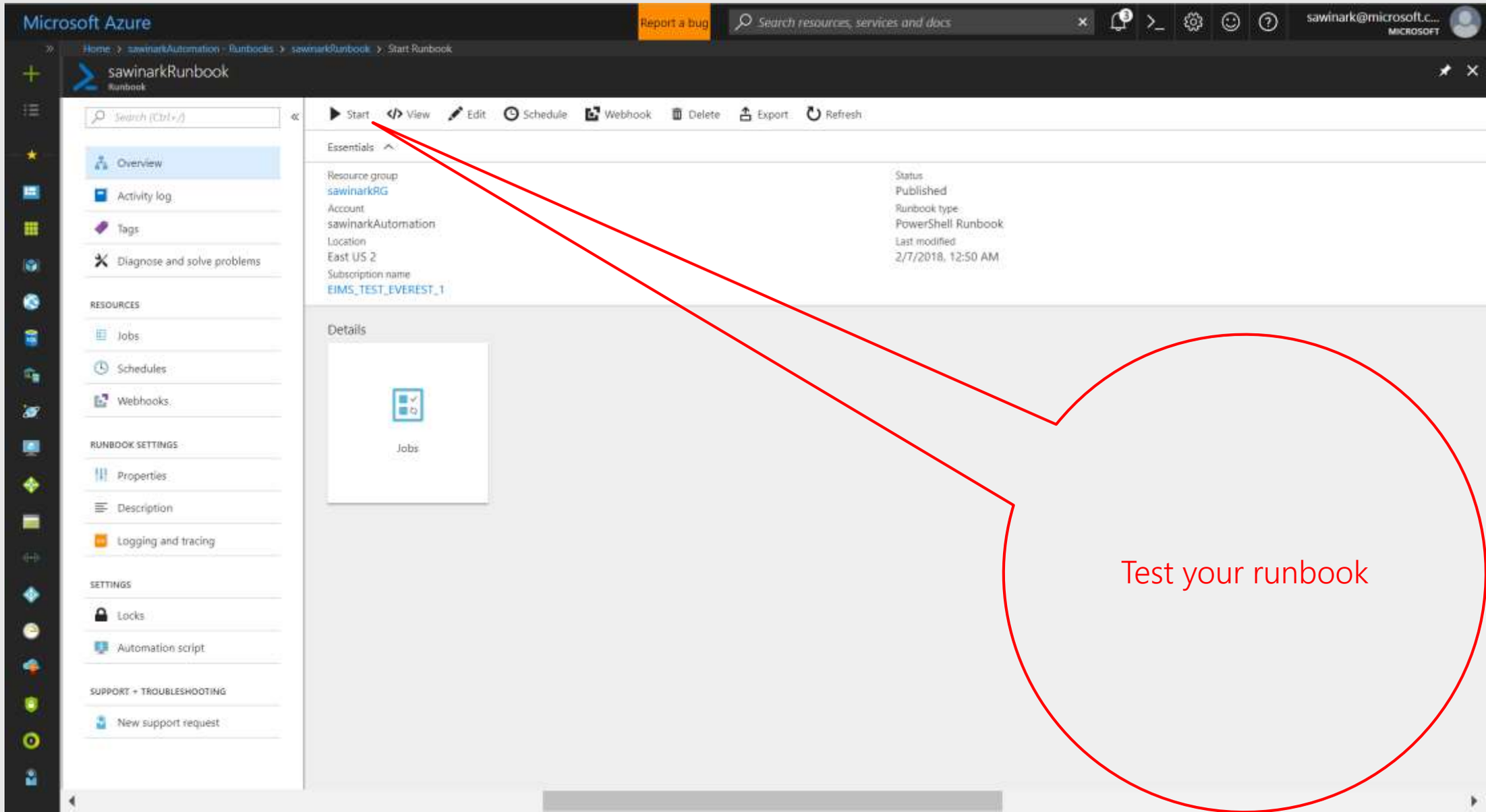
# On-Demand/Just-In-Time Provisioning

```
##### PSH script for Automation runbook to start/stop Azure-SSIS IR #####
Param
(
    [Parameter (Mandatory= $true)]
    [String] $ResourceGroupName,

    [Parameter (Mandatory= $true)]
    [String] $DataFactoryName,

    [Parameter (Mandatory= $true)]
    [String] $AzureSSISName,

    [Parameter (Mandatory= $true)]
    [String] $Operation
)

$connectionName = "AzureRunAsConnection"
try
{
    # Get the connection "AzureRunAsConnection "
    $servicePrincipalConnection=Get-AutomationConnection -Name $connectionName

    "Logging in to Azure..."
    Add-AzureRmAccount `
        -ServicePrincipal `
        -TenantId $servicePrincipalConnection.TenantId `
        -ApplicationId $servicePrincipalConnection.ApplicationId `
        -CertificateThumbprint $servicePrincipalConnection.CertificateThumbprint
}
```

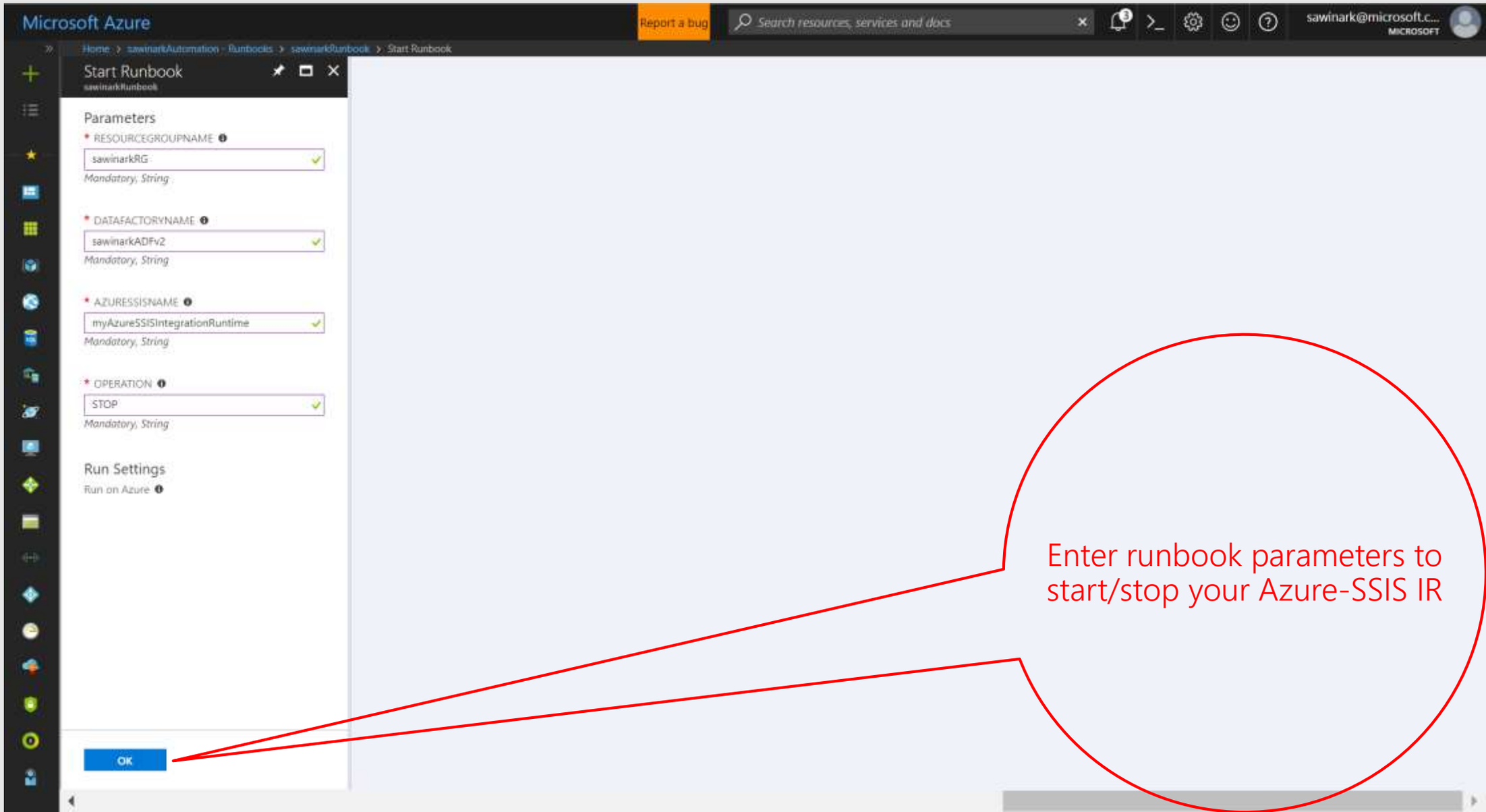
# On-Demand/Just-In-Time Provisioning

```
catch {
    if (!$servicePrincipalConnection)
    {
        $ErrorMessage = "Connection $connectionName not found."
        throw $ErrorMessage
    } else{
        Write-Error -Message $_.Exception
        throw $_.Exception
    }
}

if($Operation -eq "START" -or $operation -eq "start")
{
    "##### Starting #####"
    Start-AzureRmDataFactoryV2IntegrationRuntime -ResourceGroupName $ResourceGroupName -DataFactoryName $DataFactoryName -Name $AzureSSISName -Force
}
elseif($Operation -eq "STOP" -or $operation -eq "stop")
{
    "##### Stopping #####"
    Stop-AzureRmDataFactoryV2IntegrationRuntime -DataFactoryName $DataFactoryName -Name $AzureSSISName -ResourceGroupName $ResourceGroupName -Force
}
"##### Completed #####"
```

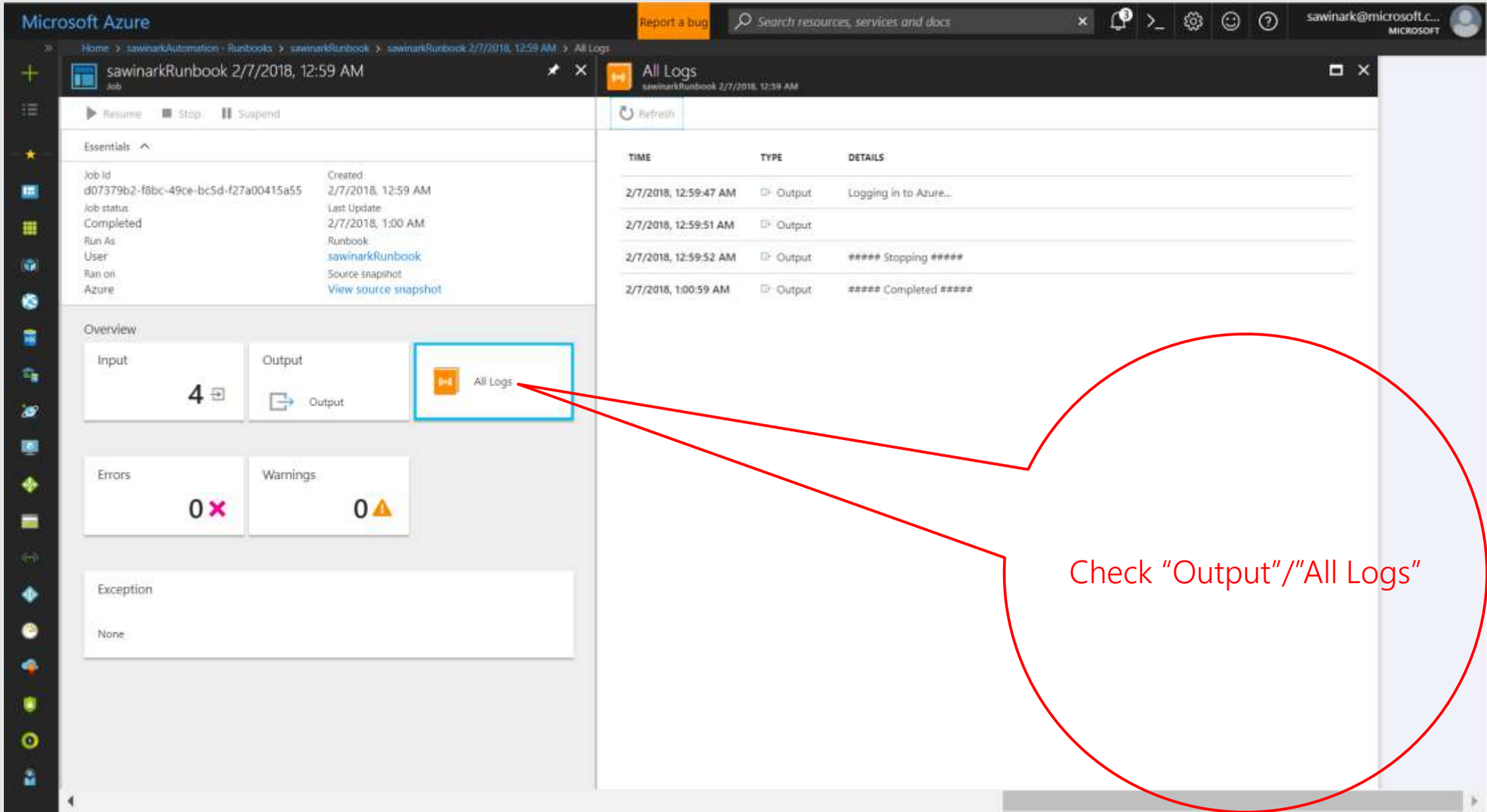
# On-Demand/Just-In-Time Provisioning



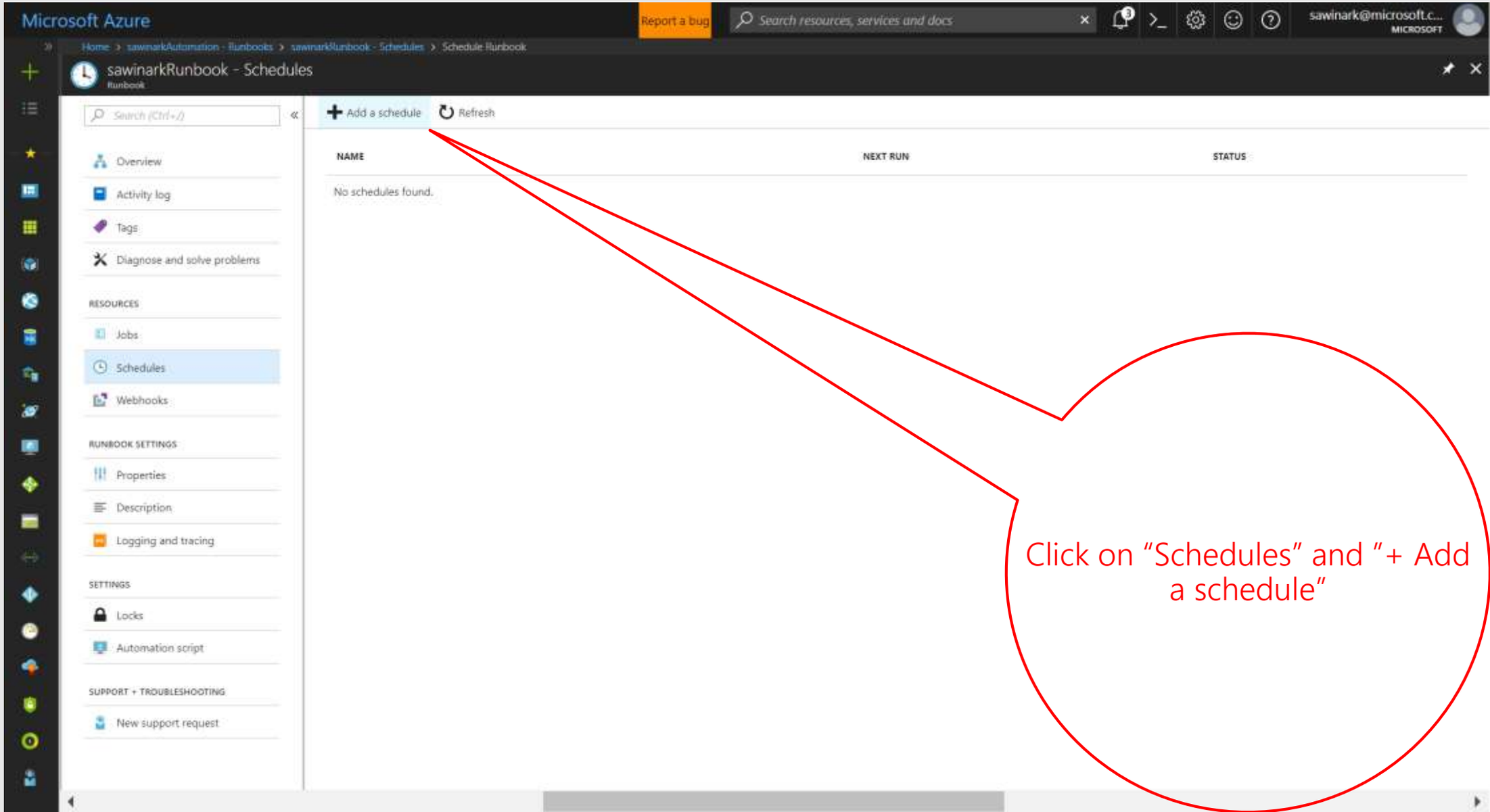
# On-Demand/Just-In-Time Provisioning



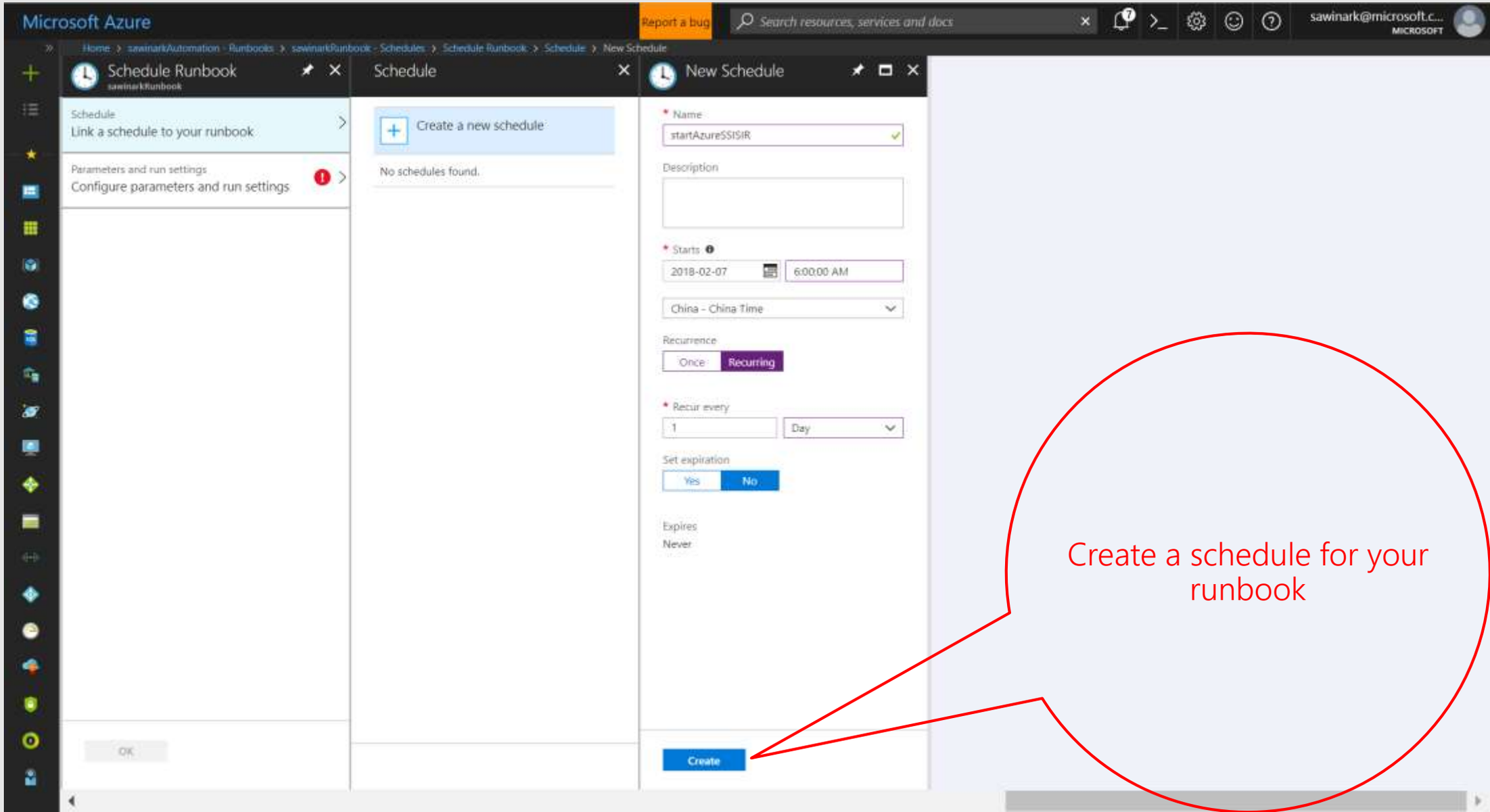
# On-Demand/Just-In-Time Provisioning



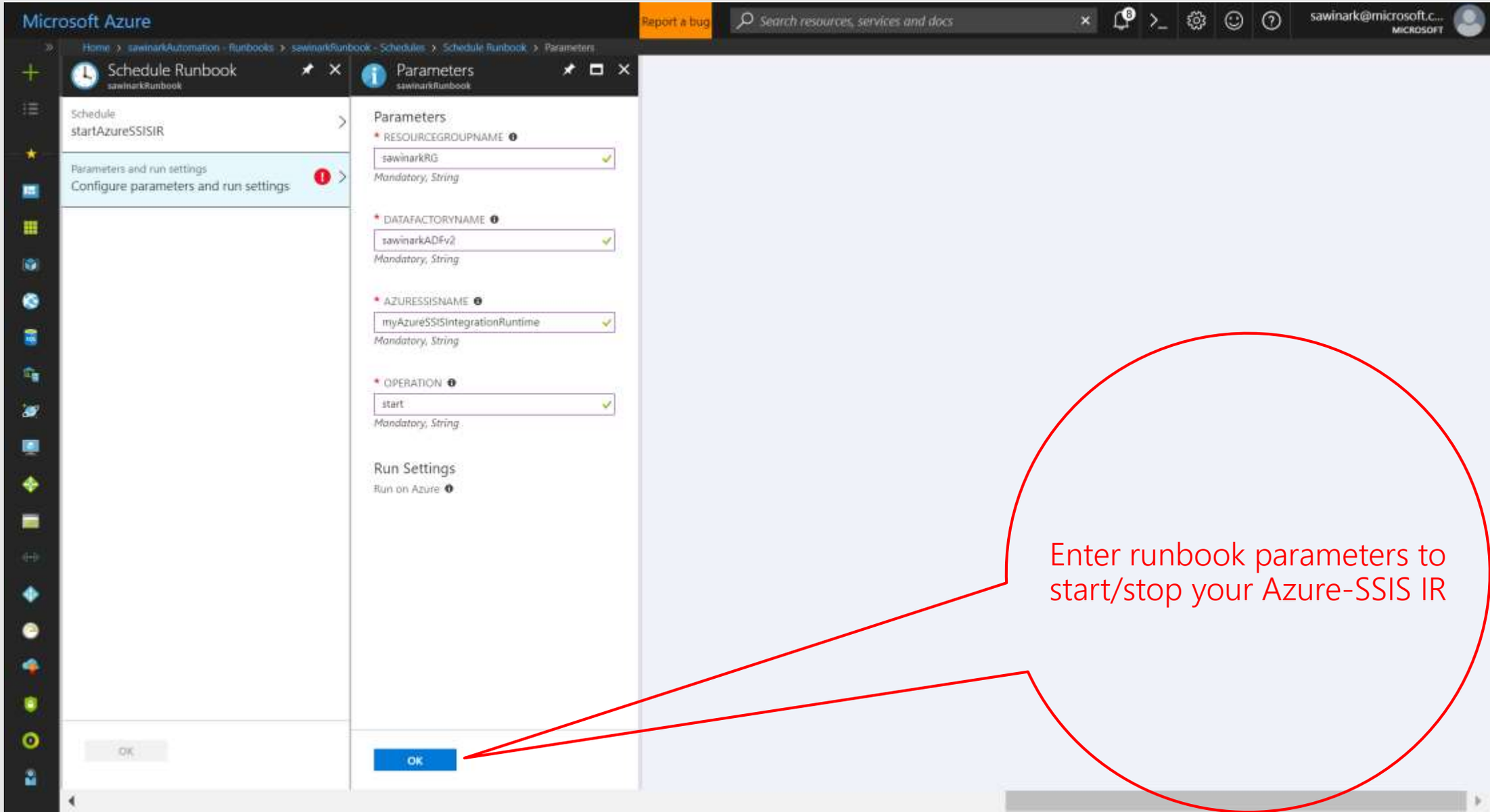
# On-Demand/Just-In-Time Provisioning



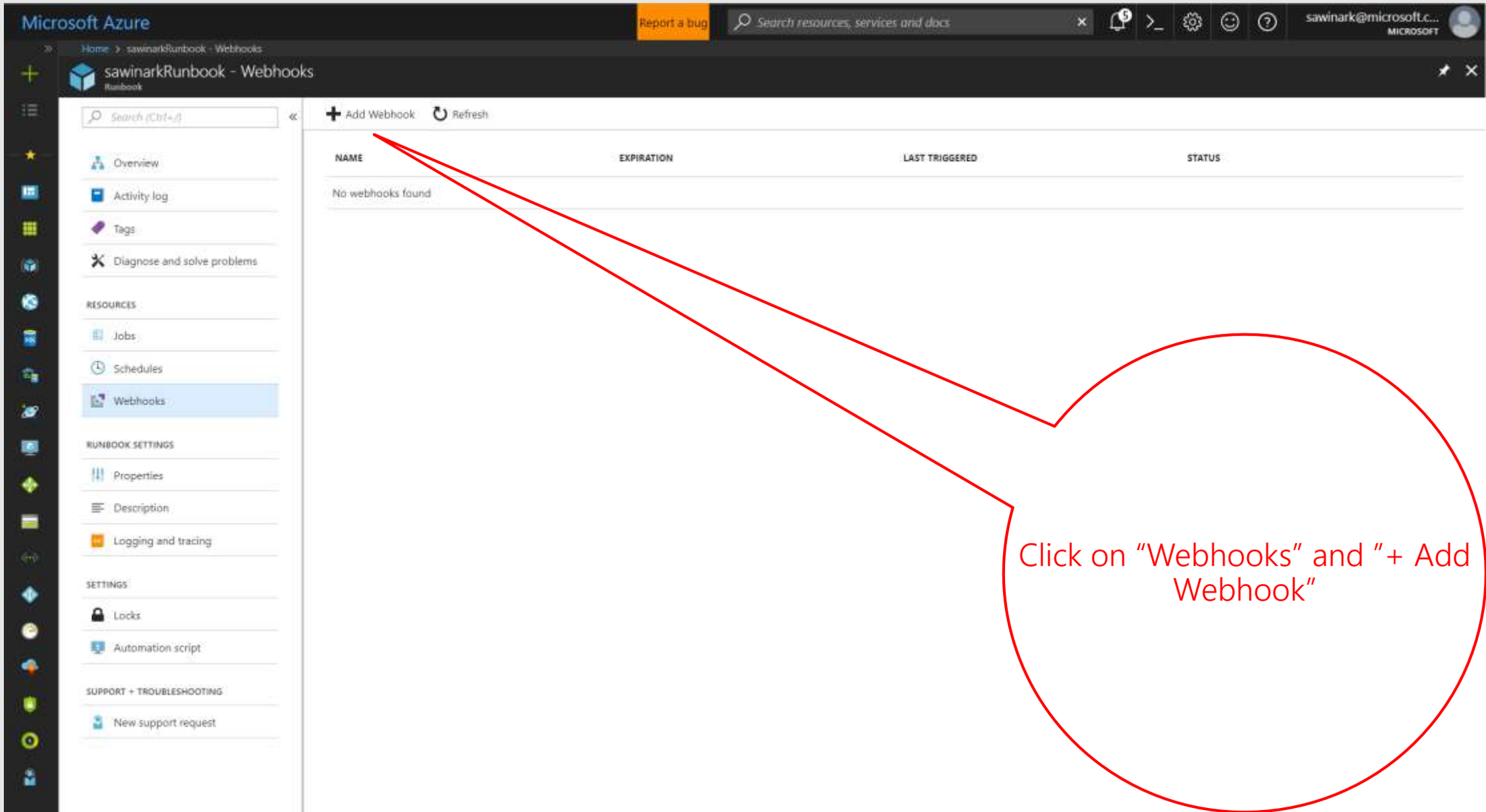
# On-Demand/Just-In-Time Provisioning



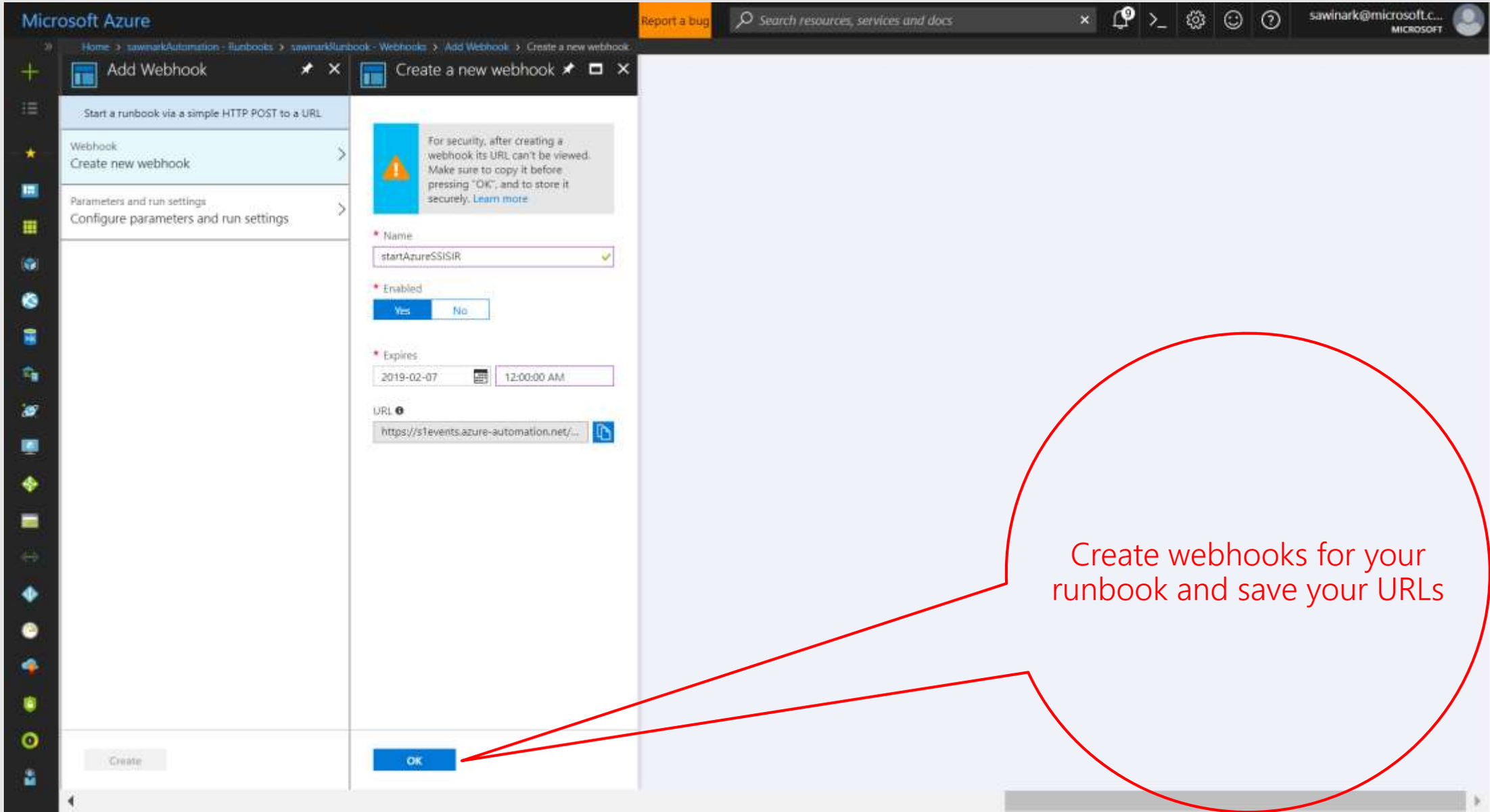
# On-Demand/Just-In-Time Provisioning



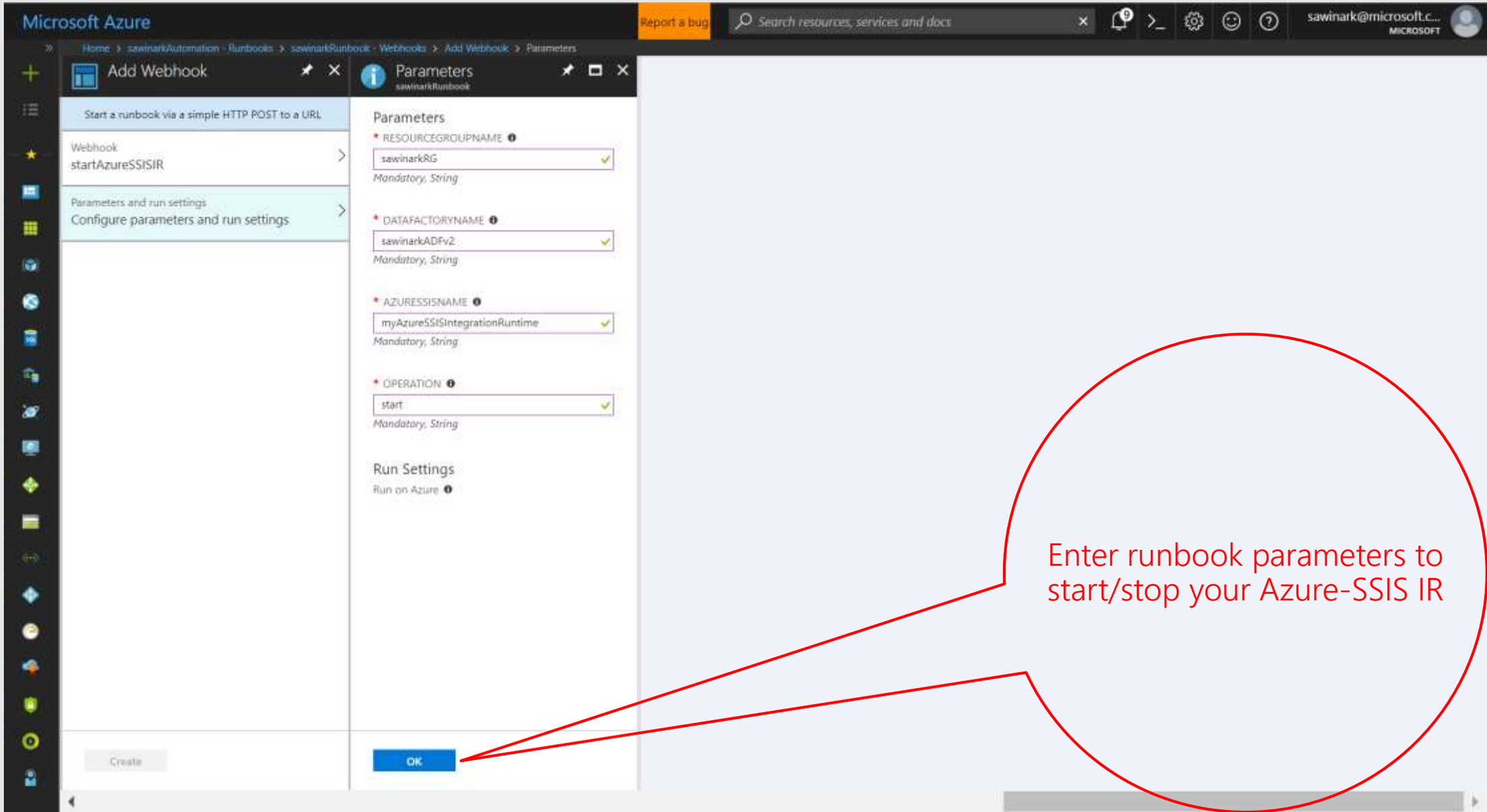
# On-Demand/Just-In-Time Provisioning



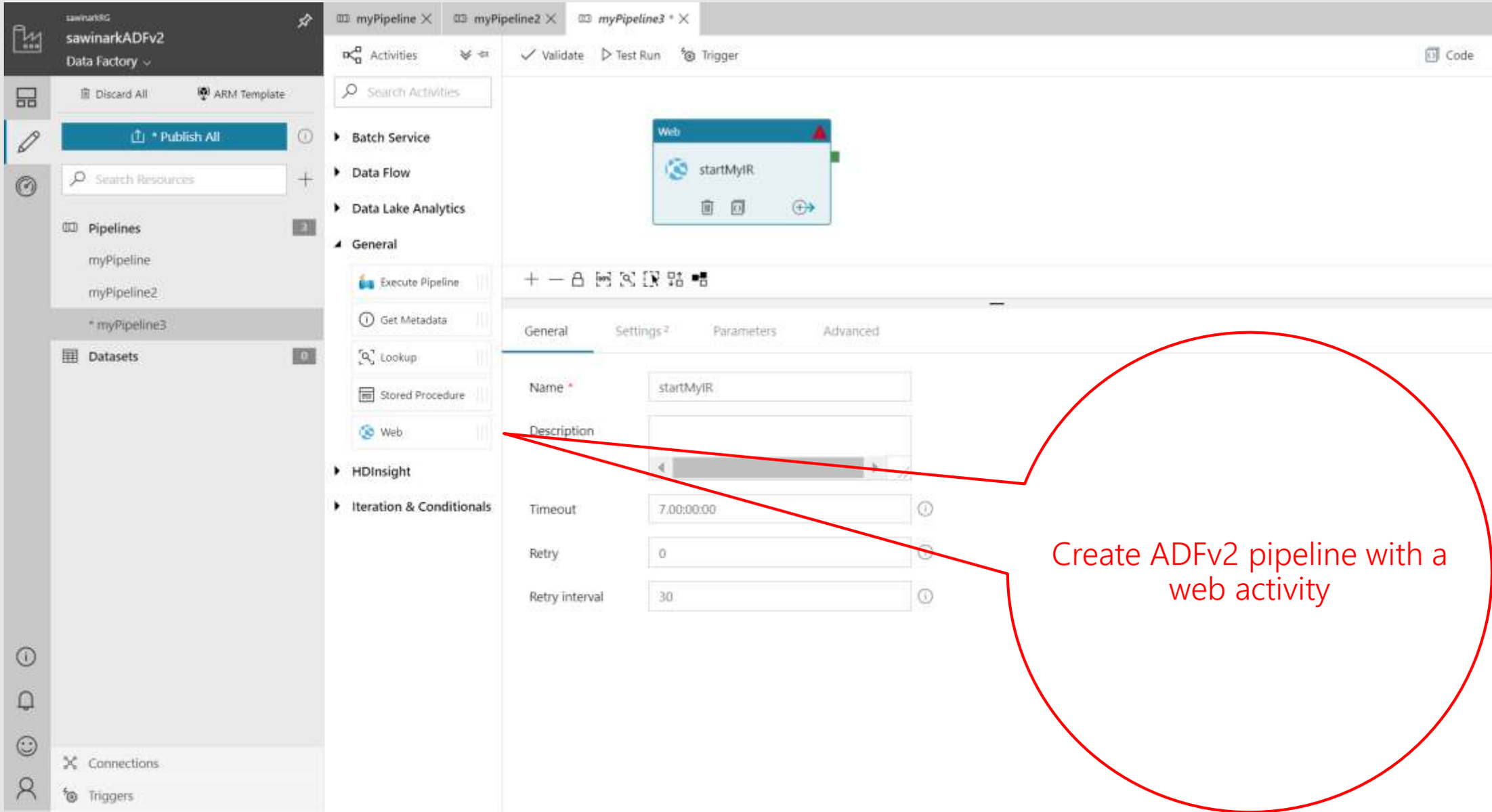
# On-Demand/Just-In-Time Provisioning



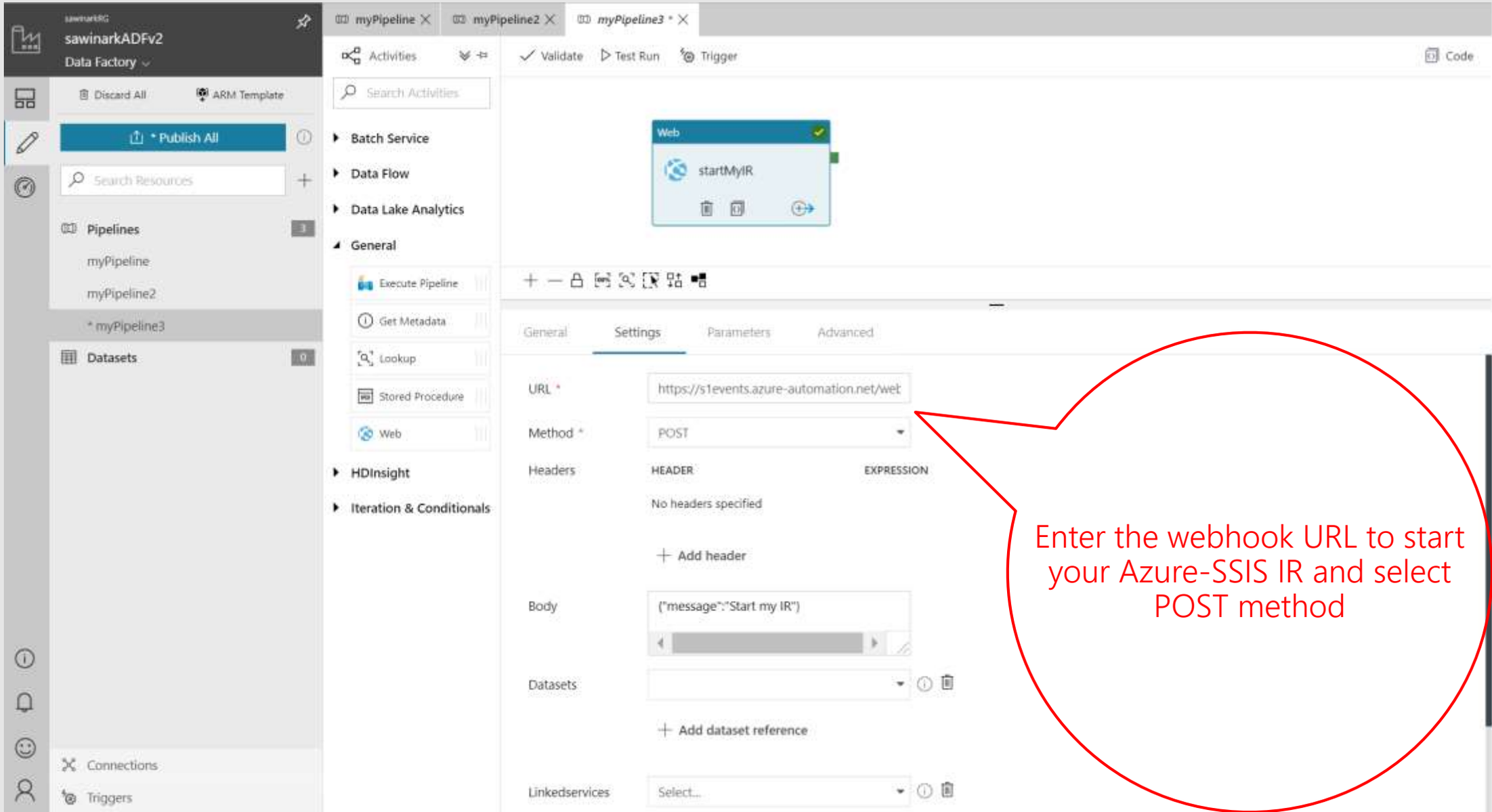
# On-Demand/Just-In-Time Provisioning



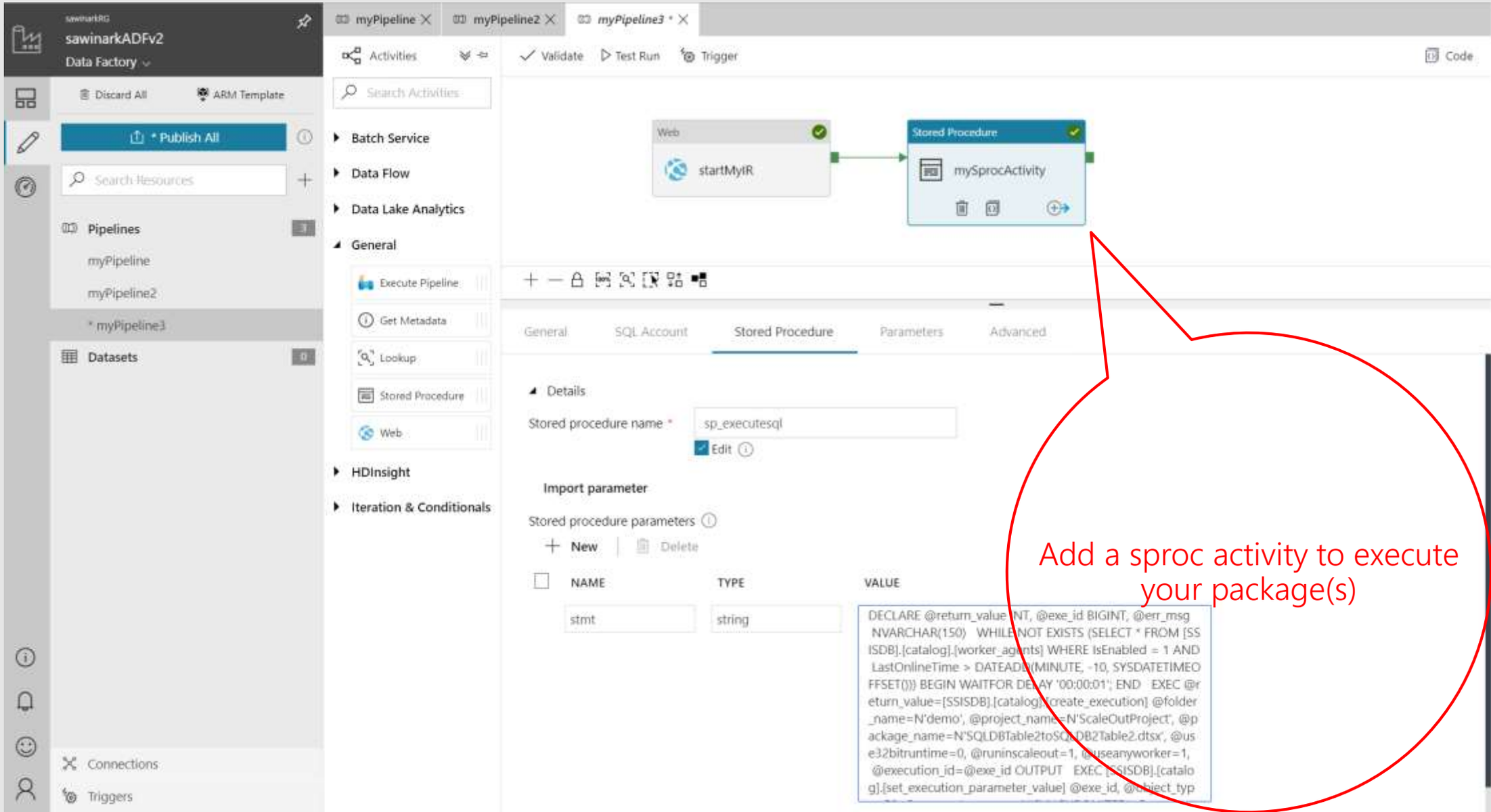
# On-Demand/Just-In-Time Provisioning



# On-Demand/Just-In-Time Provisioning



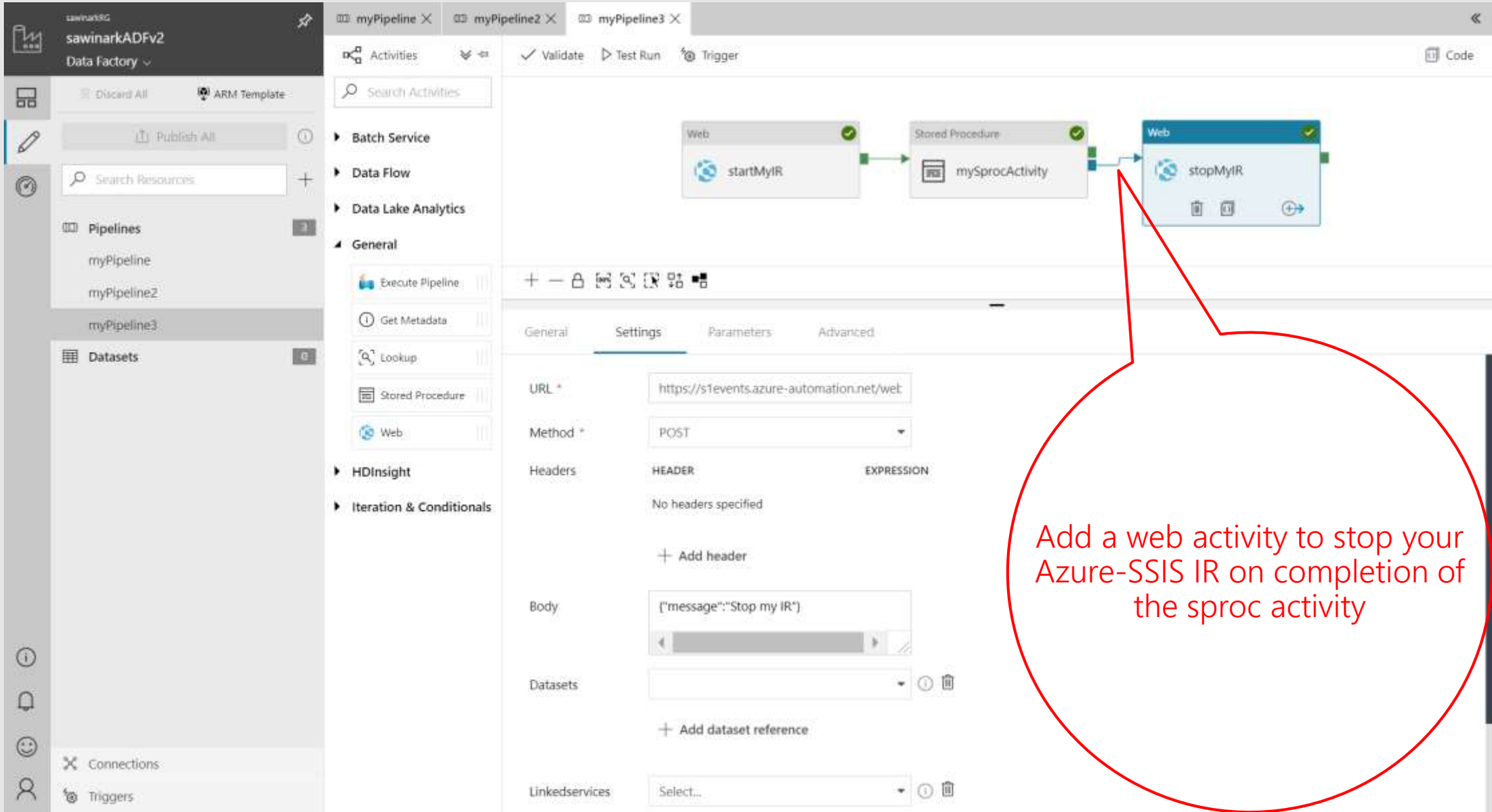
# On-Demand/Just-In-Time Provisioning



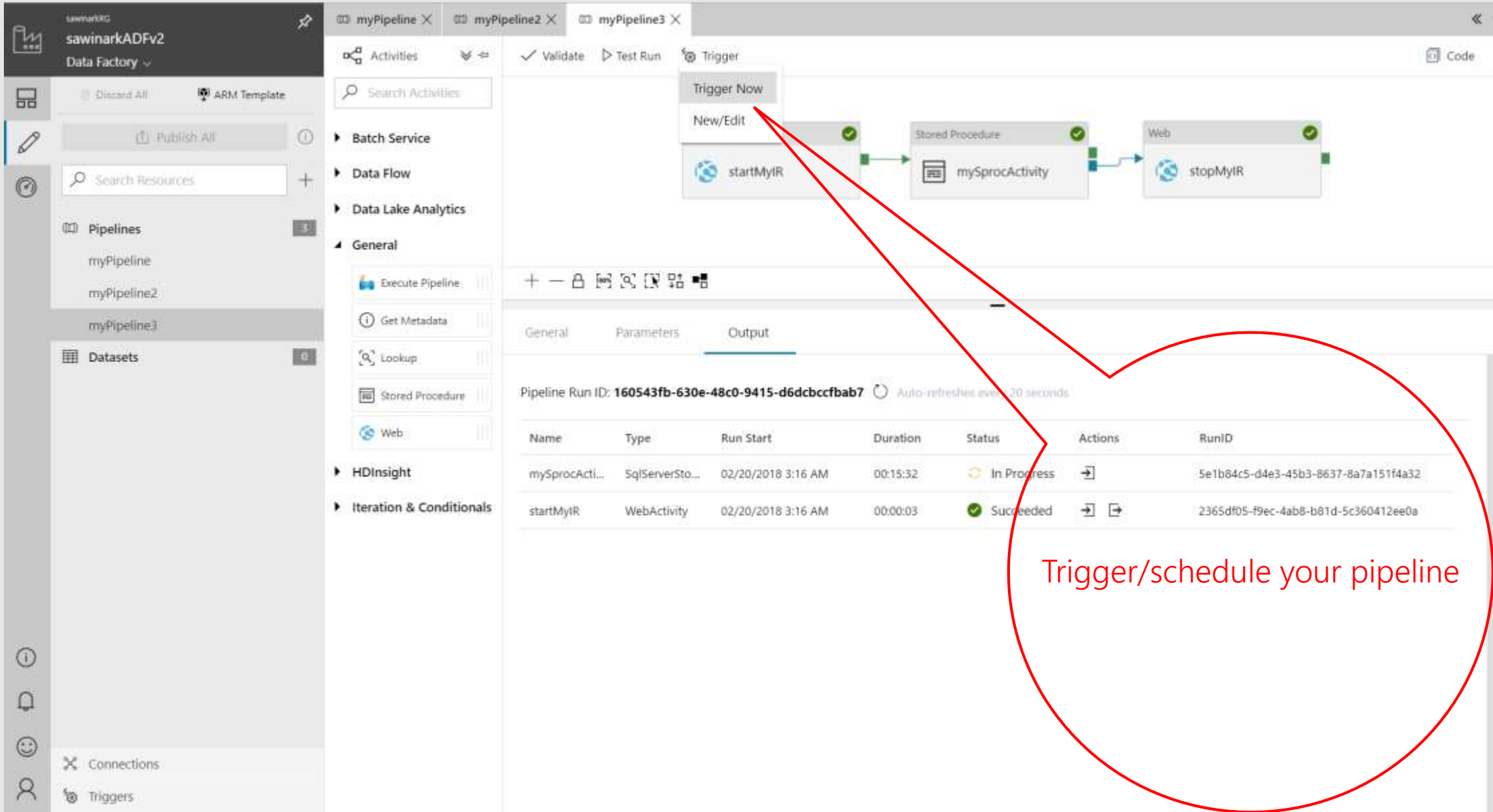
# On-Demand/Just-In-Time Provisioning

```
-- T-SQL script to create/start SSIS package execution using SSISDB sprocs
DECLARE    @return_value int, @exe_id bigint, @err_msg nvarchar(150)

-- Wait until Azure-SSIS IR is started
WHILE NOT EXISTS (SELECT * FROM [SSISDB].[catalog].[worker_agents] WHERE IsEnabled = 1 AND LastOnlineTime > DATEADD(MINUTE, -10,
SYSDATETIMEOFFSET()))
BEGIN
       WAITFOR DELAY '00:00:01';
END

EXEC @return_value = [SSISDB].[catalog].[create_execution] @folder_name=N'YourFolder', @project_name=N'YourProject',
           @package_name=N'YourPackage', @use32bitruntime=0, @runincluster=1, @useanyworker=1,
           @execution_id=@exe_id OUTPUT

-- To synchronize SSIS package execution, set SYNCHRONIZED execution parameter
EXEC [SSISDB].[catalog].[set_execution_parameter_value] @exe_id, @object_type=50, @parameter_name=N'SYNCHRONIZED', @parameter_value=1

EXEC [SSISDB].[catalog].[start_execution] @execution_id = @exe_id, @retry_count = 0

-- Raise an error for unsuccessful package execution, check package execution status = created (1)/running (2)/canceled (3)/failed (4)/
-- pending (5)/ended unexpectedly (6)/succeeded (7)/stopping (8)/completed (9)
IF (SELECT [status] FROM [SSISDB].[catalog].[executions] WHERE execution_id = @exe_id)<>7
BEGIN
           SET @err_msg=N'Your package execution did not succeed for execution ID: '+ CAST(@execution_id as nvarchar(20))
           RAISERROR(@err_msg, 15, 1)
END
```

# On-Demand/Just-In-Time Provisioning



# On-Demand/Just-In-Time Provisioning



# Resources

# Contacts

- My email: sawinark@microsoft.com

- SSIS online survey: https://www.surveybuilder.com/s/Dg6Bq?source_type=email

- SSIS Advisors Yammer Group: https://www.yammer.com/azureadvisors/#/threads/inGroup?type=in_group&feedId=12090139&view=all

# Documentations

- https://docs.microsoft.com/en-us/azure/data-factory/tutorial-create-azure-ssis-runtime-portal
- https://docs.microsoft.com/en-us/azure/data-factory/join-azure-ssis-integration-runtime-virtual-network
- https://docs.microsoft.com/en-us/sql/integration-services/lift-shift/ssis-azure-connect-to-catalog-database
- https://docs.microsoft.com/en-us/sql/integration-services/lift-shift/ssis-azure-connect-with-windows-auth
- https://docs.microsoft.com/en-us/sql/integration-services/ssis-quickstart-deploy-ssms
- https://docs.microsoft.com/en-us/sql/integration-services/ssis-quickstart-run-ssms
- https://docs.microsoft.com/en-us/sql/integration-services/lift-shift/ssis-azure-schedule-packages
- https://docs.microsoft.com/en-us/azure/data-factory/how-to-invoke-ssis-package-stored-procedure-activity
- https://docs.microsoft.com/en-us/azure/data-factory/monitor-integration-runtime#azure-ssis-integration-runtime
- https://docs.microsoft.com/en-us/azure/data-factory/manage-azure-ssis-integration-runtime
- https://docs.microsoft.com/en-us/azure/data-factory/how-to-schedule-azure-ssis-integration-runtime

Microsoft