INDEX

|  |  |  |
| --- | --- | --- |
| Sr No | Topic | Page No |
| 1 | [Deploy Resources from Phoenix-PROD to EBI-DEV](#Deploy_Phoenix_EBIDEV) | 2-8 |
| 2 | [Deploy Resources from EBI-DEV to EBI-QA](bookmark://Deploy_EBIDEV_EBIQA) | 9-14 |
| 3 | [Deploy Resources from EBI-QA to EBI-PROD](bookmark://Deploy_EBIQA_EBIPROD) | 15-18 |
| 4 | [General Steps to set up Azure DevOps CI/CD Process](bookmark://Steps_CICD) | 19 |
| 5 | [Deploy a Function App to a Target Subscription](#Deploy_FunctionApp) | 20-23 |
| 6 | [Deploy a Data factory to a Target Subscription](bookmark://Deploy_ADF) | 24-29 |
| 7 | [Deploy a Logic App to a Target Subscription](bookmark://Deploy_LogicApp) | 30-34 |
| 8 | [Deploy an API Connection to a Target Subscription](bookmark://Deploy_APIconnection) | 35-39 |

**How to deploy Azure Resources from Phoenix-PROD to EBI-DEV**

(Reference: focus-master Logic app and all its dependent resources)

1. Strategy for deployment: We will use a mixture of CI/CD Azure DevOps Pipelines, and Bicep Deployment using VS Code and Azure CLI depending on the resource being deployed.
2. List of Resources for Deployment: Keep ready - a list of all the resources that need to be deployed. (Function apps, ADF Pipelines, Logic apps and API connections)

For example,

What are all the services (based on the [focus-master ST Linkage Sheet](https://capgemininar.sharepoint.com/:x:/r/sites/FourSeasonsTeamSite/Shared%20Documents/Data%20Program%20Delivery/Migration%20Stream/Pilot%20Analysis/Focus-master%20ST%20Linkage.xlsx?d=w5139efa310f84d8088d4baaf2316086b&csf=1&web=1&e=KJZfcd)) that we need to deploy into the target subscription?

1. Function Apps

* fsprodfuncpipelineintegrationà GetFileRowCount
* fsprodfuncfocusintegrationàMDXQuery
* fsprodfuncfocusintegrationàQueryEssbase
* fsprodfuncfocusintegrationà GetMemberInfo

1. ADF Pipelines

* Crosscheck-pms\_focus
  + C1\_to\_C2\_01-Master
  + C1\_to\_C2\_02-CreateDataFile
  + C1\_to\_C2\_03-UpdateExternalTable
  + C1\_to\_C2\_04-RebuildTable
* Focus\_00\_LoadTriggerFiles
* Focus\_99\_NoPropertiesToProcess
* Focus\_01\_MainLoad
  + Focus\_02\_CubeLanding
  + Focus\_03\_CubeLanding\_Property
  + Focus\_03b\_CubeLanding\_Property\_RunMDX
  + Focus\_03a\_CubeLanding\_Property\_LandEssbaseData
  + Focus\_04\_Ingest-to-Raw
  + Focus\_05\_Build-Dim
  + Focus\_06\_Build-Fact
  + DataLake\_RemoveEmptyFile

1. API Connections

* Azure\_sftp
* Azuredatafactory\_connection
* Azuredatalake\_connection
* Cosmosdb\_connection
* Sqldw\_sproc

1. Logic Apps

* Focus-master
* Ingest-land\_files
* Admin-validate\_file
* admin-datalake\_check\_file\_exists
* crosscheck-pms\_focus
* admin-run\_adf\_pipeline
* admin-send-email

1. Order of Deployment of Resources: Keep ready - the order in which the resources need to be deployed. The order of deployment will be from least dependency to greatest dependency. For focus-master logic app, we will decide the order of dependency and hence the order of deployment based on the focus-master ST (Source to Target) Linkage Document ([Focus-master ST Linkage.xlsx).,](https://capgemininar.sharepoint.com/:x:/r/sites/FourSeasonsTeamSite/Shared%20Documents/Data%20Program%20Delivery/Migration%20Stream/Pilot%20Analysis/Focus-master%20ST%20Linkage.xlsx?d=w5139efa310f84d8088d4baaf2316086b&csf=1&web=1&e=KJZfcd) the general order will be as follows:
2. Function apps
3. ADF Pipelines
4. API Connections
5. Logic apps

Note: Within each of the services mentioned above, the resource that has least dependency will be deployed first. For example, within focus-master Logic app, multiple dependent Logic apps exist. First, we need to deploy the internal dependent Logic apps, then we need to deploy the focus-master Logic app.

1. Pre-requisites to deployment:
   1. Create Keys in Azure Key Vault – These Keys will be referenced at the time of deployment and are necessary to create Linked Services for the resources mentioned below- **(No need for Azure Key Vault if Linked service connections can be made through Managed Identity)**
      1. Function apps
      2. Azure Synapse Analytics /Dedicated SQL pool (formerly Azure SQL Data Warehouse)
      3. SQL Database
      4. Azure Synapse Analytics/Apache Spark pool (previously functionality was carried out by Azure Data Lake Analytics)
      5. ADLS Gen 2 (previously functionality was carried out by ADLS Gen 1)
      6. ADLS Gen 1 (Required for ADLS Gen 1 Linked Service in Gen1-Gen2 ADF Pipeline)
   2. Set up code repository in Data Factory: Follow steps 1,2,3 mentioned in Option 1 of [Deploy a Data factory to a Target Subscription](bookmark://Deploy_ADF) to create a development branch and main (feature) branch in the adf-phx-migr-dev Data Factory.

Use ‘[Enter Repository name for ADF]’ under ‘Repository Name’

Use ‘main’ branch as ‘Collaboration branch’

Use ‘adf\_publish’ branch as ‘Publish branch’

Import resources into this branch – select ‘development’ branch

* 1. Create Linked Services in the Azure Data Factory (adf-phx-migr-dev) in the EBI-DEV subscription (in FS.EBI.Dev.Integration Resource Group) before deployment because these Linked Service names need to be provided as parameter values during Azure Data Factory Deployment.

Linked Services need to be created for:

* + 1. Azure Key Vault
    2. Function apps
    3. Dedicated SQL Pool (formerly Azure SQL Data Warehouse)
    4. SQL Database (This will contain config tables with the config settings (master settings, focus settings, etc.) previously contained in Cosmos DB) - Decision pending from Sumit/FS team on whether SQL Database or Cosmos DB will be used for storing config settings
    5. Azure Synapse Analytics (previously functionality was carried out by Azure Data Lake Analytics)
    6. ADLS Gen 2 (previously functionality was carried out by ADLS Gen 1)
    7. ADLS Gen 1 (used in Gen1-Gen2 pipeline)

Important Note:

The Azure Data Factory ARM Template contains Dataset and Linked Service references to the services that existed in the previous architecture in the Phoenix-PROD environment. This includes Azure SQL Data Warehouse, Azure Data Lake Analytics and ADLS Gen 1. At the time of deployment, we need to provide Linked Service references to the new Linked Services that have been created above.

However, the ARM Template will create Datasets based on the previous architecture and these Datasets will reference resources in the FS.EBI.Dev.Integration Resource Group that do not exist.

This isn’t an issue, at the time of development we can change these Dataset references to the new resources that need to be used for the same functionality, example, ADLS Gen 2 instead of ADLS Gen 1.

* 1. Create Gen1-Gen2 ADF Pipeline-

This pipeline will copy the files and folders present in the ADLS Gen 1 account to the ADLS Gen 2 account.

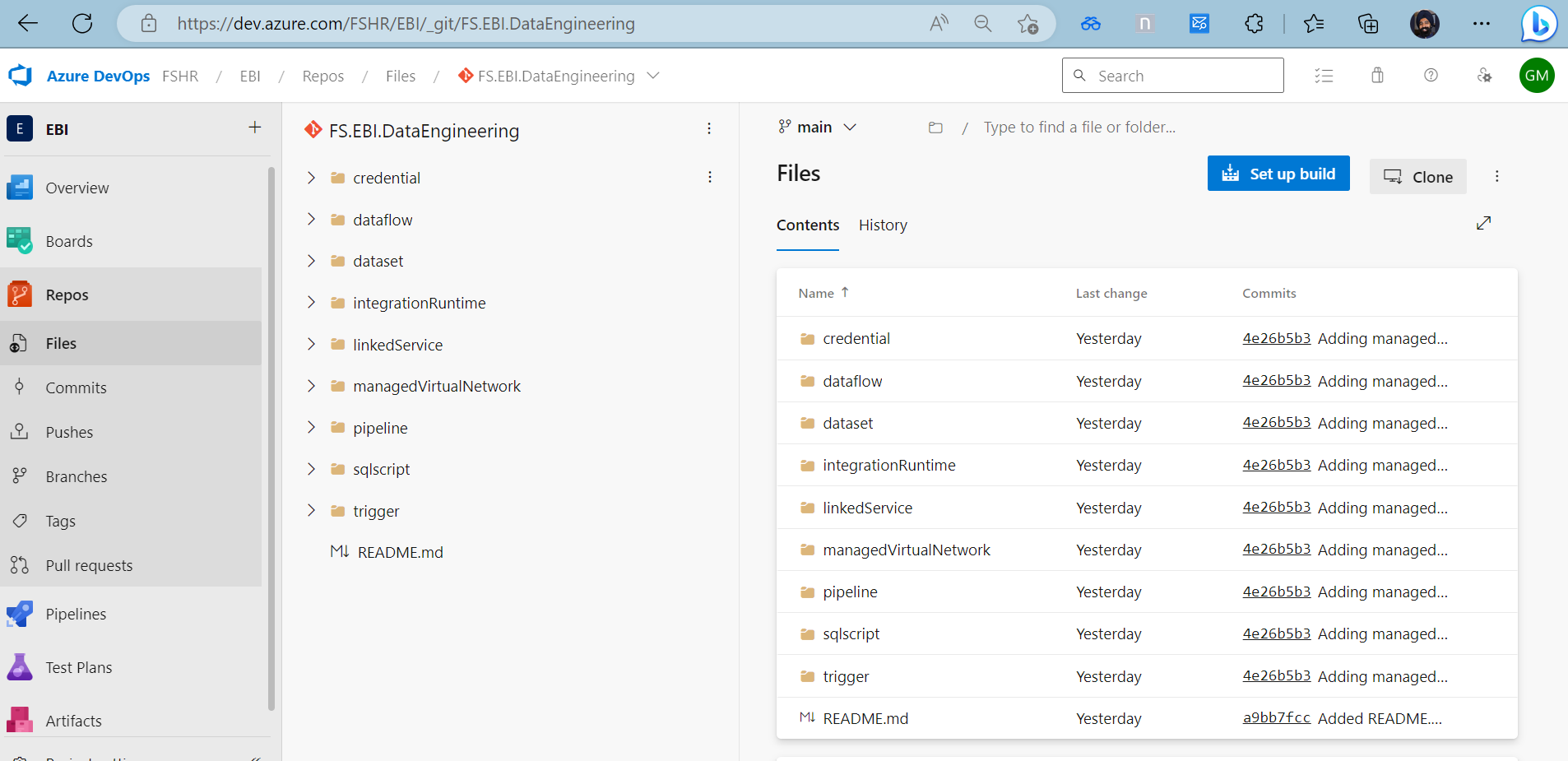
At the time of ADF deployment, we need to provide the new folder locations present in ADLS Gen 2.

Steps for migrating files and folders from Gen1 to Gen 2 are mentioned below:

* + 1. Go to adf-phx-migr-dev Data factory in FS.EBI.Dev.Integration Resource Group in EBI-DEV Subscription.
    2. Under Pipeline, click on the “...” and select ‘New Pipeline’. Name your pipeline.
    3. Under ‘Move and Transform’ drag and drop ‘Copy data’ onto the canvas.
    4. In the Copy Activity, under ‘General’, name the Copy Activity appropriately.
    5. Under ‘Source’, select ‘Source dataset’ and under ‘Sink’, select ‘Sink dataset’.
    6. To create Source/Sink dataset, click on ‘+New’, select Azure Data Lake Gen 1 for Source and Azure Data Lake Gen 2 for Sink.
    7. Select the format type of data
    8. Select the Linked Services for ADLS Gen 1 and ADLS Gen 2. They have been created in Step 3a.
    9. We can provide the folder path for the source and sink.
    10. Repeat steps (iii) to (ix) above for migrating other folders and files from ADLS Gen 1 to ADLS Gen 2.

We will schedule the pipeline to run regularly depending on the frequency of files getting added into the ADLS Gen 1 Storage Account.

* 1. Confirm ARM Template Location of Azure Resources in Azure DevOps Repo: In order to create a CI/CD Pipeline for an Azure Resource (ADF and Logic app in this case) using the steps mentioned in ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’, we need to confirm the ARM Template and ARM Template parameters file for the resource are in the ‘main’ branch of the ‘Repos/Files’ section in the Azure DevOps Project ‘Phoenix’.



1. Start Deployment of Resources: Once we have the list of resources and the order in which resources need to be deployed ready, we can start the deployment.

Note: As and when a resource is successfully deployed, tick them off the ‘List of resources for deployment’ so we have an inventory of which resources have been deployed and which resources are yet to be deployed.

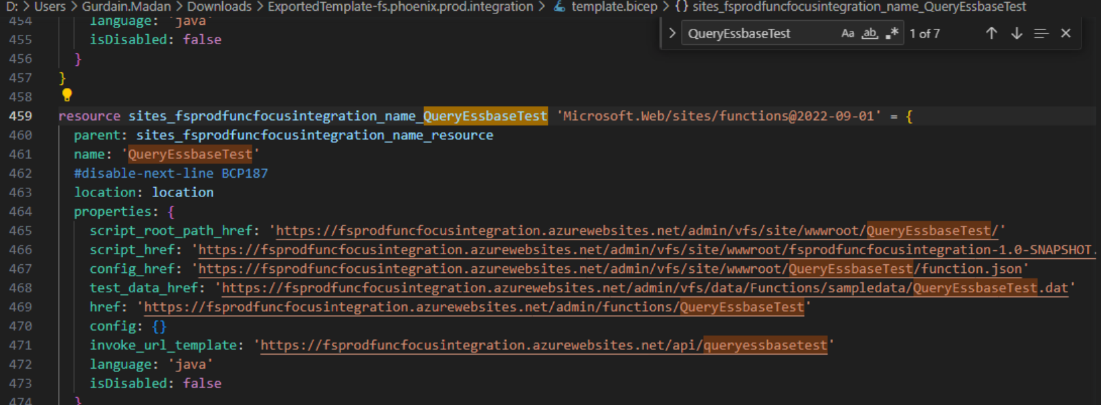
Step 1: Start Deployment of Function apps

1. From the ‘List of resources for deployment’, we have the list of Function apps that need to be deployed.
2. Select the first Function app. For example, fsprodfuncfocusintegration. Within this Function app, we have multiple functions, not all these functions need to be deployed. Now, we have 2 options -

B.1) Deploy the Function app along with all functions. Once the Function app is deployed in the target subscription, delete the Functions that aren’t required. Deleting the Functions isn’t exactly straightforward; however, it is possible - [How to remove Azure Functions without deleting the Functions App? (koskila.net)](https://www.koskila.net/how-to-remove-azure-functions-without-deleting-the-app-service/)

B.2) At the time of deployment, we can remove the code from the Bicep Template for the Functions that aren’t required.

For example, if we need to remove the Function ‘QueryEssbaseTest’ from the Bicep Template, we need to search for ‘QueryEssbaseTest’, there will be one resource created for this Function, all we need to do is delete the piece of code for this resource.



1. Once we have decided whether we want to go ahead with Option B.1 or Option B.2, then we need to refer to the [Steps to deploy a Function app in a Target Subscription](#Deploy_FunctionApp).
2. Repeat steps B, C for the next Function app in the list.

Step 2: Start Deployment of ADF Pipelines

1. From the ‘List of resources for deployment’, we have the list of ADF Pipelines that need to be deployed.
2. Follow the steps mentioned in Option 1 of – [How to deploy a Data Factory to a Target subscription](bookmark://Deploy_ADF)

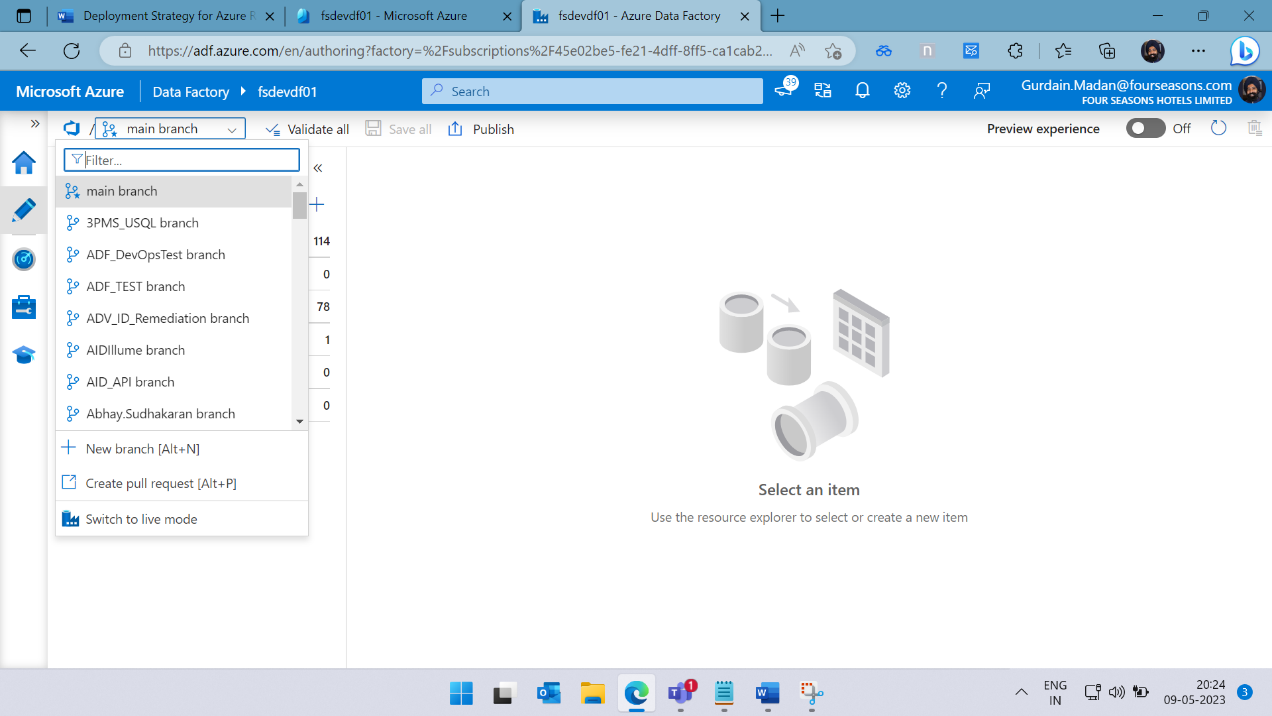
Option 1 or CI/CD DevOps deployment of ADF is the preferred approach to Option 2 or ARM Template deployment and is currently being applied for Phoenix.

Justification: Option 2 or ARM Template deployment of ADF mentioned in [How to deploy a Data Factory to a Target subscription](bookmark://Deploy_ADF) is a manual one-time process, we want our source and target to be integrated in a manner such that any changes in our source are directly deployed into our target via an automated pipeline. This is possible in Option 1.

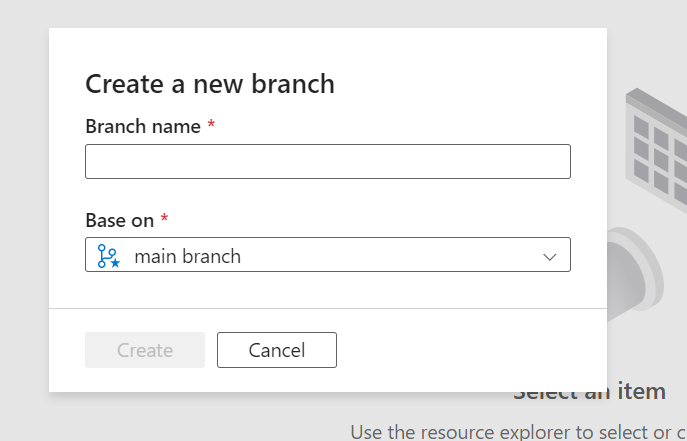
Step number 7.h.vii is important – This is where we need to make changes as per desired deployment requirements. For example, update the values for the services that have already been created, for example, key vault base URL, linked service names, etc.

Step number 7.i and 9.a are for Continuous Integration and Continuous Deployment. Depending on the requirement, we can enable or disable these options.

1. We now have our source ADF deployed in the target subscription.
2. Connect the ADF in our target subscription to Azure DevOps – Refer to Step number 1 ‘Set up code repository’ in Option 1 of [How to deploy a Data Factory to a Target subscription.](bookmark://Deploy_ADF)
3. Create a new branch – Click on ‘+ New branch [Alt + N] ‘ in the dropdown shown in the screenshot.



1. The new branch will be the development branch where all the development will be carried out.



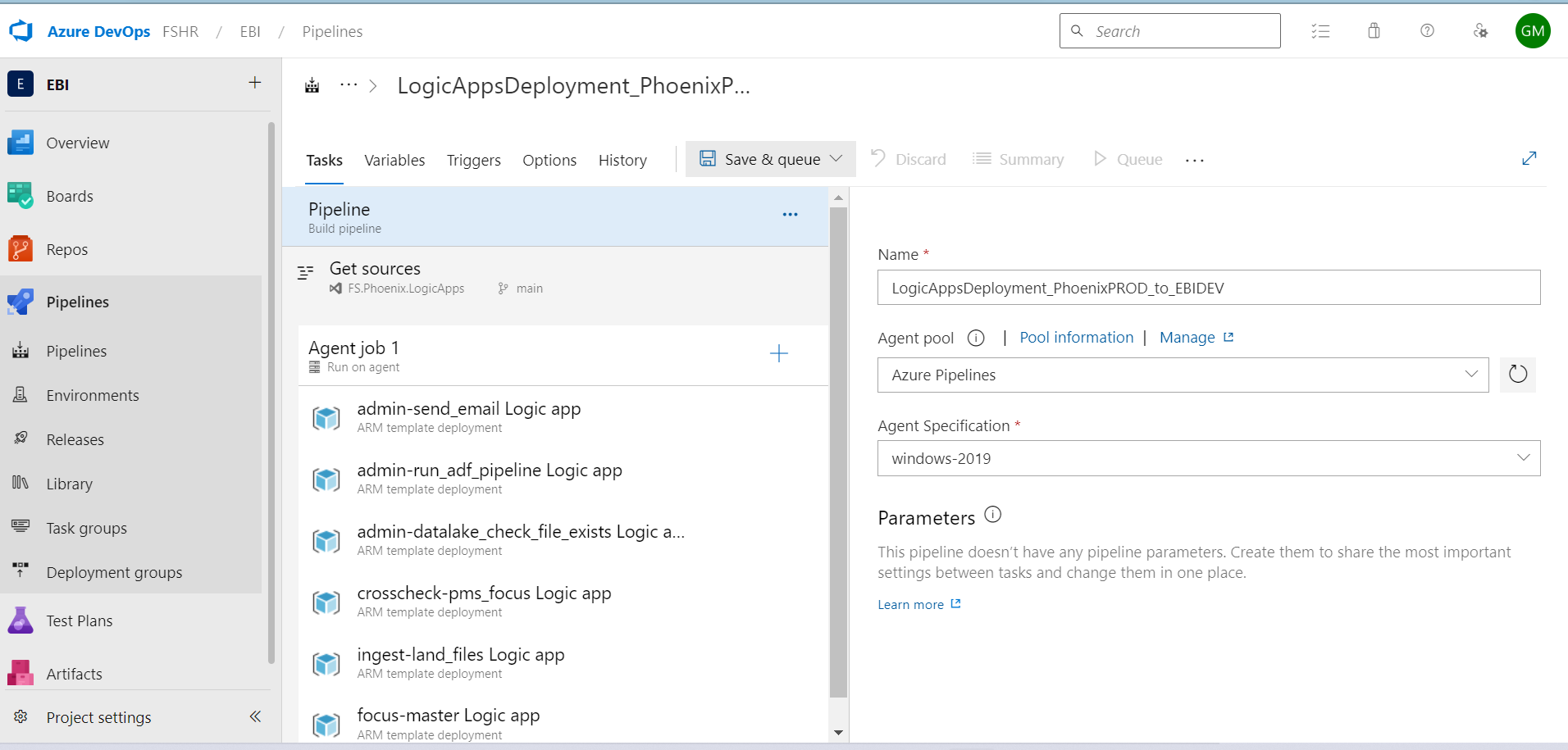
1. As development continues, Edit the main branch to have only those pipelines that are ready to be deployed to EBI-QA Environment for testing.
2. The main branch is our feature branch.

Step 3: Start Deployment of API connections

1. From the ‘List of resources for deployment’, we have the list of API connections that need to be deployed.
2. Take one API connection at a time, deploy them into the target subscription using – [Steps to deploy an API connection into a target subscription](bookmark://Deploy_APIconnection)
3. We now have our API Connections deployed in the target subscription (in a pre-existing resource group)

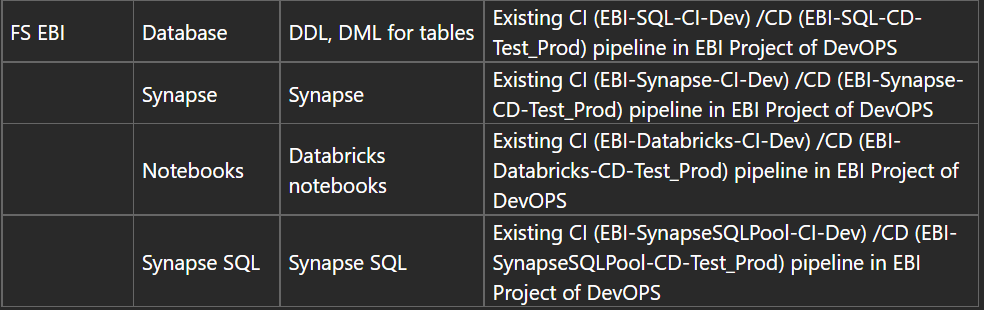
Step 4: Start Deployment of Logic apps

1. From the ‘List of resources for deployment’, we have the list of Logic apps that need to be deployed.
2. From the ‘Order of deployment of resources’, we have the order in which the logic apps need to be deployed.
3. Take the first logic app, follow the steps ( from 1a to 1h) mentioned in Option 1 of [How to deploy Logic App using ARM Template.](bookmark://Deploy_LogicApp) We now have a pipeline being created with an ARM Template Deployment Task.
4. Take the next logic app, follow steps from 1f to 1h. We have another ARM Template Deployment Task ready.
5. Repeat steps 1f to 1h for all the Logic apps in our list of Logic apps that need to be deployed. (Alternatively, right click on a task and select ‘Clone Task’, all we need to change is the ARM Template, Template Parameters and the Override Template Parameters)
6. Complete steps 1i, 1j, 2,3 and 4 in the link mentioned in C) above.
7. The final logic app to be deployed should be the master orchestration logic app (in our case, focus-master logic app)
8. The order of deployment is the order in which the tasks have been kept (Order of deployment is from top to bottom)



**How to deploy Azure Resources from EBI-DEV to EBI-QA**

Current scenario for deployment within EBI environment. (from EBI-DEV to EBI-Test\_Prod) is mentioned below for Azure Synapse Analytics (Synapse), Azure Databricks (Notebooks) and Synapse SQL :



Strategy for Deployment

We will set up a similar strategy of deployment as shown above for EBI-DEV to EBI-QA deployment. We will setup Azure DevOps CI/CD Pipelines for all the resources present in EBI-DEV Environment.

(That means that if there is any change in the source, the deployment will take place in the target automatically via the CI/CD pipeline that we create)

List of Resources:

Based on the target state architecture (reference is Slide 20 from [FS-Pilot Phase-Deck.pptx](https://capgemininar.sharepoint.com/:p:/r/sites/FourSeasonsTeamSite/Shared%20Documents/Data%20Program%20Delivery/Migration%20Stream/Presentation/FS-Pilot%20Phase-Deck.pptx?d=w2795a1000eb445e6ad7e1e1366e2271c&csf=1&web=1&e=sD67tc)), we will have the following resources in our EBI-DEV environment for development phase of Gen1-Gen2 Migration.

1. Function apps
2. ADF
3. Azure SQL Database/Synapse SQL
4. Azure Synapse Analytics
5. Logic apps
6. ADLS Gen 2
7. Azure Key Vault
8. Dedicated SQL Pool (formerly Azure SQL Data Warehouse)
9. API Connections

Order of Deployment:

As and when certain development has been completed during a sprint, we will want to deploy that into the EBI-QA Environment.

The general order of deployment will be:

1. Function apps
2. Dedicated SQL Pool
3. Synapse SQL
4. Azure Synapse Analytics
5. ADF
6. API Connections
7. Logic apps

Pre-requisites for deployment into EBI-QA Subscription:

1. ADLS Gen 2 Storage account needs to be created in the EBI-QA Subscription.
2. Azure Key Vault needs to be created in the EBI-QA Subscription.

Keys need to be created in Azure Key Vault in EBI-QA Subscription – these keys are referenced at the time of ADF Deployment and are necessary for Linked Service creation of the following resources:

* 1. Function apps
  2. Azure SQL Database/Synapse SQL
  3. Azure Synapse Analytics
  4. ADLS Gen 2
  5. Azure Key Vault
  6. Dedicated SQL Pool (formerly Azure SQL Data Warehouse)

1. Linked Services need to be created in EBI-QA Subscription for the following resources:
   1. Function apps
   2. ADF
   3. Azure SQL Database/Synapse SQL
   4. Azure Synapse Analytics
   5. Logic apps
   6. ADLS Gen 2
   7. Azure Key Vault
   8. Dedicated SQL Pool (formerly Azure SQL Data Warehouse)

These Linked Service names will be referenced at the time of deployment. (Step number 1 h vii from ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’ )

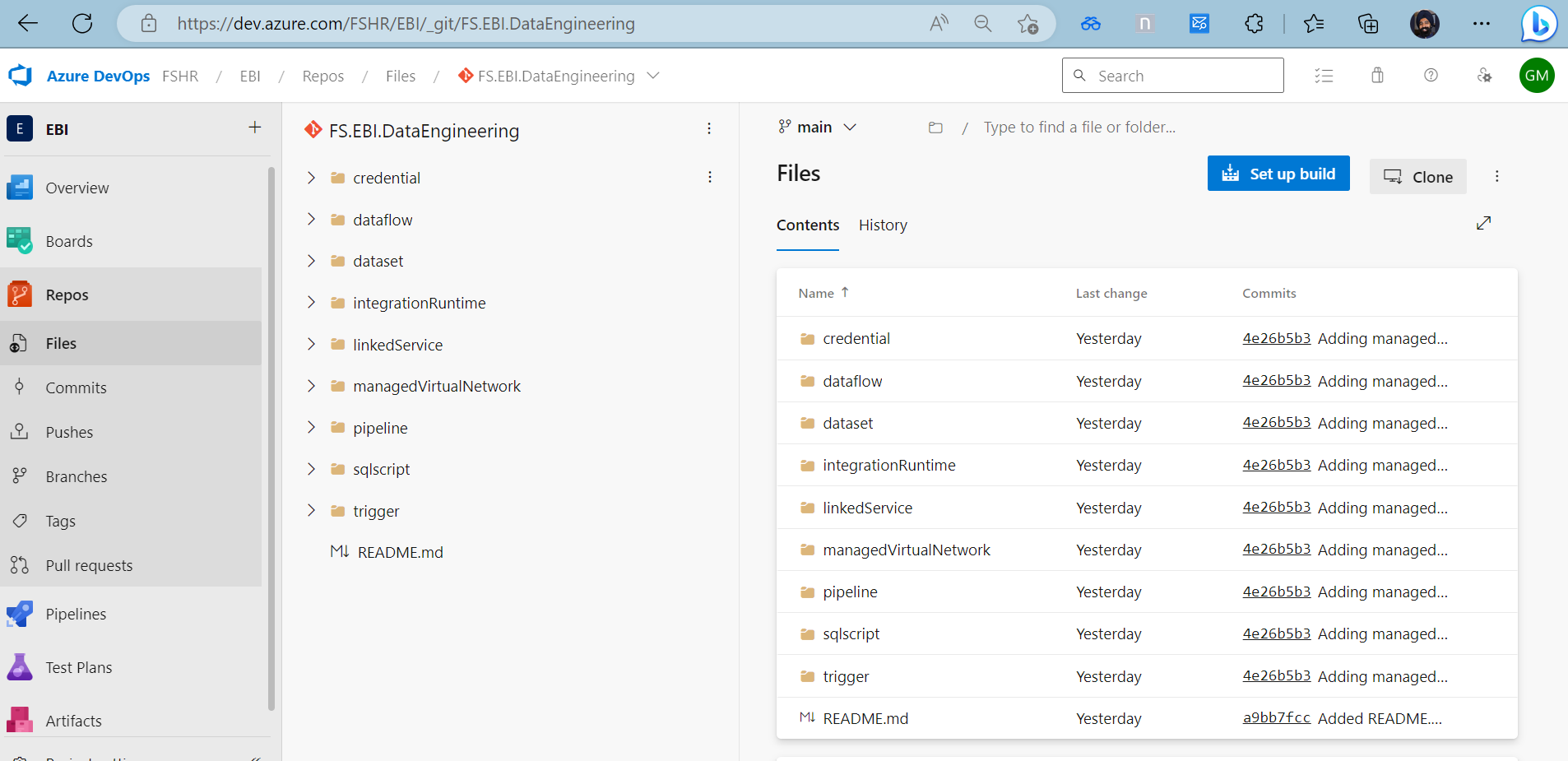
1. ADLS Gen 2 Deployment: Create an ADF Pipeline to migrate folders from ADLS Gen 2 in EBI-DEV to ADLS Gen 2 in EBI-QA Subscription.

At the time of ADF deployment, we need to provide the new folder locations present in ADLS Gen 2. (Step number 1 h vii from ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’ )

* + 1. Go to adf-phx-migr-qa Data factory in FS.EBI.QA.Integration Resource Group in EBI-QA Subscription.
    2. Under Pipeline, click on the “...” and select ‘New Pipeline’. Name your pipeline.
    3. Under ‘Move and Transform’ drag and drop ‘Copy data’ onto the canvas.
    4. In the Copy Activity, under ‘General’, name the Copy Activity appropriately.
    5. Under ‘Source’, select ‘Source dataset’ and under ‘Sink’, select ‘Sink dataset’.
    6. To create Source/Sink dataset, click on ‘+New’, select Azure Data Lake Gen 2 for Source and Azure Data Lake Gen 2 for Sink.
    7. Select the format type of data
    8. Select the Linked Services for ADLS Gen 2 in EBI-DEV and ADLS Gen 2 in EBI-QA. They have been created in Step 3a.
    9. We can provide the folder path for the source and sink.
    10. Repeat steps (iii) to (ix) above for migrating other folders and files from ADLS Gen 2 EBI-DEV to ADLS Gen 2 EBI-QA.

We will schedule the pipeline to run regularly depending on the frequency of files getting added into the ADLS Gen 2 Storage Account in EBI-DEV Subscription.

1. Confirm ARM Template Location of Azure Resources in Azure DevOps Repo: In order to create a CI/CD Pipeline for an Azure Resource using the steps mentioned in ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’, we need to confirm the ARM Template and ARM Template parameters file for the resource are in the ‘main’ branch of the ‘Repos/Files’ section in the Azure DevOps Project ‘EBI’. Refer to the screenshot below.

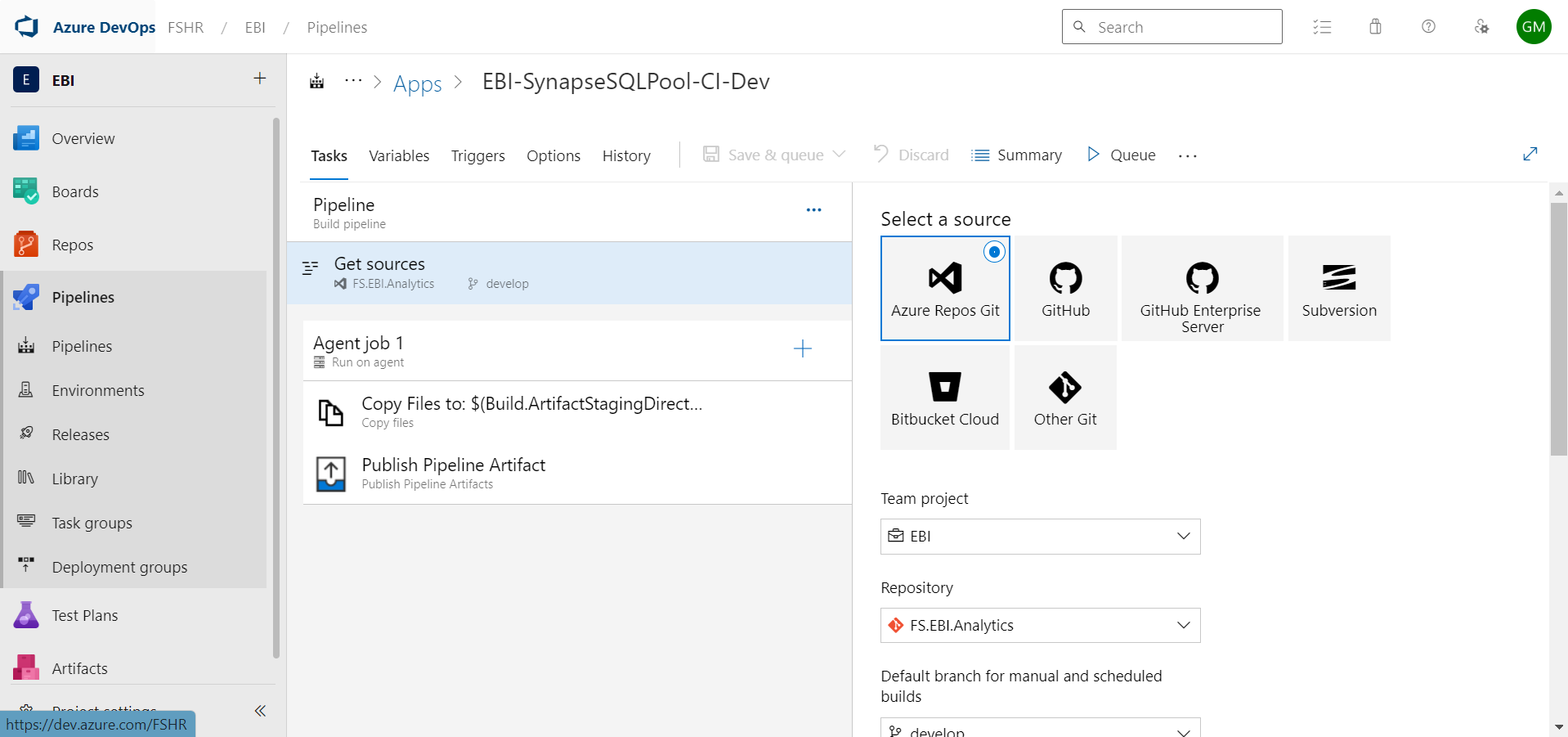


Start Deployment:

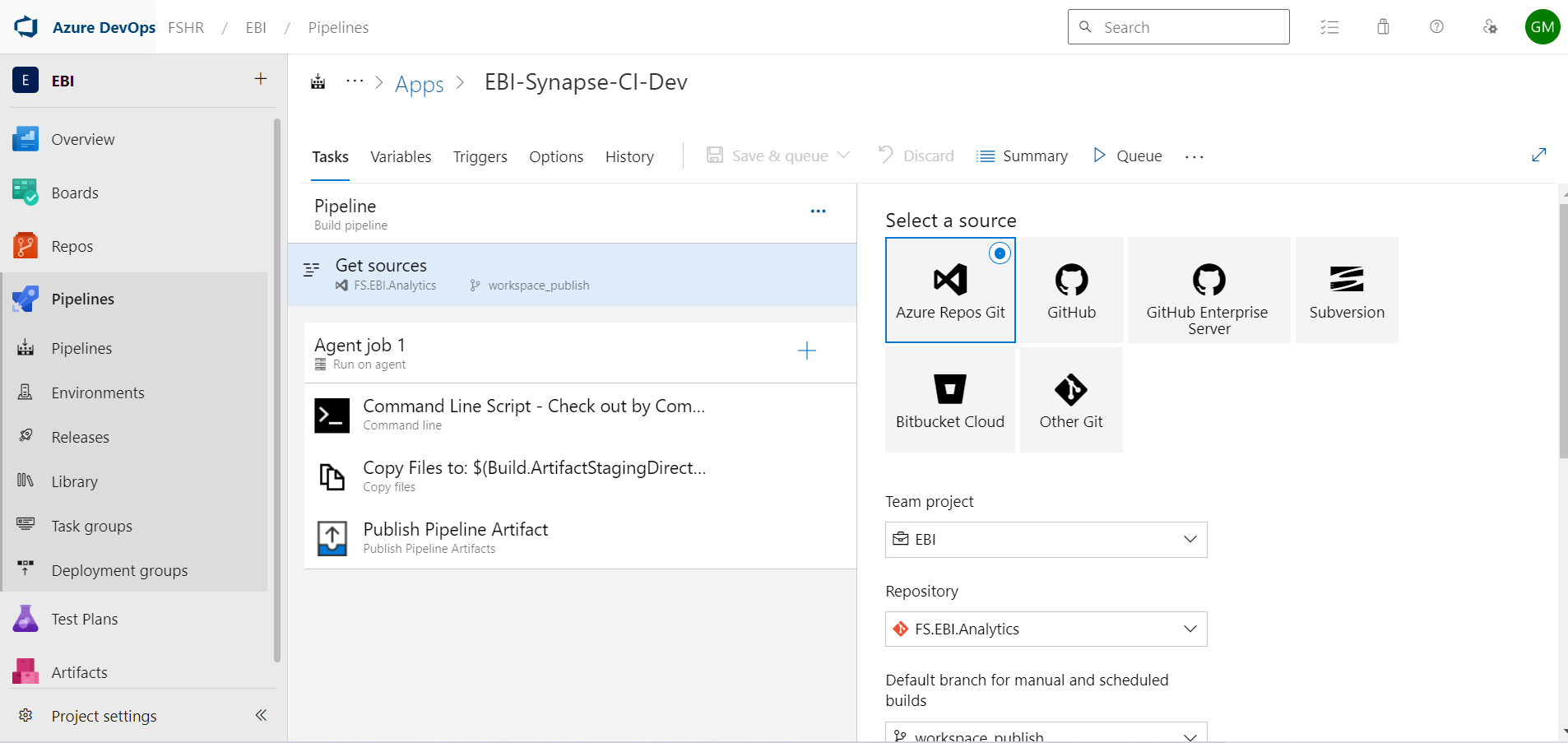
Create CI/CD Pipeline and Release:

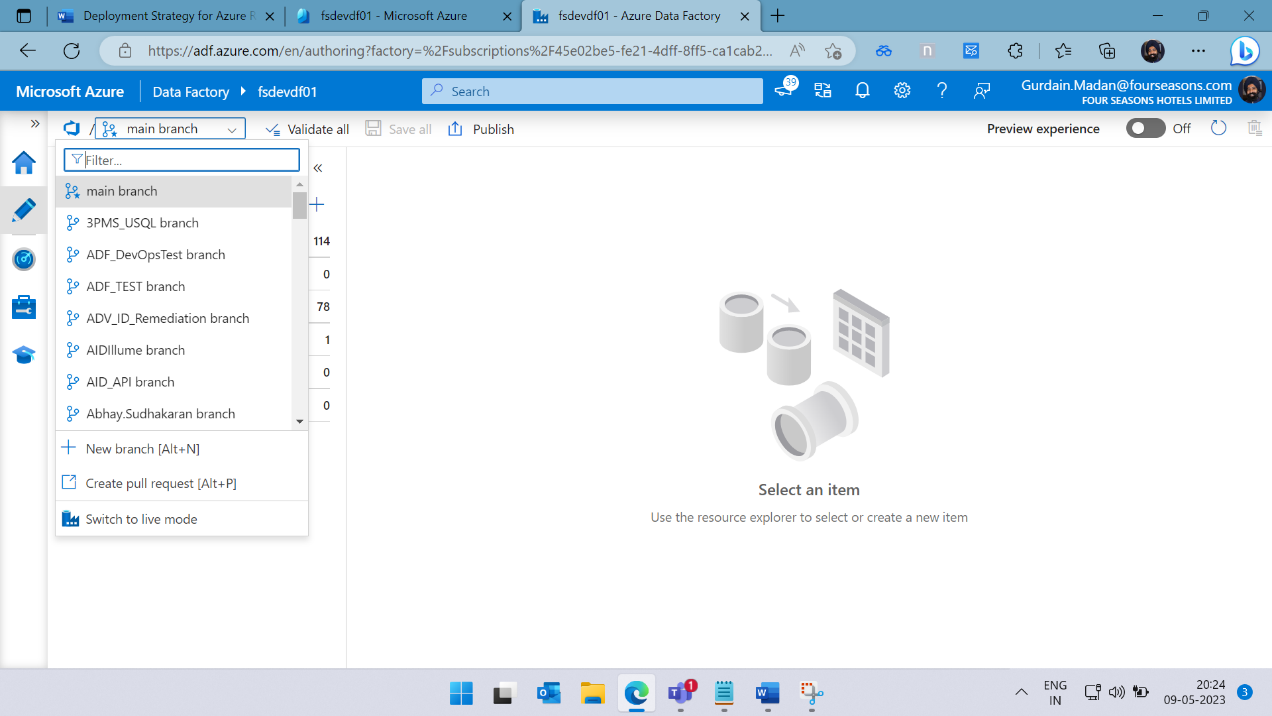
1. Function apps:
   1. Start following the steps mentioned in ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’
   2. Step 1d – Select ‘main’ branch as it will have the updated ARM Template and ARM Template parameters
   3. Complete steps till 1h. In 1 h vii, make sure we refer to the new names, new linked services, new keys, etc.
   4. One task has been completed under an Agent job for one Function app
   5. Repeat steps 1f, 1g and 1h for the other Function apps.
   6. Complete steps 1i, 1j, 2,3 and 4.
2. Dedicated SQL Pool and Azure SQL Database (or Synapse SQL):
   1. Start following the steps mentioned in ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’
   2. Step 1d – Select ‘main’ branch as it will have the updated files (Pipeline Artifacts)
   3. Complete remaining steps
   4. Refer to EBI-SynapseSQLPool-CI-Dev Pipeline in FSHB/EBI/Pipelines location.

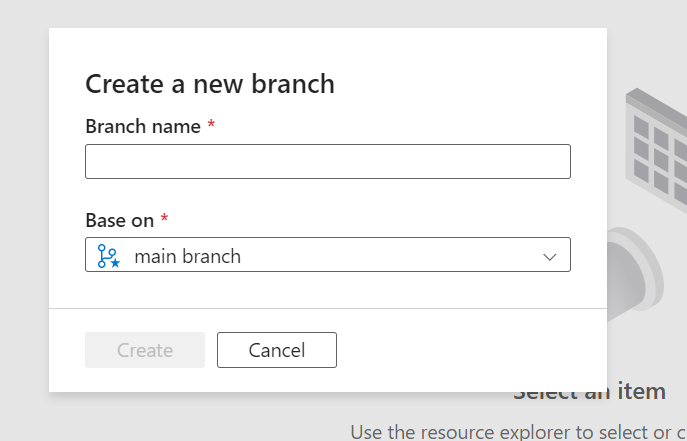
Reference:



1. Azure Synapse Analytics:
   1. Start following the steps mentioned in ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’
   2. Step 1d – Select ‘main’ branch as it will have the updated files (Pipeline Artifacts)
   3. Complete remaining steps
   4. Refer to EBI-Synapse-CI-Dev Pipeline in FSHB/EBI/Pipelines location.

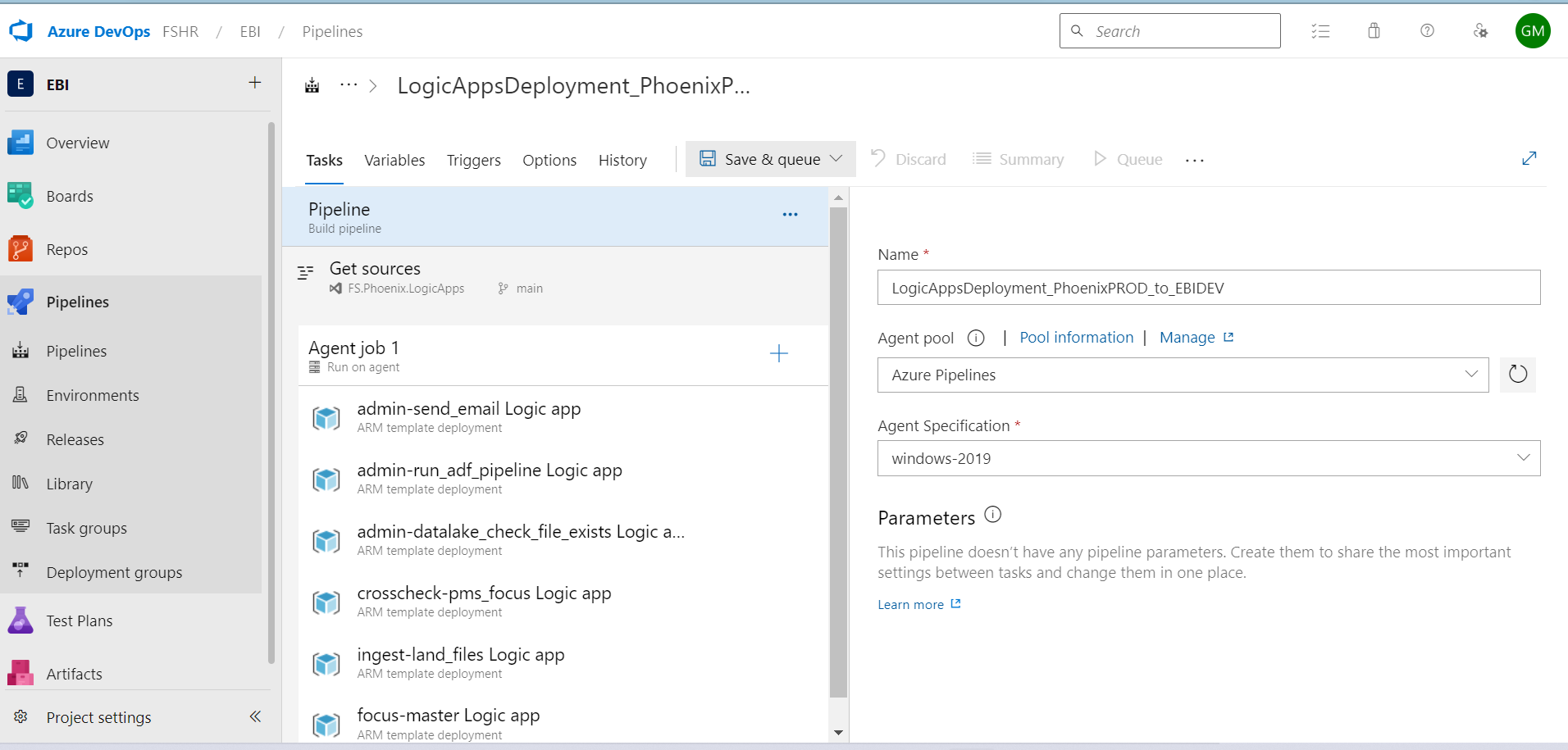


1. ADF:
   1. Move the pipelines that need to be deployed in the EBI-QA Subscription into the ‘main’ branch.
   2. Follow the steps mentioned in Option 1 of – [How to deploy a Data Factory to a Target subscription](bookmark://Deploy_ADF) . We can start from Step 2, since a code repository is already created.
   3. Step number 7.h.vii is important – This is where we need to make changes as per desired deployment requirements. For example, update the values for the services that have already been created in the EBI-QA Subscription, for example, key vault base URL, linked service names, etc.
   4. Step number 7.i and 9.a are for Continuous Integration and Continuous Deployment. Depending on the requirement, we can enable or disable these options.
   5. We now have our source ADF deployed in the target subscription.
   6. Connect the ADF in our target subscription to Azure DevOps – Refer to Step number 1 ‘Set up code repository’ in Option 1 of [How to deploy a Data Factory to a Target subscription.](bookmark://Deploy_ADF)
   7. Create a new branch – Click on ‘+ New branch [Alt + N] ‘ in the dropdown shown in the screenshot. Call it development branch. 
   8. The new branch will be the development branch where all the development and testing will be carried out.



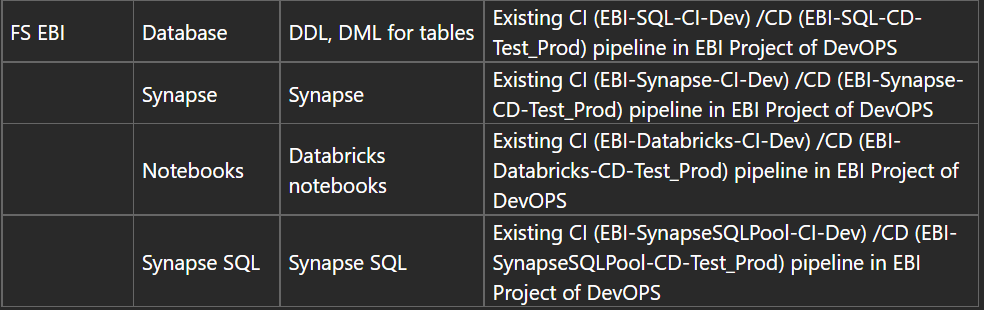
* 1. As development and testing continues, Edit the main branch to have only those pipelines that are ready to be deployed to EBI-PROD Environment for testing.
  2. The main branch is our feature branch.

1. API Connections:
   1. From the ‘List of resources for deployment’, we have the list of API connections that need to be deployed.
   2. Take one API connection at a time, deploy them into the target subscription using – [General Steps to set up CI/CD Process](bookmark://Steps_CICD)
   3. Step number 1 h 6 and 1 h 7 are important steps and we need to know in which Repo the ARM Template and ARM Template Parameters for the API connections are stored.
   4. We now have our CI/CD process set up to deploy API Connections into the target subscription.
2. Logic apps:
   1. From the ‘List of resources for deployment’, we have the list of Logic apps that need to be deployed.
   2. From the ‘Order of deployment of resources’, we have the order in which the logic apps need to be deployed.
   3. Take the first logic app, follow the steps ( from 1a to 1h) mentioned in Option 1 of [How to deploy Logic App using ARM Template.](bookmark://Deploy_LogicApp) We now have a pipeline being created with an ARM Template Deployment Task.
   4. Take the next logic app, follow steps from 1f to 1h. We have another ARM Template Deployment Task ready.
   5. Repeat steps 1f to 1h for all the Logic apps in our list of Logic apps that need to be deployed. (Alternatively, right click on a task and select ‘Clone Task’, all we need to change is the ARM Template, Template Parameters and the Override Template Parameters)
   6. Complete steps 1i, 1j, 2,3 and 4 in the link mentioned in step number ‘c’ above.
   7. The order of deployment is the order in which the tasks have been kept (which is from top to bottom)



**How to deploy Azure Resources from EBI-QA to EBI-PROD**

Current scenario for deployment within EBI environment. (from EBI-DEV to EBI-Test\_Prod) is mentioned below for Azure Synapse Analytics (Synapse), Azure Databricks (Notebooks) and Synapse SQL :



Strategy for Deployment

We will set up a similar strategy of deployment as shown above for EBI-QA to EBI-PROD deployment., the only difference being that this will be a bulk deployment. We will setup Azure DevOps CI/CD Pipelines to deploy on a bulk level, for example, the entire focus-master Logic app along with all the dependent logic apps, function apps, ADF Pipelines and other resources will be deployed at one go.

If we need to deploy multiple Logic apps, then we need to deploy one Logic app (along with all dependent resources) at a time.

List of Resources:

Based on the target state architecture (reference is Slide 20 from [FS-Pilot Phase-Deck.pptx](https://capgemininar.sharepoint.com/:p:/r/sites/FourSeasonsTeamSite/Shared%20Documents/Data%20Program%20Delivery/Migration%20Stream/Presentation/FS-Pilot%20Phase-Deck.pptx?d=w2795a1000eb445e6ad7e1e1366e2271c&csf=1&web=1&e=sD67tc)), we will have the following resources in our EBI-QA environment.

1. Function apps
2. ADF
3. Azure SQL Database/Synapse SQL
4. Azure Synapse Analytics
5. Logic apps
6. ADLS Gen 2
7. Azure Key Vault
8. Dedicated SQL Pool (formerly Azure SQL Data Warehouse)
9. API Connections

Order of Deployment:

The order of deployment will be from least dependency to greatest dependency. For focus- master logic app, we will decide the order of dependency and hence the order of deployment based on the focus-master ST (Source to Target) Linkage Document ([Focus-](https://capgemininar.sharepoint.com/:x:/r/sites/FourSeasonsTeamSite/Shared%20Documents/Data%20Program%20Delivery/Migration%20Stream/Pilot%20Analysis/Focus-master%20ST%20Linkage.xlsx?d=w5139efa310f84d8088d4baaf2316086b&csf=1&web=1&e=KJZfcd) master ST Linkage.xlsx). , the general order will be as follows:

1. Function apps
2. Synapse SQL and Dedicated SQL Pool
3. Azure Synapse Analytics
4. ADF
5. API Connections
6. Logic apps

Note: Within each of the services mentioned above, the resource that has least dependency will be deployed first. For example, within focus-master Logic app, multiple dependent Logic apps exist. First, we need to deploy the internal dependent Logic apps, then we need to deploy the focus-master Logic app.

Pre-requisites for deployment into EBI-PROD Subscription:

1. ADLS Gen 2 Storage account needs to be created in the EBI-PROD Subscription.
2. Azure Key Vault needs to be created in the EBI-PROD Subscription.

Keys need to be created in Azure Key Vault in EBI-PROD Subscription – these keys are referenced at the time of ADF Deployment and are necessary for Linked Service creation of the following resources:

* 1. Function apps
  2. Azure SQL Database/Synapse SQL
  3. Azure Synapse Analytics
  4. ADLS Gen 2
  5. Azure Key Vault
  6. Dedicated SQL Pool (formerly Azure SQL Data Warehouse)

1. Linked Services need to be created in EBI-PROD Subscription for the following resources:
   1. Function apps
   2. ADF
   3. Azure SQL Database/Synapse SQL
   4. Azure Synapse Analytics
   5. Logic apps
   6. ADLS Gen 2
   7. Azure Key Vault
   8. Dedicated SQL Pool (formerly Azure SQL Data Warehouse)

These Linked Service names will be referenced at the time of deployment. (Step number 1 h vii from ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’ )

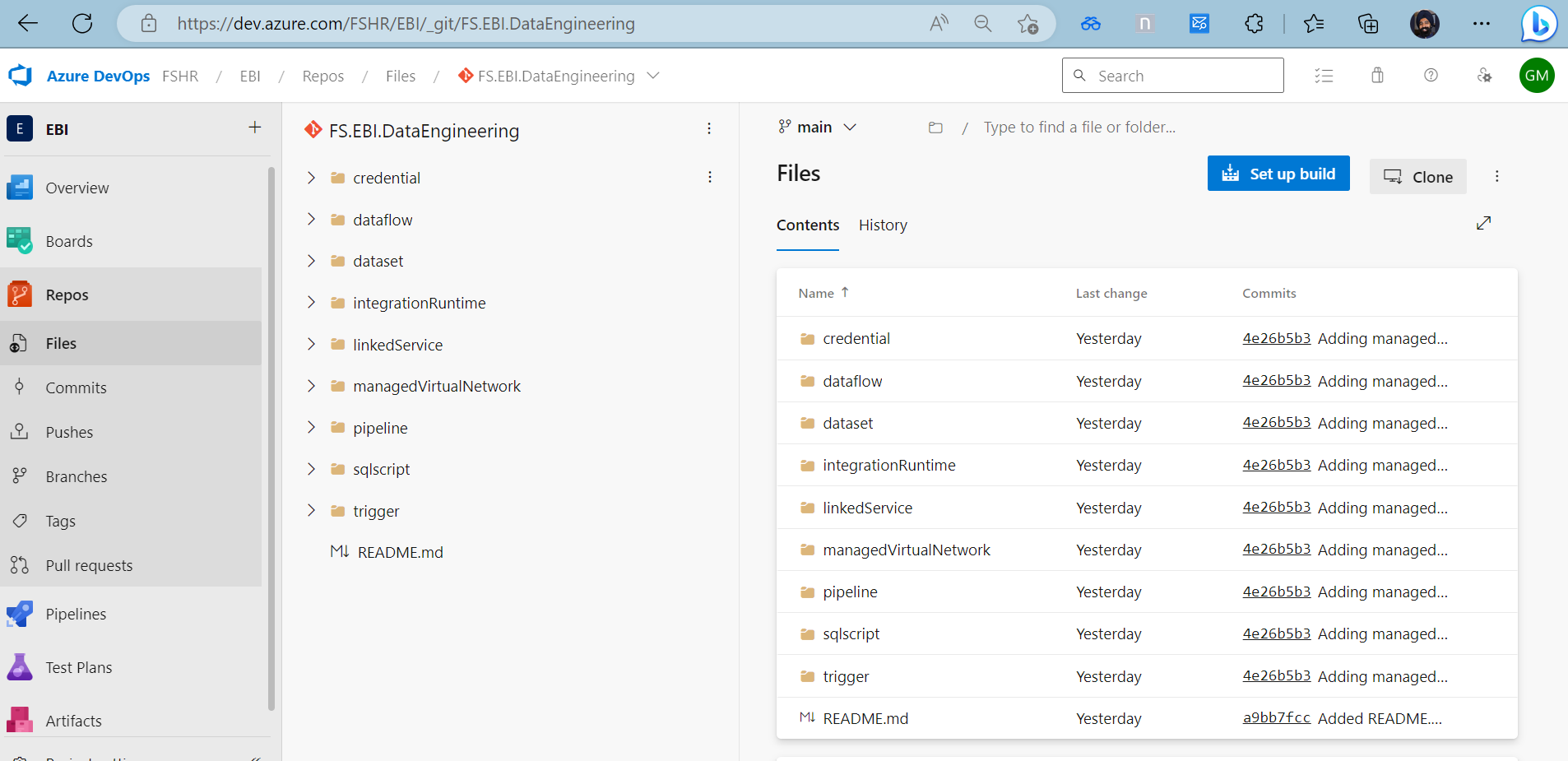
1. ADLS Gen 2 Deployment: Create an ADF Pipeline to migrate folders from ADLS Gen 2 in EBI-QA to ADLS Gen 2 in EBI-PROD Subscription.

At the time of ADF deployment, we need to provide the new folder locations present in ADLS Gen 2. (Step number 1 h vii from ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’ )

* + 1. Go to adf-phx-migr-prod Data factory in FS.EBI.PROD.Integration Resource Group in EBI-PROD Subscription.
    2. Under Pipeline, click on the “...” and select ‘New Pipeline’. Name your pipeline.
    3. Under ‘Move and Transform’ drag and drop ‘Copy data’ onto the canvas.
    4. In the Copy Activity, under ‘General’, name the Copy Activity appropriately.
    5. Under ‘Source’, select ‘Source dataset’ and under ‘Sink’, select ‘Sink dataset’.
    6. To create Source/Sink dataset, click on ‘+New’, select Azure Data Lake Gen 2 for Source and Azure Data Lake Gen 2 for Sink.
    7. Select the format type of data
    8. Select the Linked Services for ADLS Gen 2 in EBI-QA and ADLS Gen 2 in EBI-PROD. They have been created in Step 3a.
    9. We can provide the folder path for the source and sink.
    10. Repeat steps (iii) to (ix) above for migrating other folders and files from ADLS Gen 2 EBI-QA to ADLS Gen 2 EBI-PROD.

We will schedule the pipeline to run regularly depending on the frequency of files getting added into the ADLS Gen 2 Storage Account in EBI-QA Subscription.

1. Confirm ARM Template Location of Azure Resources in Azure DevOps Repo: In order to create a CI/CD Pipeline for an Azure Resource using the steps mentioned in ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’, we need to confirm the ARM Template and ARM Template parameters file for the resource are in the ‘main’ branch of the ‘Repos/Files’ section in the Azure DevOps Project ‘EBI’. Refer to the screenshot below.

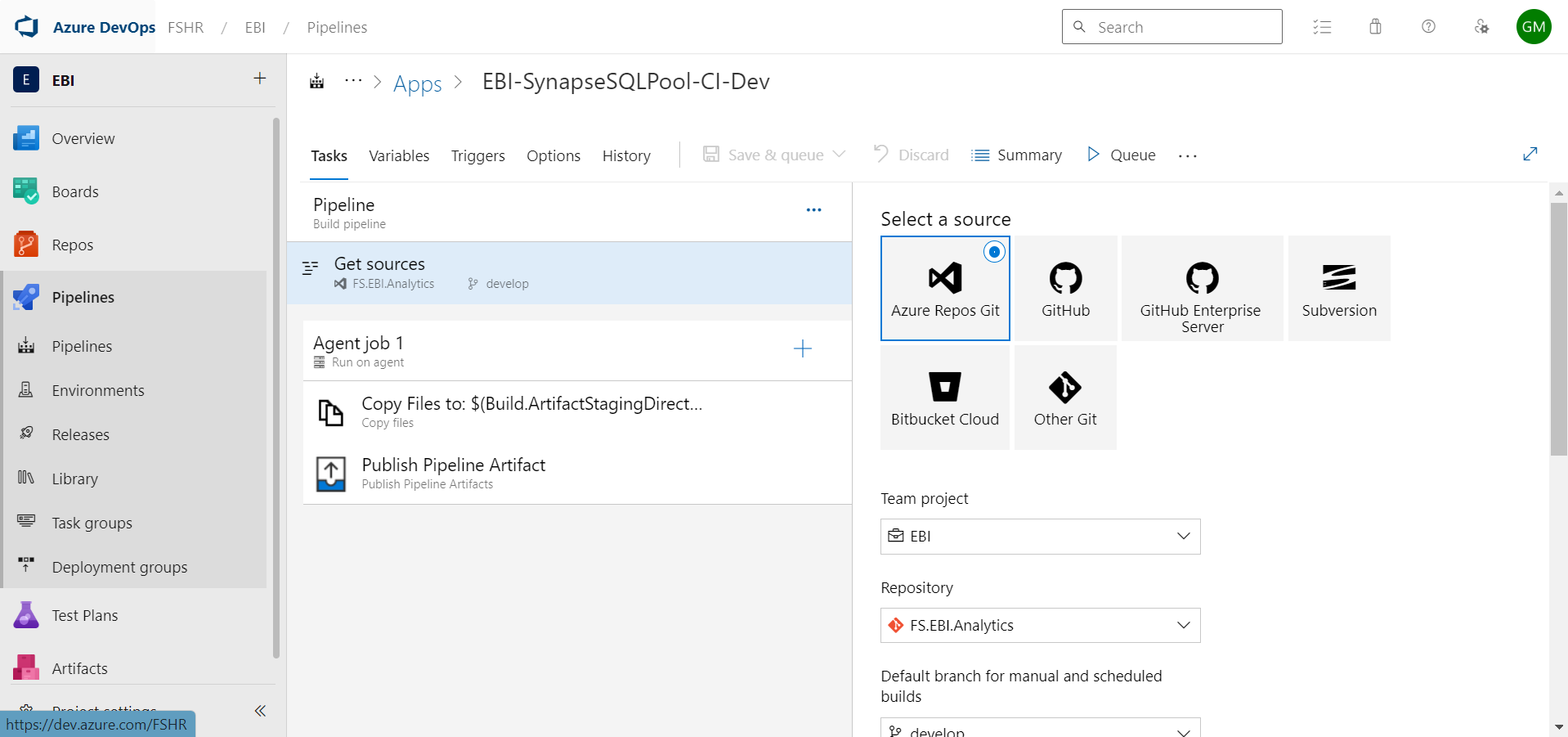


Start Deployment:

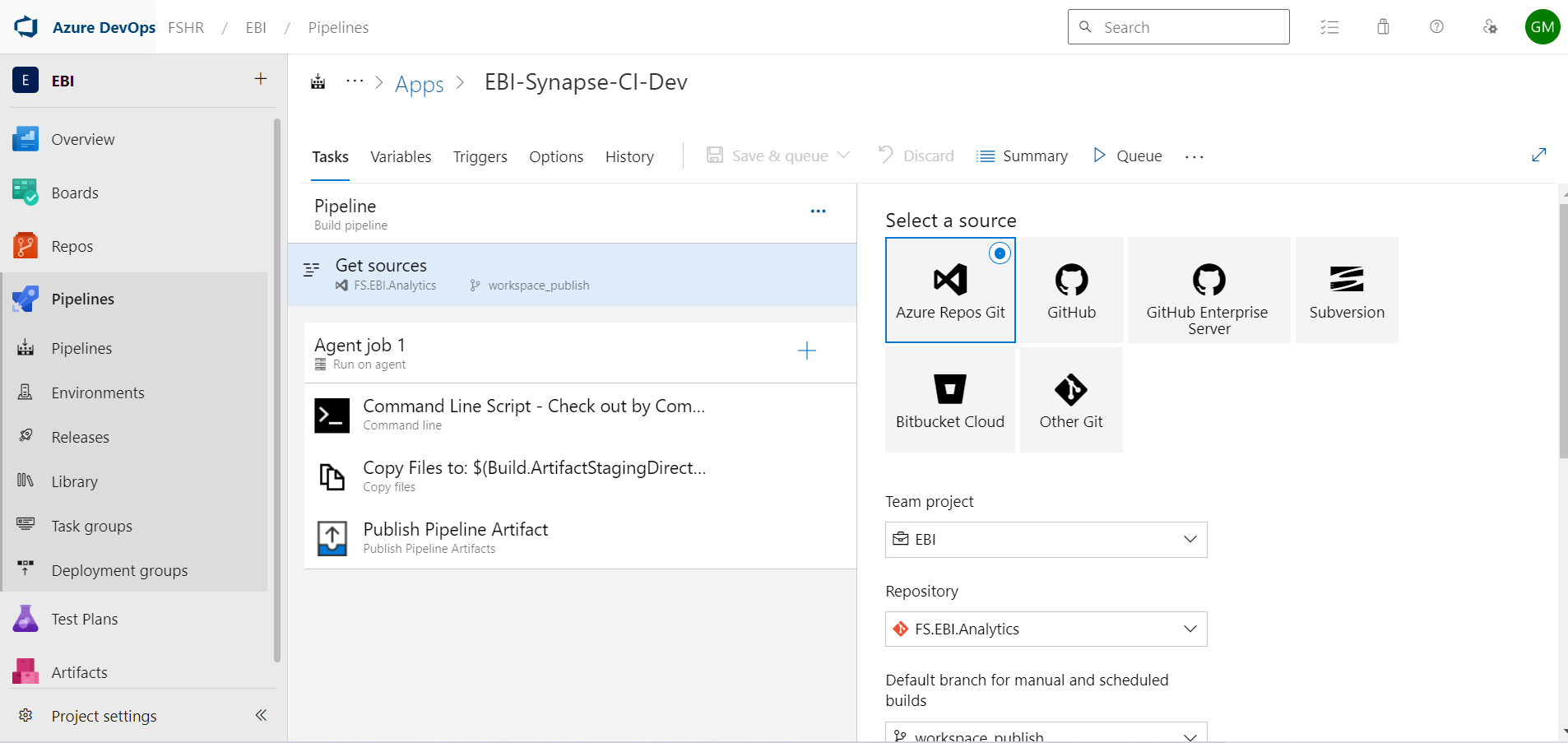
Create CI/CD Pipeline and Release:

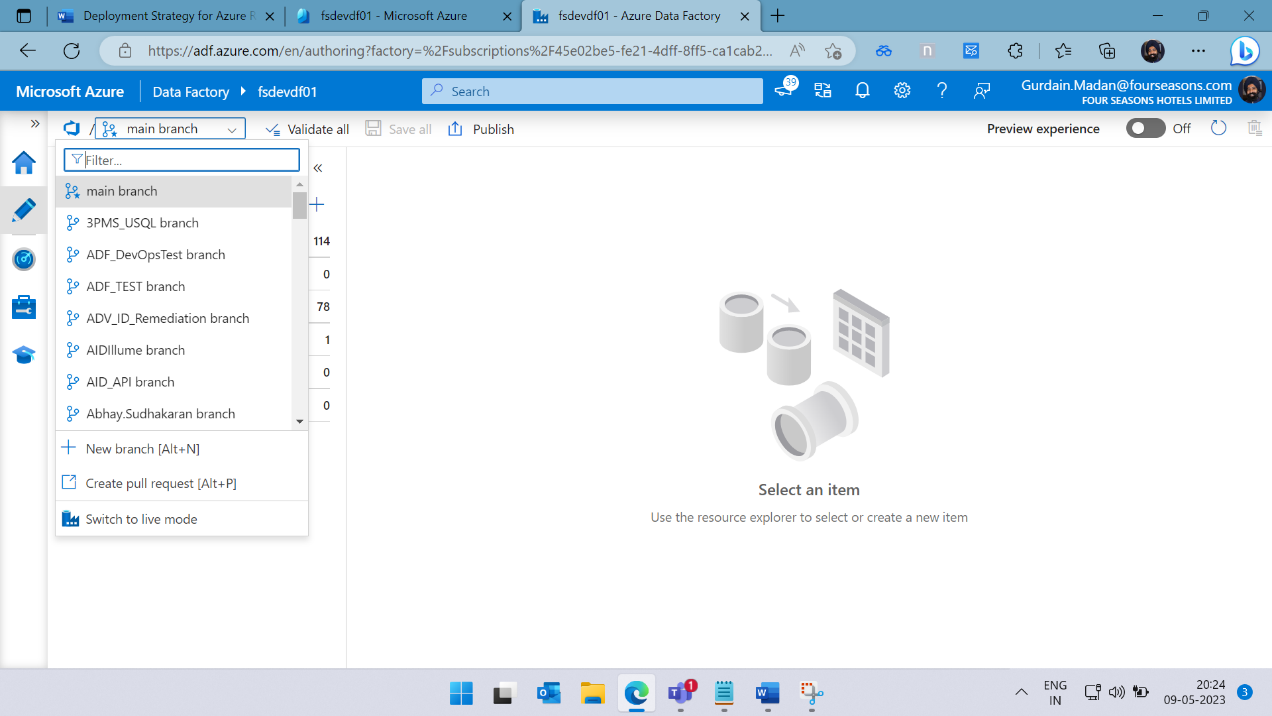
1. Function apps:
   1. Start following the steps mentioned in ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’
   2. Step 1d – Select ‘main’ branch as it will have the updated ARM Template and ARM Template parameters
   3. Complete steps till 1h. In 1 h vii, make sure we refer to the new names, new linked services, new keys, etc.
   4. One task has been completed under an Agent job for one Function app
   5. Repeat steps 1f, 1g and 1h for the other Function apps.
   6. Complete steps 1i, 1j, 2,3 and 4.
2. Dedicated SQL Pool and Azure SQL Database (or Synapse SQL):
   1. Start following the steps mentioned in ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’
   2. Step 1d – Select ‘main’ branch as it will have the updated files (Pipeline Artifacts)
   3. Complete remaining steps
   4. Refer to EBI-SynapseSQLPool-CI-Dev Pipeline in FSHB/EBI/Pipelines location.

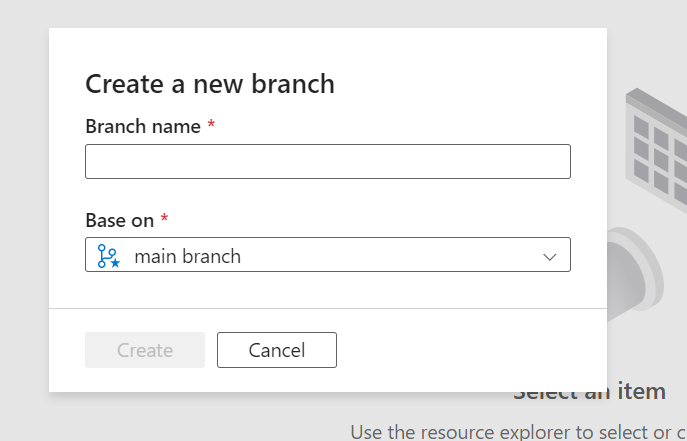
Reference:



1. Azure Synapse Analytics:
   1. Start following the steps mentioned in ‘[General steps to set up a CI/CD Process](bookmark://Steps_CICD)’
   2. Step 1d – Select ‘main’ branch as it will have the updated files (Pipeline Artifacts)
   3. Complete remaining steps
   4. Refer to EBI-Synapse-CI-Dev Pipeline in FSHB/EBI/Pipelines location.

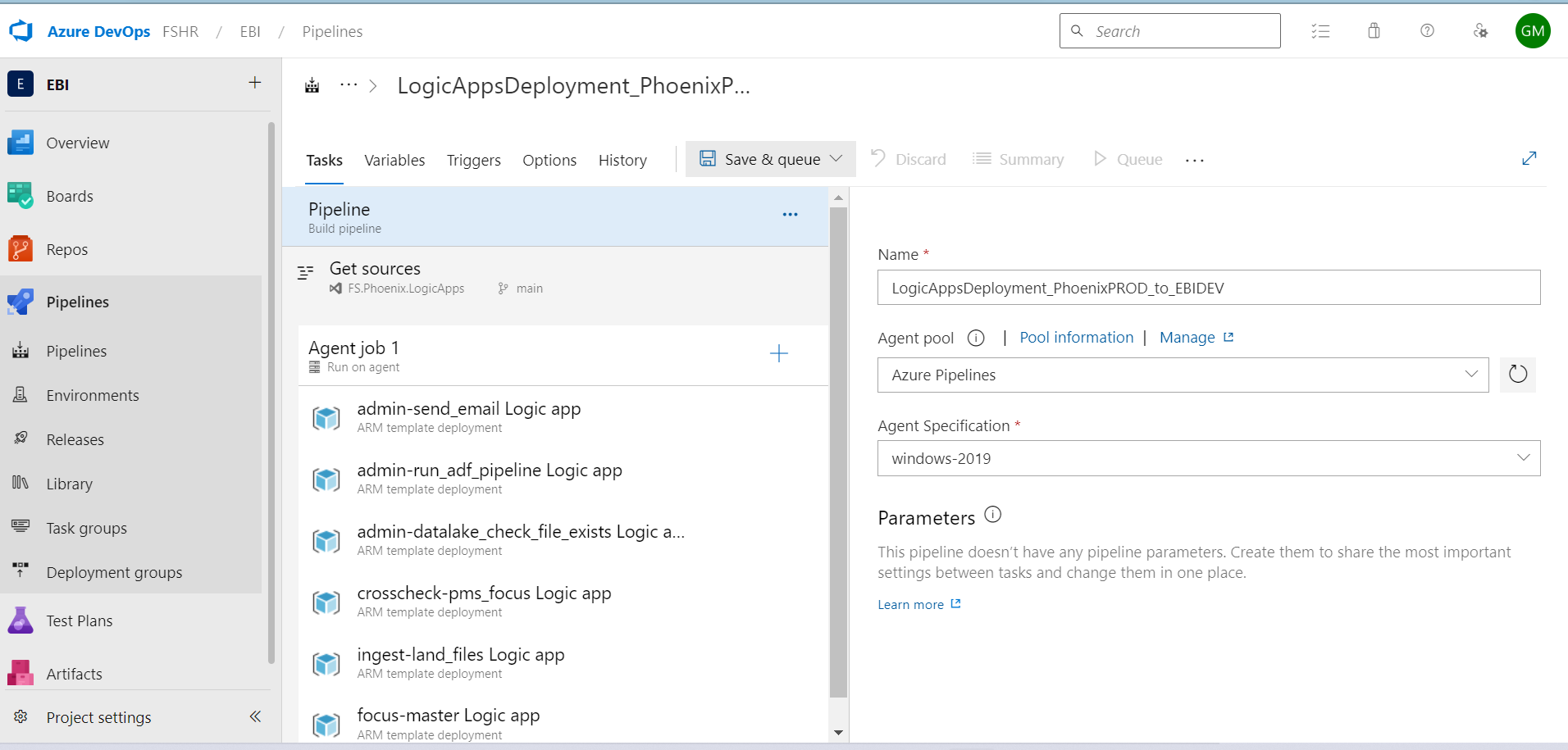


1. ADF:
   1. Move the pipelines that need to be deployed in the EBI-QA Subscription into the ‘main’ branch.
   2. Follow the steps mentioned in Option 1 of – [How to deploy a Data Factory to a Target subscription](bookmark://Deploy_ADF) . We can start from Step 2, since a code repository is already created.
   3. Step number 7.h.vii is important – This is where we need to make changes as per desired deployment requirements. For example, update the values for the services that have already been created in the EBI-QA Subscription, for example, key vault base URL, linked service names, etc.
   4. Step number 7.i and 9.a are for Continuous Integration and Continuous Deployment. Depending on the requirement, we can enable or disable these options.
   5. We now have our source ADF deployed in the target subscription.
   6. Connect the ADF in our target subscription to Azure DevOps – Refer to Step number 1 ‘Set up code repository’ in Option 1 of [How to deploy a Data Factory to a Target subscription.](bookmark://Deploy_ADF)
   7. Create a new branch – Click on ‘+ New branch [Alt + N] ‘ in the dropdown shown in the screenshot. Call it development branch. 
   8. The new branch will be the development branch where all the development and testing will be carried out.



* 1. As development and testing continues, Edit the main branch to have only those pipelines that are ready to be deployed to EBI-PROD Environment for testing.
  2. The main branch is our feature branch.

1. API Connections:
   1. From the ‘List of resources for deployment’, we have the list of API connections that need to be deployed.
   2. Take one API connection at a time, deploy them into the target subscription using – [General Steps to set up CI/CD Process](bookmark://Steps_CICD)
   3. Step number 1 h 6 and 1 h 7 are important steps and we need to know in which Repo the ARM Template and ARM Template Parameters for the API connections are stored.
   4. We now have our CI/CD process set up to deploy API Connections into the target subscription.
2. Logic apps:
   1. From the ‘List of resources for deployment’, we have the list of Logic apps that need to be deployed.
   2. From the ‘Order of deployment of resources’, we have the order in which the logic apps need to be deployed.
   3. Take the first logic app, follow the steps ( from 1a to 1h) mentioned in Option 1 of [How to deploy Logic App using ARM Template.](bookmark://Deploy_LogicApp) We now have a pipeline being created with an ARM Template Deployment Task.
   4. Take the next logic app, follow steps from 1f to 1h. We have another ARM Template Deployment Task ready.
   5. Repeat steps 1f to 1h for all the Logic apps in our list of Logic apps that need to be deployed. (Alternatively, right click on a task and select ‘Clone Task’, all we need to change is the ARM Template, Template Parameters and the Override Template Parameters)
   6. Complete steps 1i, 1j, 2,3 and 4 in the link mentioned in step number ‘c’ above.
   7. The order of deployment is the order in which the tasks have been kept (which is from top to bottom)



General steps to set up a CI/CD Process (Continuous Integration/Continuous Deployment) are below:

1. Select Repos/Pipelines - Create a Pipeline by clicking on ‘New pipeline’
   1. At the bottom of the screen, click on ‘Use the classic editor’
   2. Select a source as ‘Azure Repos Git’
   3. Select Team project, Repository
   4. Select ‘main’ branch as the Default branch for manual and scheduled builds, select Continue
   5. Select a template, click on (*or start with an)* ‘Empty job’
   6. Under Tasks tab, Click on the ‘+’ sign next to Agent job 1 (Add a task to Agent job 1)
   7. Search ‘arm’, Add ‘ARM template deployment’
   8. Configure settings for ARM template deployment
      1. Select Deployment scope
      2. Select Azure Resource Manager connection (having access to the selected deployment scope)
      3. Select the Azure subscription (target subscription for deployment)
      4. Select ‘Create or Update Resource Group’ in Action
      5. Select the ‘Resource Group’ into which you would like to deploy the resource, Specify location
      6. Select Template location as Linked artifact, Template and Template parameters using the ‘…’ on the right
      7. IMPORTANT STEP: The changes that we want to make for the new environment. For example, change the ‘value’ of factoryName or change the connection string for storage account.
   9. If we want this pipeline to be triggered every time a publish takes place, we need to go to Triggers (Next to Tasks and Variables), select ‘Enable continuous integration’
   10. Click on ‘Save & queue’
2. If everything is proper and in order, we will get a green tick next to the pipeline name which means deployment has successfully taken place
3. Create a Release in Pipelines/Releases - ‘Create release’
   1. Under Artifacts, add the pipeline that we created in Step number 7, Enable the Continuous deployment trigger (symbol on the artifact name)
   2. For Stages, start with an ‘Empty Job’, select Stage name, stage will be created
4. Our Continuous Integration/Continuous Deployment process is ready)

**Steps to deploy Function App into a target subscription**

(Reference: fsprodfuncpipelineintegration Function App in FS.Phoenix.Prod.Integration Resource Group in FS-Phoenix-PROD Subscription)

1. Create an Azure DevOps Pipeline to automatically export the Function app ARM Template onto the Azure DevOps Repository
2. Create an Azure DevOps Pipeline to automatically deploy the ARM Template into the target subscription

**Create an Azure DevOps Pipeline to automatically export the Function app ARM Template onto the Azure DevOps Repository**

To automatically export an ARM (Azure Resource Manager) template of a Function App into a repository in Azure DevOps, you can use Azure DevOps Pipelines and the Azure CLI. Here's a step-by-step guide on how to achieve this:

1. **Set up your Azure DevOps repository**: Create a new repository in Azure DevOps where you want to store the ARM template.

--> Created a repository with the name ‘FS.Phoenix.FunctionApps’ in FSHR/Phoenix.

1. **Configure Azure DevOps Pipeline**: Set up a pipeline in Azure DevOps to automatically export the ARM template. Use the classic editor to define the pipeline.

--> Created the pipeline in the EBI Project ([Steps:](bookmark://Deploy_LogicApp) Follow till 1.d in Option 1)

Select source as FS.Phoenix.FunctionApps

1. **Add a pipeline trigger**: Configure the pipeline to trigger automatically whenever there is a change in the Function App resources. You can set the trigger based on specific branches or paths.

To configure the pipeline trigger, follow these steps:

1. Open your Azure DevOps pipeline configuration.
2. Locate the section where you define the trigger for your pipeline. (**‘Triggers’ tab**) This section determines when the pipeline should be executed.
3. Add a trigger condition that matches the changes in the Function App resources. You can use the **paths** parameter to specify the path(s) to monitor for changes.

**Enabled continuous integration**

**Added Path filter as Include: ‘/ARM Templates/\*’**

This configuration will trigger the pipeline when there are changes in the folder of the repository, such as modifications to the Function App code.

1. Save your pipeline configuration.

With this trigger configuration, whenever there are changes in the specified paths (in this case, the **/ARM Templates** folder), the pipeline will automatically start executing, and the ARM template export process will be triggered.

Note: It's important to adjust the trigger conditions based on your specific repository structure and the location of your Function App resources to ensure that the pipeline triggers correctly.

1. **Add the Azure CLI task**: In your Azure DevOps pipeline, add a task to install the Azure CLI. This task will enable you to run Azure CLI commands within your pipeline.

--> Click the ‘+’ sign to add a task to Agent Job 1

--> Search Azure CLI

--> Add an inline script: az functionapp deployment source show --name <Function App Name> --resource-group <Resource Group Name> --output json > arm-template.json command to retrieve the ARM template JSON.

--> Replace **<Azure subscription>**, **<Function App Name>**, and **<Resource Group Name>** with the appropriate values.

1. **Publish the ARM template**: Add a task to publish the exported ARM template to your Azure DevOps repository. Use the **Publish Build Artifacts** task to copy the generated ARM template to the artifact staging directory.

task: PublishBuildArtifacts@1 displayName: 'Publish ARM Template' inputs: PathtoPublish: '$(Build.SourcesDirectory)' ArtifactName: 'ARM Template'

1. **Commit and push the changes**: Add a task to commit and push the changes to your Azure DevOps repository. You can use the **Command Line** task to achieve this.

- script: | git config --global user.email "[youremail@example.com](mailto:youremail@example.com)" git config --global [user.name](http://user.name/) "Your Name" git add . git commit -m "Auto-exported ARM template" git push displayName: 'Commit and Push Changes'

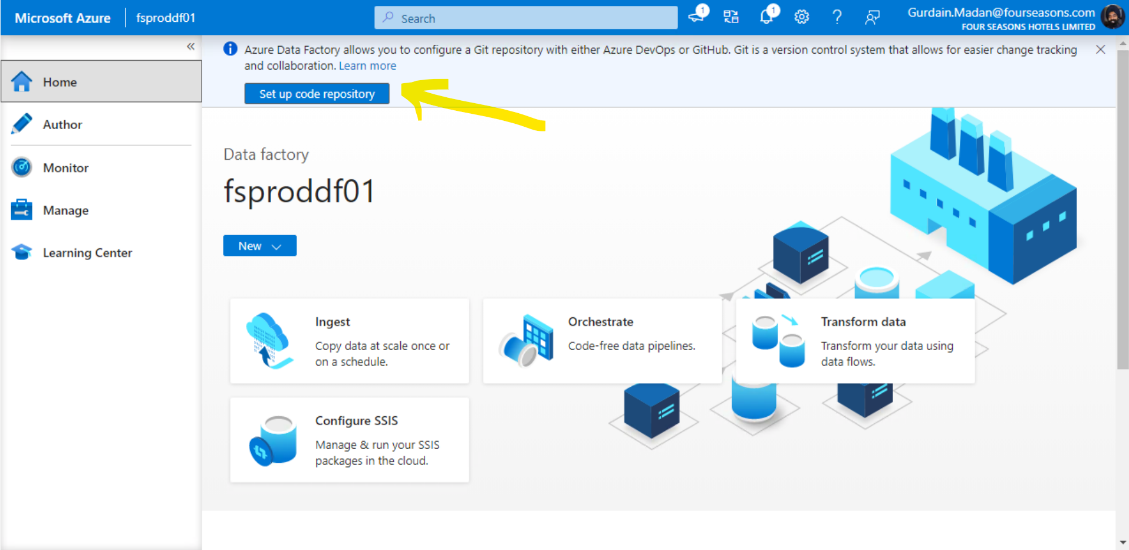
1. **Save and run the pipeline**: Save your pipeline configuration and trigger a manual run/Queue to test the process. Verify that the ARM template is exported and pushed to the repository in Azure DevOps.

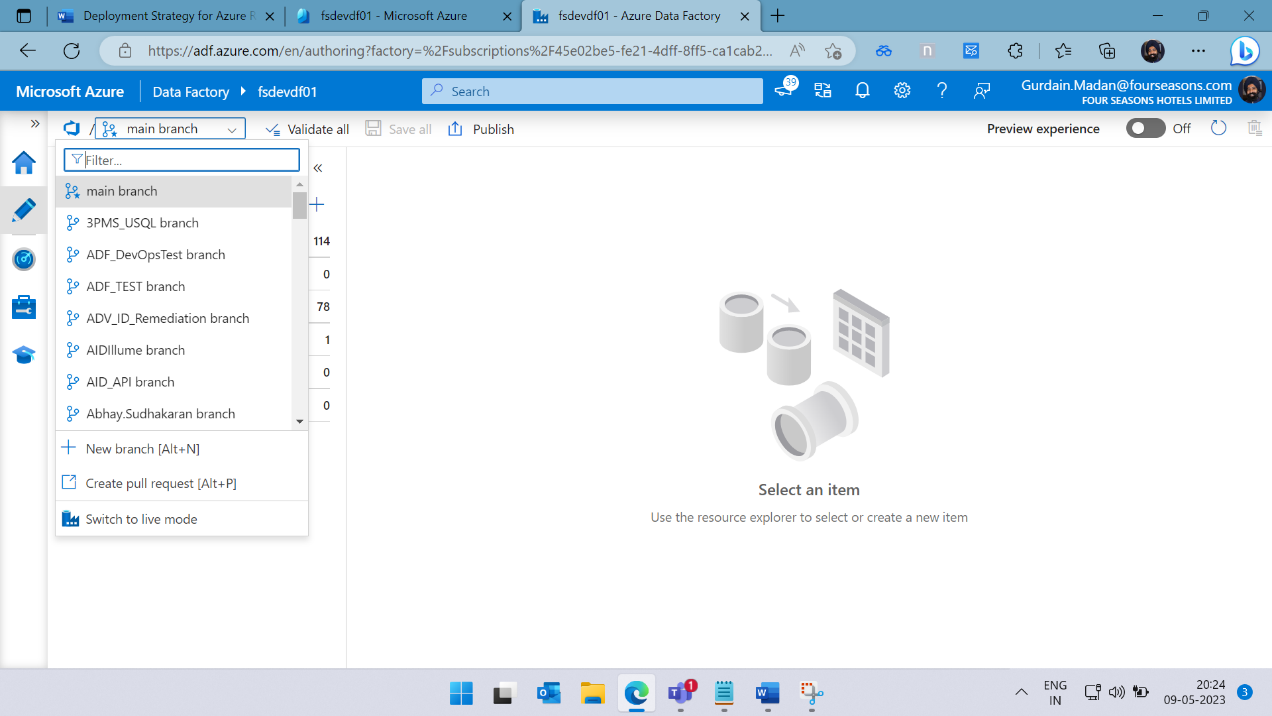
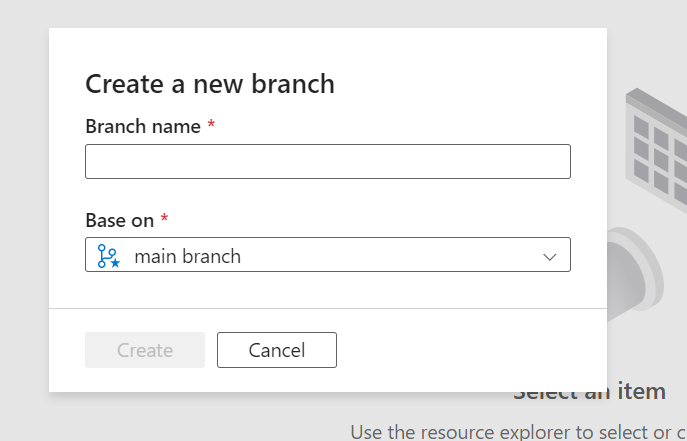
Steps to deploy ARM Template of Pipelines in Azure Data factory into a new/existing resource group of a target subscription

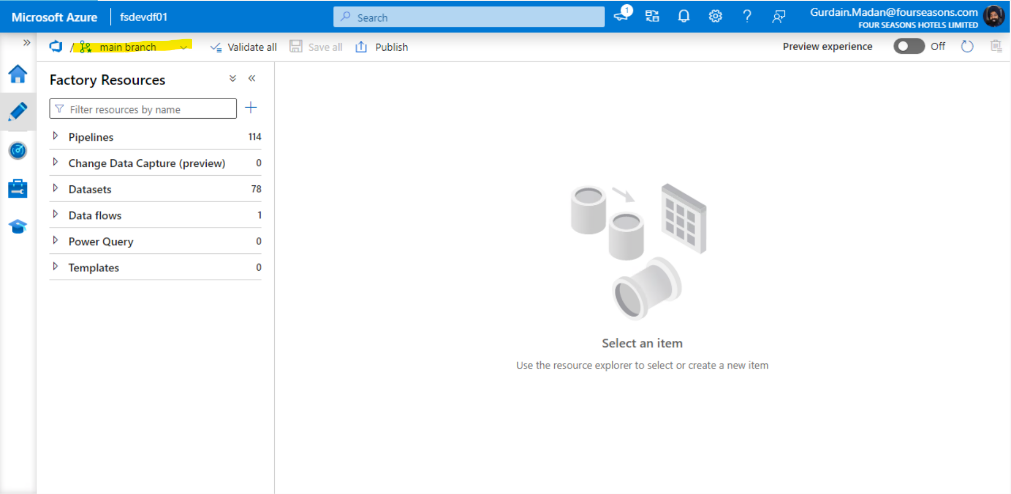
Option 1: using Azure DevOps Git

(Reference: fsproddf01 Data factory (V2) in FS.Phoenix.Prod.Integration Resource Group in FS-Phoenix-PROD Subscription)

1. Set up code repository for the Data factory

* Open Azure Data Factory Studio
* Click on ‘Set up code repository’
* 
* Select ‘Azure DevOps Git’ in Repository Type
* Select Azure Active Directory from dropdown
* Let ‘Select repository’ remain selected, choose a ‘Azure DevOps organization name’ from the dropdown
* Select a ‘Project Name’ from the dropdown
* Select a ‘Repository Name’ from the dropdown
* Select Collaboration Branch as ‘main’ branch(Select the branch name where you will collaborate with others and from which you will publish)
* Select Publish Branch as adf\_publish (Select the branch name where you will collaborate with others and from which you will publish)
* Select Root Folder (Displays the name of the folder to the location of your Azure Data Factory JSON resources are imported.)
* ‘Import existing resources to repository’ - This action exports each resource individually (that is, the linked services and datasets are exported into separate JSONs). When this box isn't selected, the existing resources aren't imported.
* Click on ‘Apply’

1. Create a new branch – Click on ‘+ New branch [Alt + N] ‘ in the dropdown shown in the screenshot.
2. The new branch will be the development branch where all the development will be carried out.
3. 
4. As development continues, Edit the main branch to have only those pipelines that are ready to be deployed to EBI-QA Environment for testing.
5. The main branch is our feature branch.
6. Make sure the final production pipelines are in the main branch. If any code needs to be pulled from a different branch into the main branch, select ‘Create pull request’ in the highlighted dropdown.



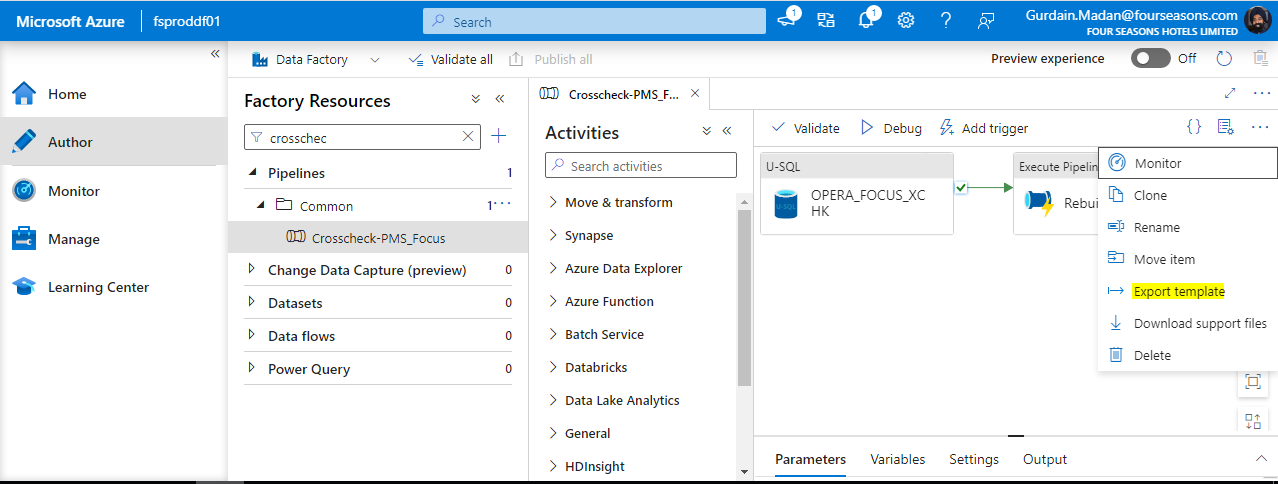
1. Our final pipelines and resources are now stored in the main branch of our source repository
2. Go to Azure DevOps - select the project that contains the main branch with our final production pipelines.
3. Select Repos/Files - Select the Data factory with our pipelines (Reference is fsdevdf01)
4. Select Repos/Pipelines - Create a Pipeline by clicking on ‘New pipeline’
   1. At the bottom of the screen, click on ‘Use the classic editor’
   2. Select a source as ‘Azure Repos Git’
   3. Select Team project, Repository
   4. Select ‘adf\_publish’ as the Default branch for manual and scheduled builds, select Continue
   5. Select a template, click on (*or start with an)* ‘Empty job’
   6. Under Tasks tab, Click on the ‘+’ sign next to Agent job 1 (Add a task to Agent job 1)
   7. Search ‘arm’, Add ‘ARM template deployment’
   8. Configure settings for ARM template deployment
      1. Select Deployment scope
      2. Select Azure Resource Manager connection (having access to the selected deployment scope)
      3. Select the Azure subscription (target subscription for deployment)
      4. Select ‘Create or Update Resource Group’ in Action
      5. Select the ‘Resource Group’ into which you would like to deploy the resource, Specify location
      6. Select Template location as Linked artifact, Template and Template parameters using the ‘…’ on the right
      7. IMPORTANT STEP: The changes that we want to make for the new environment. For example, change the ‘value’ of factoryName or change the connection string for storage account.
   9. If we want this pipeline to be triggered every time a publish takes place, we need to go to Triggers (Next to Tasks and Variables), select ‘Enable continuous integration’
   10. Click on ‘Save & queue’
5. If everything is proper and in order, we will get a green tick next to the pipeline name which means deployment has successfully taken place
6. Create a Release in Pipelines/Releases - ‘Create release’
   1. Under Artifacts, add the pipeline that we created in Step number 7, Enable the Continuous deployment trigger (symbol on the artifact name)
   2. For Stages, start with an ‘Empty Job’, select Stage name, stage will be created
7. Our Continuous Integration/Continuous Deployment process is ready

Option 2: using ARM Template for deployment

(Reference: Crosscheck-PMS\_focus pipeline in fsproddf01 Data factory (V2) in FS.Phoenix.Prod.Integration Resource Group in FS-Phoenix-PROD Subscription)

1. Export ARM Template (JSON) of pipeline as shown below.

In Author section of Azure Data Factory Studio, go to the specific pipeline, click on the 3 dots (Actions) and then select Export Template.



1. Decompile ARM Template JSON into Bicep using VS Code (with Bicep extensions)
   1. Install Bicep extension in Visual Studio Code (in Amazon WorkSpaces)
   2. Extract the downloaded zip folder ‘Crosscheck-PMS\_Focus.zip’
   3. Open ‘Crosscheck-PMS\_Focus.json’ in Visual Studio Code
   4. Install Microsoft Azure CLI ([How to install the Azure CLI | Microsoft Learn](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli))
   5. Install Azure CLI Tools in Visual Studio Code Extension
   6. Open a new Terminal (Ctrl+Shift+`), run ‘az login’ command, log into Azure account
   7. Install Bicep CLI using ‘az bicep install’ command
   8. Right click on ‘Crosscheck-PMS\_Focus.json’ and then select ‘Decompile into Bicep’. A new file called ‘Crosscheck-PMS\_Focus.bicep’ will be created
   9. Resolve the errors in the ‘Crosscheck-PMS\_Focus.bicep’ file

Error Type 1: The property "cancelAfter" is not allowed on objects of type "PipelinePolicy"

Resolution: Remove all instances of “cancelAfter: {}” from the .bicep file

Error Type 2: The property "lastPublishTime" is not allowed on objects of type "Pipeline". Permissible properties include "concurrency", "description", "runDimensions", "variables".

Resolution: Remove all instance of “lastPublishTime” from the .bicep file. For example, “lastPublishTime: 2022-08-26T18:15:33Z”

Error Type 3: The enclosing array expected an item of type "module[] | (resource | module) | resource[]", but the provided item was of type "string".bicep(BCP034)

Resolution: Replace the string with a reference to the resource name. For example, replace the string '${factoryId}/pipelines/C1\_to\_C2\_01-Master' with reference to the resource factoryName\_C1\_to\_C2\_01\_Master which has name property as ${factoryName}/C1\_to\_C2\_01-Master'.

'${factoryId}/pipelines/C1\_to\_C2\_01-Master' à

factoryName\_C1\_to\_C2\_01\_Master

Error Type 4: The property "authorizationType" expected a value of type "'Key' | 'RBAC'" but the provided value is of type "'Rbac'". Did you mean "'RBAC'"?bicep(BCP088)

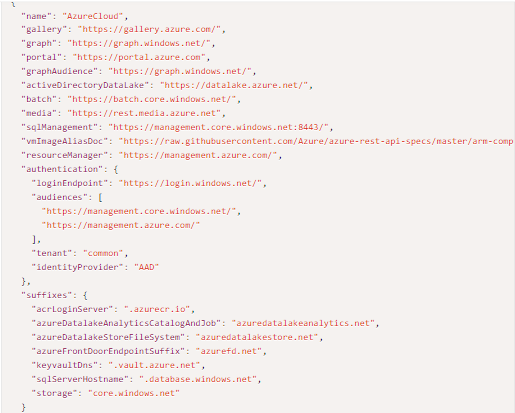
Resolution: Change ‘Rbac’ to ‘RBAC’

Error Type 5: Secure parameters should not have hardcoded defaults (except for empty or newGuid()).bicep corehttps://aka.ms/bicep/linter/secure-parameter-default

Resolution: Remove the hardcoded default value. The default value is given in the ARMTemplateParametersForFactory.json file which will be provided as an input at the time of deployment.

Error Type 6: Environment URLs should not be hardcoded. Use the environment() function to ensure compatibility across clouds.

Reference: environment() function



* Found this disallowed host: "vault.azure.net"

https://aka.ms/bicep/linter/no-hardcoded-env-urls

Resolution: Replace “vault.azure.net” with “${environment().suffixes.keyVaultDns}”

* Found this disallowed host: "azuredatalakestore.net"

https://aka.ms/bicep/linter/no-hardcoded-env-urls

Resolution: Replace “azuredatalakestore.net” with

“${environment().suffixes.azureDatalakeStoreFileSystem}”

* 1. Our bicep template is now ready for deployment.

1. Decide the scope of deployment

* resourceGroup (default) – All resources will be deployed to an existing resource group within the target subscription.
* subscription level – change the scope of deployment to subscription level by adding the line targetScope=’subscription’ at the top of the file.

All resources will be deployed to a new resource group in the target subscription. However, not all resources can be deployed using subscription level deployment.

1. Deploy the template at the desired scope level

* Set the default subscription for all the Azure CLI commands that you run in this session by running the below command in VS Code.

az account set --subscription "Your Subscription Name or ID"

* For resource Group level deployment, run the following commands in Azure CLI in Visual Studio Code
  + Before actual deployment, use the ‘-c’ confirm command to preview changes that will take place during deployment. For example,

az deployment group create --resource-group testrg --name rollout01 \

--template-file rgtemplate.bicep --parameters @myparameters.json -c

* + Confirm the changes from the previous step are as desired. Then run the following command for deployment

az deployment group create --resource-group testrg --name rollout01 \

--template-file rgtemplate.bicep --parameters @myparameters.json

* For subscription level deployment, run the following command in Azure CLI in Visual Studio Code: (location will be the location of the resource group where we want to deploy the resources)
* Before actual deployment, use the ‘-c’ confirm command to preview changes that will take place during deployment. For example,

templateFile="template.bicep"

today=$(date +"%d-%b-%Y")

deploymentName="sub-scope-"$today

az deployment sub create \

--name $deploymentName \

--location westus \

--template-file $templateFile\

-c

* + Confirm the preview changes are as desired for deployment, then run the deployment command in Azure CLI in Visual Studio Code (location will be the location of the resource group where we want to deploy the resources)

templateFile="template.bicep"

today=$(date +"%d-%b-%Y")

deploymentName="sub-scope-"$today

az deployment sub create \

--name $deploymentName \

--location westus \

--template-file $templateFile

* Deploy at multiple scopes: (Not required currently for our use case)

Here's an example Bicep file that's deployed with a targetScope of a subscription, but uses a module to deploy some resources to a resource group:

targetScope = 'subscription'

module networkModule 'modules/network.bicep' = {

scope: resourceGroup('ToyNetworking')

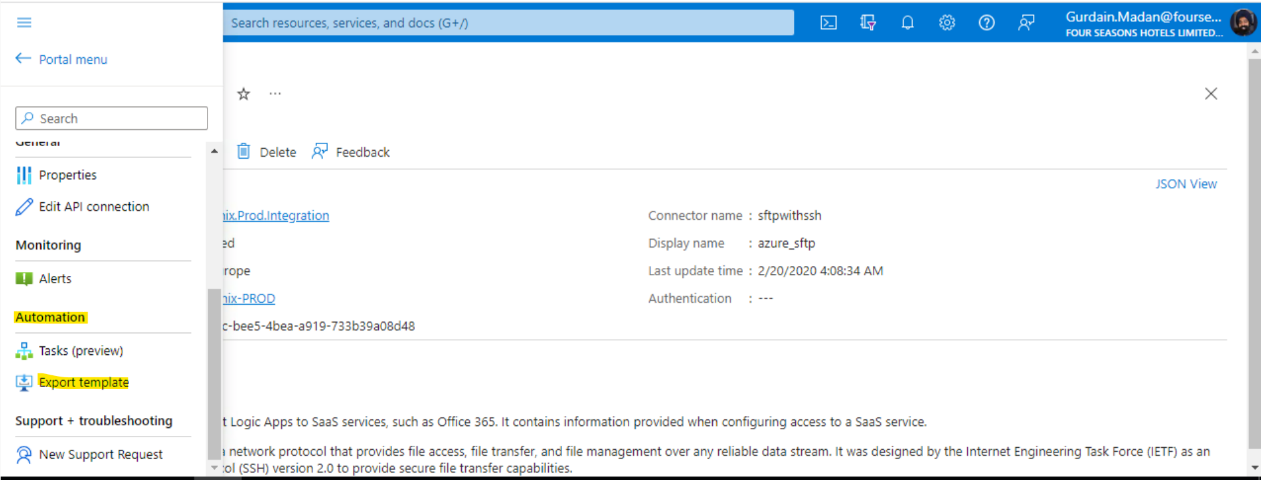
name: 'networkModule'

}

Steps to deploy ARM Template of Azure API Connection into a new/existing resource group of a target subscription

(Reference: sftpwithssh (“azure\_sftp”) API Connection in FS.Phoenix.Prod.Integration Resource Group in FS-Phoenix-PROD Subscription)

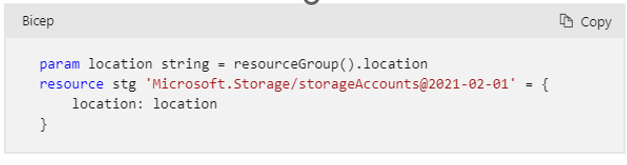
1. Download the ARM Template for the API Connection from the Azure Portal under Automation/Export Template in the homepage of the specific API Connection (in this case, sftpwithssh or azure\_sftp)



1. Decompile the ARM Template JSON to Bicep using VS Code
   1. Install Bicep extension in Visual Studio Code (in Amazon WorkSpaces)
   2. Extract the downloaded zip folder ‘ExportedTemplate-{ResourceGroupName}.zip’ , in this case ‘ExportedTemplate- FS.Phoenix.Prod.Integration’
   3. Open template.json and parameters.json (file will exist if parameters are present in the resource) in Visual Studio Code
   4. Install Microsoft Azure CLI (How to install the Azure CLI | Microsoft Learn)
   5. Install Azure CLI Tools in Visual Studio Code Extension
   6. Open a new Terminal (Ctrl+Shift+`), run ‘az login’ command, log into Azure account
   7. Install Bicep CLI using ‘az bicep install’ command
   8. Right click on ‘template.json’ and then select ‘Decompile into Bicep’. A new file called ‘template.bicep’ will be created
   9. Resolve the errors in the template.bicep file

Error Type 1: A resource location should not use a hard-coded string or variable value. Please use a parameter value, an expression, or the string 'global'. Found: 'northeurope'

Resolution: Replace the hardcoded value with a parameter ‘location’. Define the parameter at the start of the file as param location string = resourceGroup().location . We will provide this value at the time of deployment; this is the location of the resource group that the resources will be deployed to.



Error Type 2: The property "kind" does not exist in the resource or type definition, although it might still be valid. If this is an inaccuracy in the documentation, please report it to the Bicep Team.

Resolution: This may not be an issue at the moment and will not affect deployment.

You can use the **#disable-next-line BCP187** directive above the line where the warning is being thrown to disable the errors. For example:

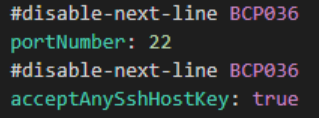


Error Type 3: The property "portNumber" expected a value of type "string" but the provided value is of type "22".

The property "acceptAnySshHostKey" expected a value of type "string" but the provided value is of type "true".

Resolution: This may not be an issue at the moment and will not affect deployment.

You can use the **#disable-next-line BCP187** directive above the line where the warning is being thrown to disable the errors. For example:



Error Type 4: If property "id" represents a resource ID, it must use a symbolic resource reference, be a parameter or start with one of these functions: extensionResourceId, guid, if, reference, resourceId, subscription, subscriptionResourceId, tenantResourceId.

Resolution:

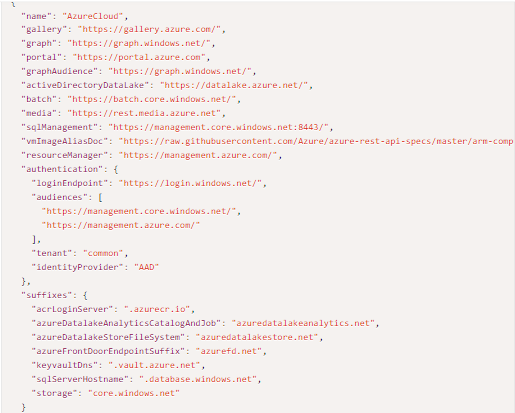
Replace 

With



Error Type 5: Environment URLs should not be hardcoded. Use the environment() function to ensure compatibility across clouds. Found this disallowed host: "management.azure.com"

Resolution: Replace “https://management.azure.com” with “${environment().resourceManager}”



* 1. Our bicep template is now ready for deployment

1. Decide the scope of deployment

* resourceGroup (default) – All resources will be deployed to an existing resource group within the target subscription.
* subscription level – change the scope of deployment to subscription level by adding the line targetScope=’subscription’ at the top of the file.

All resources will be deployed to a new resource group in the target subscription. However, not all resources can be deployed using subscription level deployment.

1. Deploy the template at the desired scope level

* Set the default subscription for all the Azure CLI commands that you run in this session by running the below command in VS Code.

az account set --subscription "Your Subscription Name or ID"

* For resource Group level deployment, run the following commands in Azure CLI in Visual Studio Code
* Before actual deployment, use the ‘-c’ confirm command to preview changes that will take place during deployment. For example,

az deployment group create --resource-group testrg --name rollout01 \

--template-file rgtemplate.bicep --parameters @myparameters.json -c

* Confirm the changes from the previous step are as desired. Then run the following command for deployment

az deployment group create --resource-group testrg --name rollout01 \

--template-file rgtemplate.bicep --parameters @myparameters.json

* For subscription level deployment, run the following command in Azure CLI in Visual Studio Code: (location will be the location of the resource group where we want to deploy the resources)
* Before actual deployment, use the ‘-c’ confirm command to preview changes that will take place during deployment. For example,

templateFile="template.bicep"

today=$(date +"%d-%b-%Y")

deploymentName="sub-scope-"$today

az deployment sub create \

--name $deploymentName \

--location westus \

--template-file $templateFile\

-c

* + Confirm the preview changes are as desired for deployment, then run the deployment command in Azure CLI in Visual Studio Code (location will be the location of the resource group where we want to deploy the resources)

templateFile="template.bicep"

today=$(date +"%d-%b-%Y")

deploymentName="sub-scope-"$today

az deployment sub create \

--name $deploymentName \

--location westus \

--template-file $templateFile

* Deploy at multiple scopes: (Not required currently for our use case)

Here's an example Bicep file that's deployed with a targetScope of a subscription, but uses a module to deploy some resources to a resource group:

targetScope = 'subscription'

module networkModule 'modules/network.bicep' = {

scope: resourceGroup('ToyNetworking')

name: 'networkModule'

}

Steps to deploy Logic App into a target subscription

Option 1: Create CI/CD Azure DevOps Pipeline

1. Select Repos/Pipelines in Azure DevOps - Create a Pipeline by clicking on ‘New pipeline’
   1. At the bottom of the screen, click on ‘Use the classic editor’

for the SOURCE,

* 1. Select a source as ‘Azure Repos Git’
  2. Select Team project, Repository
  3. Select the Default branch for manual and scheduled builds, select Continue
  4. Select a template, click on (*or start with an)* ‘Empty job’
  5. Under Tasks tab, Click on the ‘+’ sign next to Agent job 1 (Add a task to Agent job 1)
  6. Search ‘arm’, Add ‘ARM template deployment’
  7. Configure settings for ARM template deployment
     1. Select Deployment scope
     2. Select Azure Resource Manager connection (having access to the selected deployment scope)
     3. Select the Azure subscription (target subscription for deployment)
     4. Select ‘Create or Update Resource Group’ in Action
     5. Select the ‘Resource Group’ into which you would like to deploy the resource, Specify location
     6. Select Template location as Linked artifact, Template and Template parameters using the ‘…’ on the right
     7. Select Override template parameters - IMPORTANT STEP: The changes that we want to make for the new environment. For example, change the ‘value’ of Name or change the connection string for storage account.

1. If we want this pipeline to be triggered every time a publish takes place, we need to go to Triggers (Next to Tasks and Variables), select ‘Enable continuous integration’

j. Click on ‘Save & queue’

1. If everything is proper and in order, we will get a green tick next to the pipeline name which means deployment has successfully taken place
2. Create a Release in Pipelines/Releases - ‘Create release’

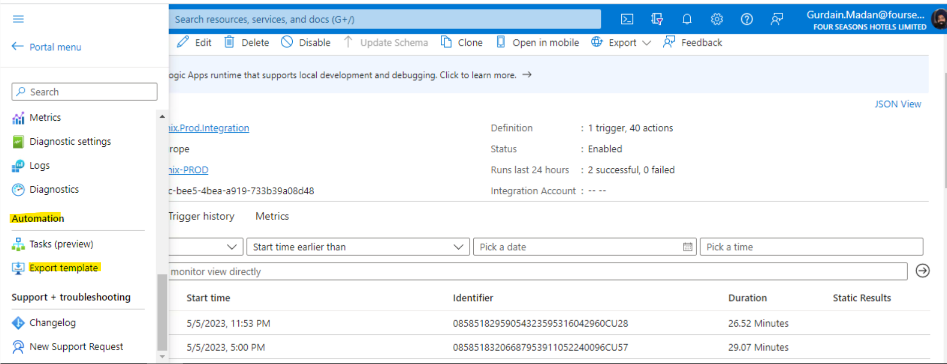
* Under Artifacts, add the pipeline that we created in Step number 1, Enable the Continuous deployment trigger (symbol on the artifact name)
* For Stages, start with an ‘Empty Job’, select Stage name, stage will be created

1. Our Continuous Integration/Continuous Deployment process is ready

Option 2:

(Reference: focus-master Logic App in FS.Phoenix.Prod.Integration Resource Group in FS-Phoenix-PROD Subscription)

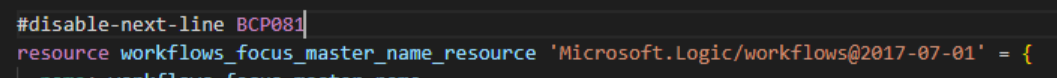
1. Download the ARM Template for the Logic App from the Azure Portal under Automation/Export Template in the homepage of the specific Logic app (in this case, focus-master)



1. Decompile the ARM Template JSON to Bicep using VS Code
   1. Install Bicep extension in Visual Studio Code (in Amazon WorkSpaces)
   2. Extract the downloaded zip folder ‘ExportedTemplate-{ResourceGroupName}.zip’ , in this case ‘ExportedTemplate- FS.Phoenix.Prod.Integration’
   3. Open template.json and parameters.json (file will exist if parameters are present in the resource) in Visual Studio Code
   4. Install Microsoft Azure CLI ([How to install the Azure CLI | Microsoft Learn](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli))
   5. Install Azure CLI Tools in Visual Studio Code Extension
   6. Open a new Terminal (Ctrl+Shift+`), run ‘az login’ command, log into Azure account
   7. Install Bicep CLI using ‘az bicep install’ command
   8. Right click on ‘template.json’ and then select ‘Decompile into Bicep’. A new file called ‘template.bicep’ will be created
   9. Resolve the errors in the template.bicep file

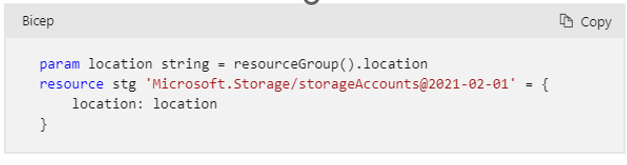
Error Type 1: Resource type "Microsoft.Logic/workflows@2017-07-01" does not have types available.

Resolution: Although a warning is shown, compilation will still succeed - you are just missing some of the enhanced type validation available for other resources. You can use the **#disable-next-line BCP081** directive above the line where the warning is being thrown to disable the errors. For example:



Error Type 2: A resource location should not use a hard-coded string or variable value. Please use a parameter value, an expression, or the string 'global'. Found: 'northeurope'

Resolution: Replace the hardcoded value with a parameter ‘location’. Define the parameter at the start of the file as param location string = resourceGroup().location . We will provide this value at the time of deployment, this is the location of the resource group that the resources will be deployed to.



* 1. Our bicep template is now ready for deployment

1. Decide the scope of deployment

* resourceGroup (default) – All resources will be deployed to an existing resource group within the target subscription.
* subscription level – change the scope of deployment to subscription level by adding the line targetScope=’subscription’ at the top of the file.

All resources will be deployed to a new resource group in the target subscription. However, not all resources can be deployed using subscription level deployment.

1. Deploy the template at the desired scope level

* Set the default subscription for all the Azure CLI commands that you run in this session by running the below command in VS Code.

az account set --subscription "Your Subscription Name or ID"

* For resource Group level deployment, run the following commands in Azure CLI in Visual Studio Code
  + Before actual deployment, use the ‘-c’ confirm command to preview changes that will take place during deployment. For example,

az deployment group create --resource-group testrg --name rollout01 \

--template-file rgtemplate.bicep --parameters @myparameters.json -c

* + Confirm the changes from the previous step are as desired. Then run the following command for deployment

az deployment group create --resource-group testrg --name rollout01 \

--template-file rgtemplate.bicep --parameters @myparameters.json

* For subscription level deployment, run the following command in Azure CLI in Visual Studio Code: (location will be the location of the resource group where we want to deploy the resources)
* Before actual deployment, use the ‘-c’ confirm command to preview changes that will take place during deployment. For example,

templateFile="template.bicep"

today=$(date +"%d-%b-%Y")

deploymentName="sub-scope-"$today

az deployment sub create \

--name $deploymentName \

--location westus \

--template-file $templateFile\

-c

* + Confirm the preview changes are as desired for deployment, then run the deployment command in Azure CLI in Visual Studio Code (location will be the location of the resource group where we want to deploy the resources)

templateFile="template.bicep"

today=$(date +"%d-%b-%Y")

deploymentName="sub-scope-"$today

az deployment sub create \

--name $deploymentName \

--location westus \

--template-file $templateFile

* Deploy at multiple scopes: (Not required currently for our use case)

Here's an example Bicep file that's deployed with a targetScope of a subscription, but uses a module to deploy some resources to a resource group:

targetScope = 'subscription'

module networkModule 'modules/network.bicep' = {

scope: resourceGroup('ToyNetworking')

name: 'networkModule'

}