Runbook for Maintaining ARM Templates in MLOps

Table of Contents

Introduction to Azure Resource Manager (ARM) Templates

Azure Resource Manager (ARM) templates are a foundational component of infrastructure as code (IaC) in Microsoft Azure. These templates are written in JSON (JavaScript Object Notation) and are used to define and deploy Azure resources in a declarative and repeatable manner.

Why are ARM Templates Used?

Azure Resource Manager (ARM) templates are used for managing and deploying resources in Microsoft Azure. They are JSON files that define the infrastructure and configuration for your Azure resources. ARM templates serve several important purposes:

ARM templates are a critical tool for managing Azure resources efficiently, consistently, and securely, providing a foundation for Infrastructure as Code practices in Azure.

1. Declarative Infrastructure Definition: ARM templates in Azure allow the declarative definition of infrastructure, emphasizing the desired state of resources and their relationships.

2. Automated Resource Management: Automation tools like Azure PowerShell or DevOps can be used with ARM templates for resource provisioning, minimizing human error and ensuring consistency.

3. Consistency Across Environments: Infrastructure defined in code promotes consistency, enabling easy replication of configurations in different regions, subscriptions, or for various applications.

4. Versioning and Change Tracking: ARM templates can be stored in source control, facilitating versioning, code reviews, and change tracking, ensuring proper documentation of infrastructure modifications.

5. Scalable and Secure Architecture: ARM templates support the creation of scalable architectures with various resource types and also allow the enforcement of compliance and security policies, making them a foundation for efficient, secure, and scalable infrastructure management in Azure.

Benefits

1. **Consistency:** ARM templates promote consistent resource deployments, reducing configuration drift and manual errors.
2. **Scalability:** Templates are highly scalable, making them suitable for provisioning resources across multiple environments, regions, and projects.
3. **Reusability:** Well-designed templates can be reused for similar resource configurations, saving time and effort.
4. **Traceability:** ARM templates provide a clear audit trail of resource changes and deployments, enhancing transparency and compliance.

5. **Automation:** They facilitate automation of infrastructure provisioning, enabling rapid development and deployment of applications.

Key Concepts

- **Declarative Approach:** ARM templates describe the desired state of Azure resources rather than specifying the sequence of steps to create them. This declarative approach allows for idempotent deployments, where the same template can be used repeatedly without causing conflicts.
- **Resource Definition:** Templates define various Azure resources such as virtual machines, storage accounts, databases, and more, along with their configurations, dependencies, and relationships.
- **Parameterization:** ARM templates support parameters, enabling flexibility and customization when deploying resources. Parameters allow you to input values at deployment time, making templates reusable in various scenarios.
- **Dependency Management:** Templates handle the automatic resolution of resource dependencies, ensuring that resources are provisioned in the correct order to avoid deployment failures.
- **Version Control:** Like application code, ARM templates can be version-controlled using tools like Git, allowing for collaboration, change tracking, and rollbacks.
- **Automated Deployment:** ARM templates can be deployed through various methods, including the Azure Portal, Azure CLI, Azure PowerShell, and continuous integration/continuous deployment (CI/CD) pipelines.

How to Create/Generate ARM Template

Creating the ARM template manually requires the understanding of the each and every component of it mentioned above in the definition. If user knows what ARM template components mean, then manual creation of the ARM template can be opted. As basic use-case of ARM template is to create and maintain infrastructure, following two approaches can be opted for the creation of resource using ARM template.

1. Create a new resource using manually created ARM template
2. For specific resource, use default available templates.

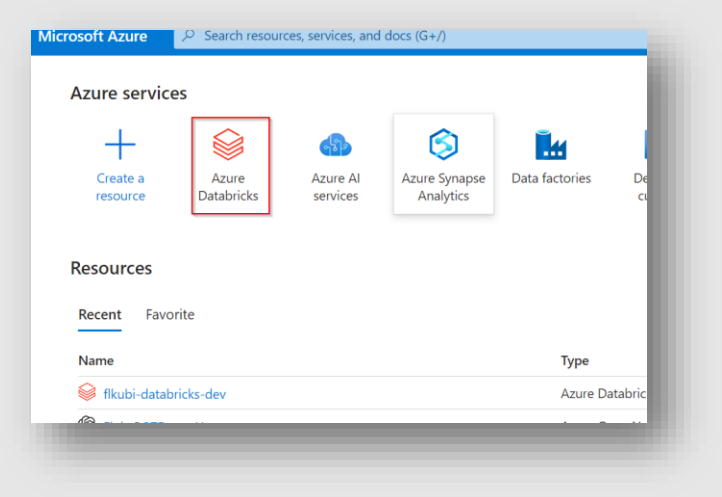
ARM template creation can be done via Azure Portal or via tool like Visual Studio Code.

ARM template generation can be done via Azure Portal, or via Azure CLI or PowerShell with queries.

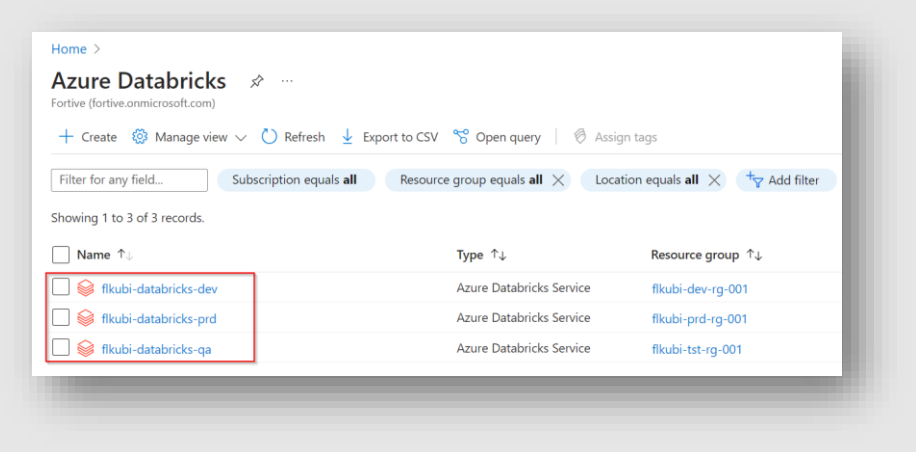
How to Generate ARM Templates for already existing resources

All the resources, created with/without use of ARM template are associated to an ARM code. The code can be generated from the portal for each of the resources. Refer to the following example for getting the code.

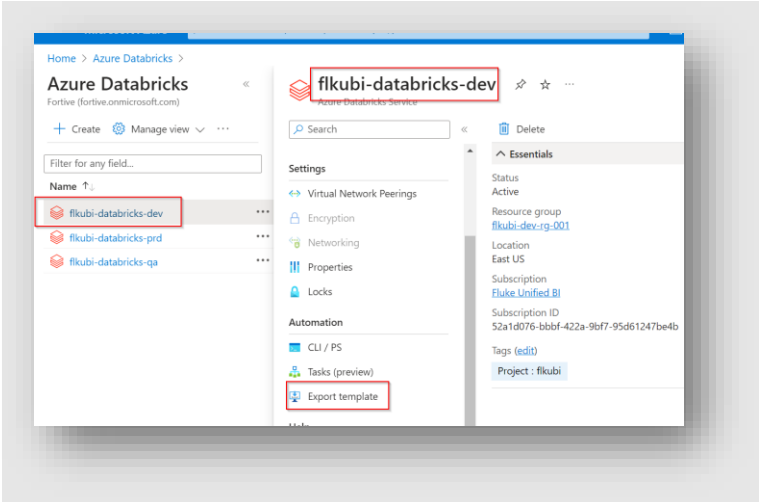
1. Go to portal.azure.com
2. Go to the Resource for which you want to generate ARM template.
3. For Databricks resource, go to Azure Databricks.



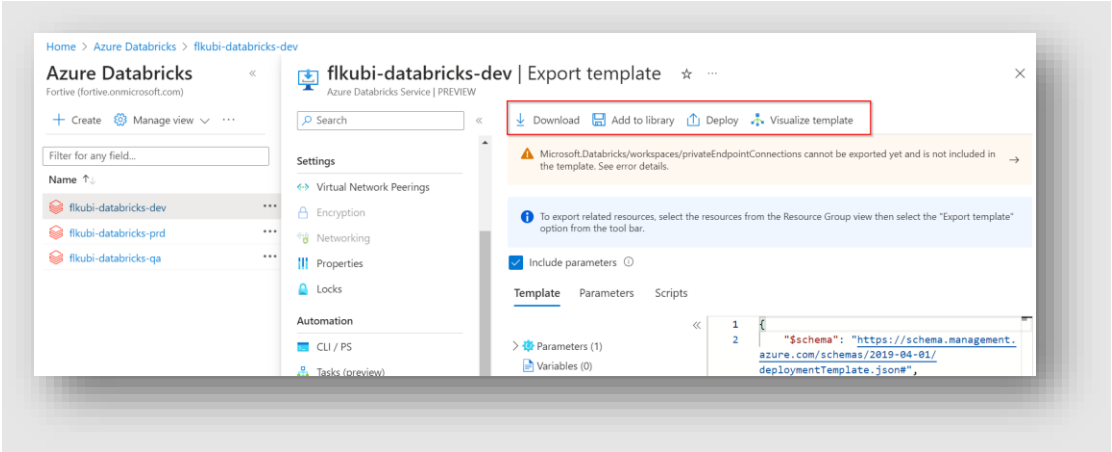
4. Now click on particular resource workspace/instance.



5. Click on the Export Template option on the left panel.



6. It will generate the template code for that resource. Portal will provide different options for it to save in library, redeploy from there. It will also allow to see/visualize the ARM template. All the options can be accessed via the top horizontal bar as shows in the below image.



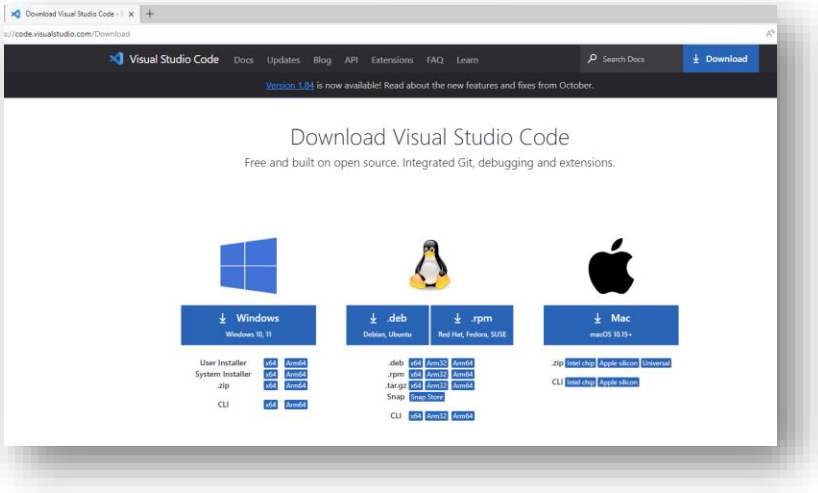
Create and Deploy your First ARM Template (Azure Storage Account)

In this section, we will be creating the ARM template for Azure Storage Account with the following resources:

a. Azure Storage Account

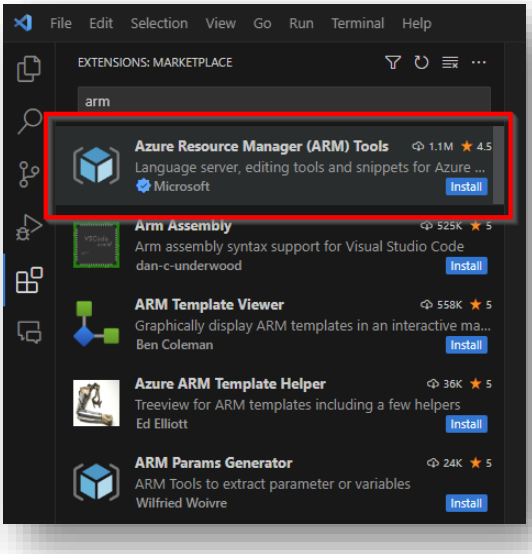
Prerequisites steps to create the ARM template in Visual Studio Code

1. Install Visual Studio Code.



Choose the appropriate version of Visual Studio Code and download and install it.

2. From the Visual Studio Code Extension Marketplace, install the extension: [Azure Resource Manager (ARM) Tools]

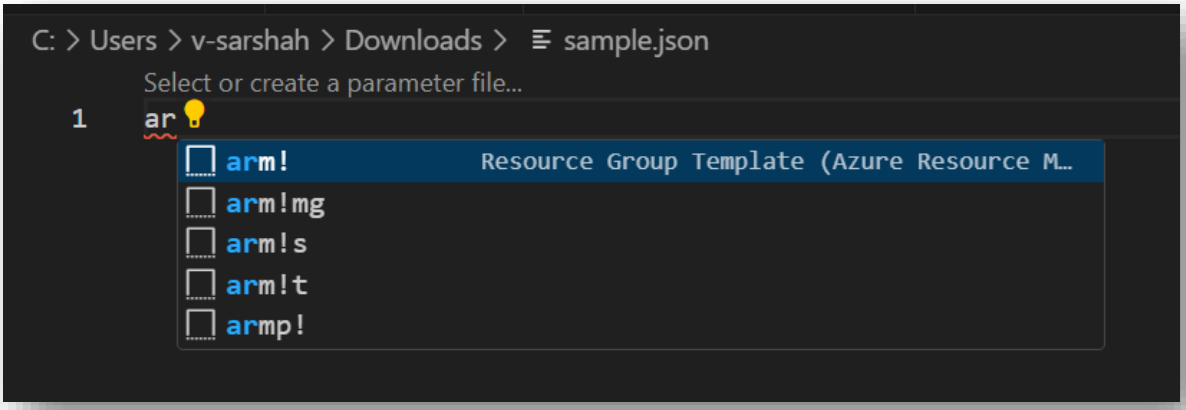


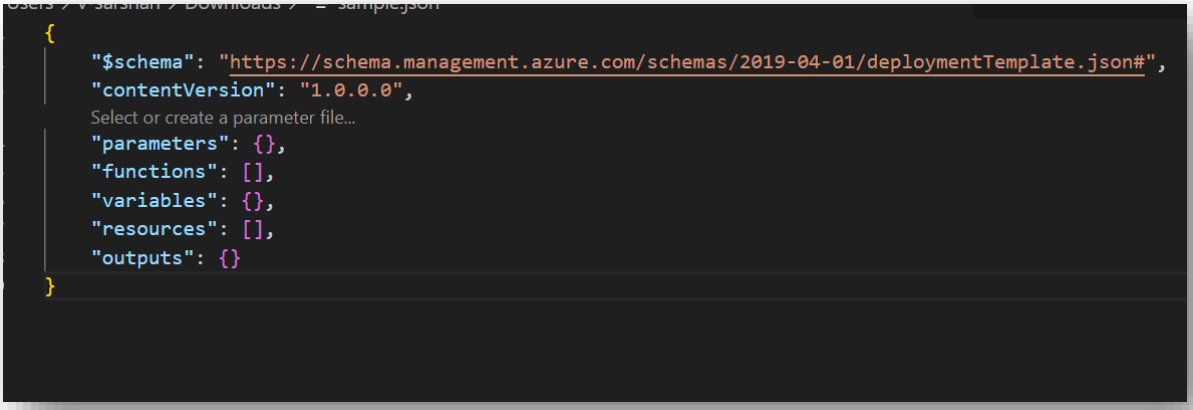
3. Please refer to the section [Deployment of ARM Template Using Azure CLI] and install the Azure CLI in the system and login using the ID-password.

ARM Template Creation

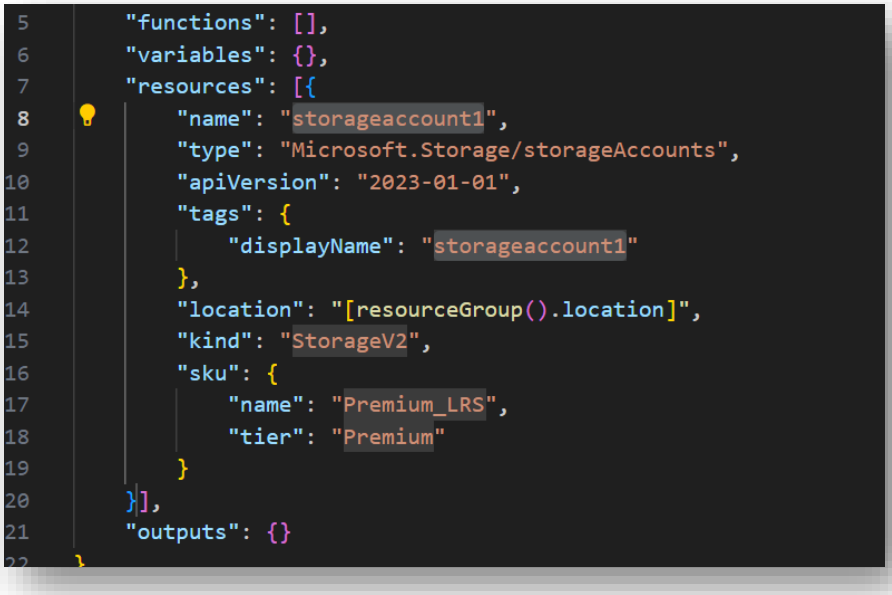
The resource group where the storage account is going to be created is already existing with name 'flk-erpchatbot-dev'.

1. Open the visual studio code, create a new file and save the file with ARMStorage.json name.
2. Now write arm!. It will show you multiple options, where arm! Command will create a sample ARM template schema.





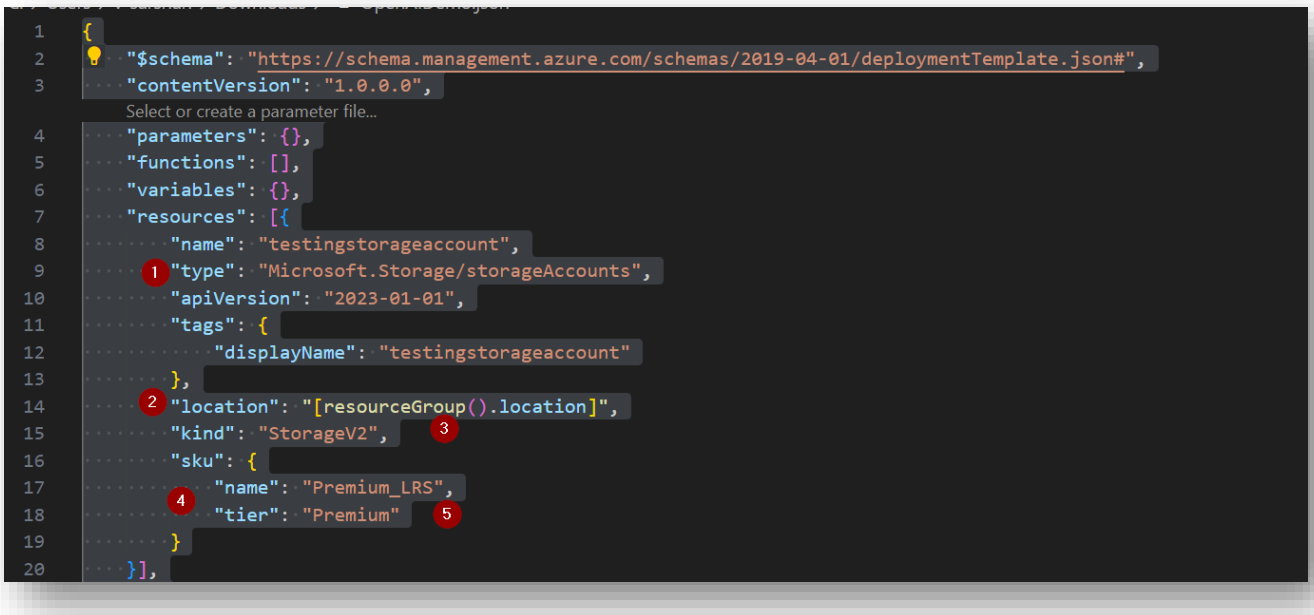
3. Now type arm- in the resources part. It will provide all available sample template of the resources. Select **arm-storage** from the list.



4. Update storage account name to ‘testingstorageaccount’ in the list.
5. Following is the sample JSON that will be created when you add.

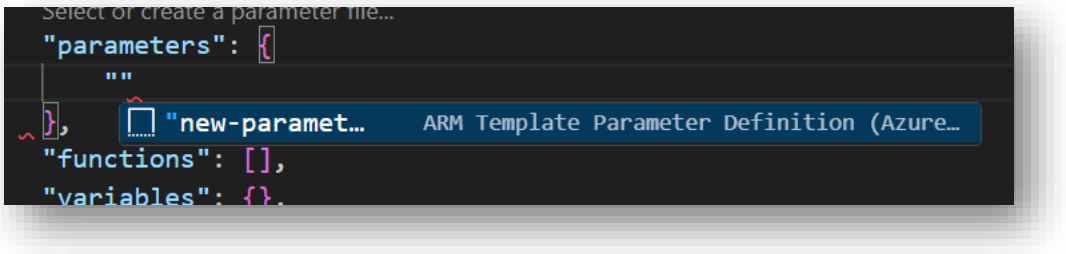
```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "functions": [],
    "variables": {},
    "resources": [{
        "name": "testingstorageaccount",
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2023-01-01",
        "tags": {
            "displayName": "testingstorageaccount"
        },
        "location": "[resourceGroup().location]",
        "kind": "StorageV2",
        "sku": {
            "name": "Premium_LRS",
            "tier": "Premium"
        }
    }],
    "outputs": {}
}
```

6. Now lets understand the template.
7. Type refers to type the services that you are creating. Please refer to the section [Appendix A - Structure of ARM Templates]
    a. Name is name of the storage account name.
    b. Location refers to the location where storage account will be deployed. It is given reference of the same location as the resource group.
    c. Kind refers to the Storage account type. Based on features and pricing is associated with the type of storage account. We have selected Storage V2, which is popular to have hierarchical namespace if we want.
    d. Pricing page for Storage account: Link
    e. Point 5 refers to Tier of the storage account. Standard and Premium are two options. The details of those can be checked in the pricing page.
    f. There are other main components like Variables, Parameters and Functions.
        i. **Parameters**: Parameters contain values for the resource. It can be taken as User Input, can be overridden easily. While deploying resources from Dev to QA or to other environment, resource connection strings and other dynamic things are handled and overridden in Parameters.
        ii. **Variables**: Variables as used internally in the ARM template where single value is referenced or used at multiple places then variables are useful.
        iii. **Functions**: Functions handle dynamicity in the ARM template. Different types of function like logical (i.e., if, and, true), string (i.e., concat, indexOf) numerical (i.e., int, float) etc., are included in the ARM template library



Parameterize the ARM Template

1. Lets add a parameter which allows us to set the name of Storage Account dynamically. Following is a sample parameter addition code. Inside the Parameters main tag, just by typing something, it will recommend you this sample code. Add this code in the Parameter array.



a. It will show the code like following

```
        "parameter1": {
            "type": "string",
            "metadata": {
                "description": "description"
            }
        }
```

2. Update it like following. Give name of the Parameter as EnvironmentName, and provide the proper description to it.

```
        "EnvironmentName": {
            "type": "string",
            "metadata": {
                "description": "The parameter depicts the environment name
i.e., Dev,QA or Prod."
            }
        }
```

a. Repeat the steps and add following parameters.

Note: Add comma ‘,’ after every parameter/variables/ or functions are defined.

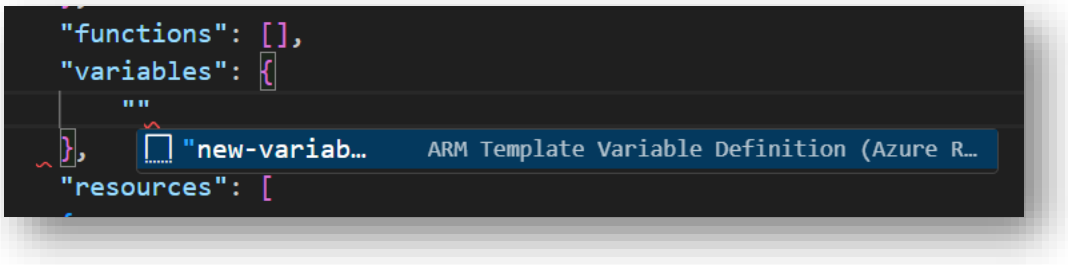
i. StorageAccountName: Represents the storage account name
ii. StorageAccountKind: Represents the storage account kind (i.e., Storage, StorageV2)
iii. StorageAccountSKUName: Represents storage account SKU name (i.e., Standard_LRS, Premium_LRS etc.)
iv. StorageAccountSKUTier: Represents storage account tier (i.e., Standard, Premium)

v. Refer the following code for the reference.

```
 "parameters": {
        "StorageAccountName": {
            "type": "string",
            "metadata": {
                "description": "The parameter depicts the name of the storage
account"
            },
        "EnvironmentName": {
            "type": "string",
            "metadata": {
                "description": "The parameter depicts the environment name
i.e., Dev,QA or Prod."
            },
        "StorageAccountKind": {
            "type": "string",
            "metadata": {
                "description": "The parameter depicts the environment name
i.e., StorageV2,Storage"
            },
        "StorageAccountSKUName": {
            "type": "string",
            "metadata": {
                "description": "The parameter depicts the environment name
i.e., Premium_LRS,Standard_LRS etc."
            },
        "StorageAccountSKUTier": {
            "type": "string",
            "metadata": {
                "description": "The parameter depicts the environment name
i.e., Premium,Standard etc."
            }

        }
    }
```

3. Now lets add variable which stores the name of the storage account. Same way that we added a parameter, by typing anything in variables section will give an option to create a new variable schema.



4. While clicking on it, it will add sample code for variable creation as below.
    a. “variable1: “value”
5. Now update the variable like following with the base name of the storage account.

```
"StorageAccount_APIVersion": "2023-01-01"
```

Use of Functions + Parameters + Variables

1. Now, Lets add the reference of the parameter in the ARM template where resource is getting created. Any referenced item should be called in side a rectangle brackets like [].
    a. We will create a resource with name **temp1t1es1t** *and different environment. Lets say, **devtemp1t1es1t** for dev, **qatemp1t1es1t** for QA and **prdtemp1t1es1t** for Prod. We will use **concat** function to create such name,

        *** : Name of the storage is chosen based on the availability of the names. It should be unique across the all Azure Storages. If normal names are not available, it will throw error in the deployment. Numeric values can be added to the name to make it unique.**

    b. Update the name of the resource with following:
    c. "name":" temp1t1es1t" ==> “**name**": **"[concat(parameters(StorageAccountName), parameters('EnvironmentType'))]**", Now whenever we deploy this, it will ask you the parameters name as user input.
    d. Update display name as “displayname”: " temp1t1es1t " ➔ **"displayName": "[toLower(concat(parameters('EnvironmentName'),parameters('StorageAccountName')))]",**
    e. Similarly update all the variables accordingly as given below.
        i. Update
        ii. Update API version as "apiVersion":"2023-01-01" ➔**"apiVersion": "[variables('StorageAccount_APIVersion')]"**
        iii. Update "kind":"StorgeV2" ➔ **"kind": "[parameters('StorageAccountKind')]"**
        iv. Update SKU name from "name": "Premium_LRS" ➔ **"name": "[parameters('StorageAccountSKUName')]"**
        v. Update SKU tier as "tier": "Premium" ➔ **”tier”= "[parameters('StorageAccountSKUTier')]"**

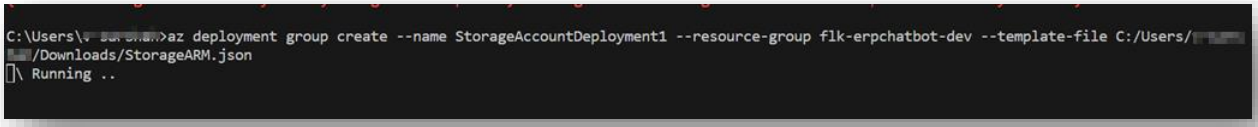
f. Parameters are referenced with parameters('<parametername>') and variables are referenced with variables('<variablename>').
g. Please refer to the following attached file for the full reference.

i.

Deploy Sample ARM Template

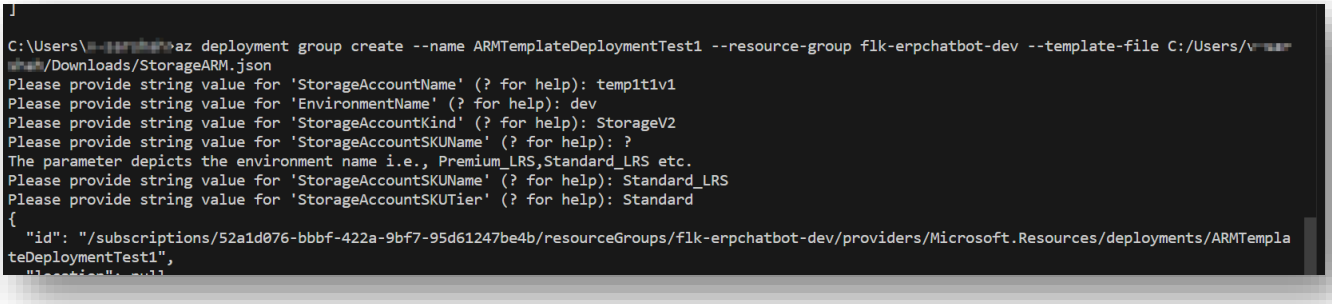
1. To deploy the simple ARM template StorageARM.json file, we will use Azure CLI.
2. Refer to the section for installation of Azure CLI and Log in.
3. Now, run the following command.
   a. az deployment group create --name <nameofdeployment> --resource-group <nameoftheresourcegroup> --template-file <templatefilepath>
   b. Replace the name nameofdeployment with any name that you want.
   c. Nameoftheresourcegroup should be replaced with resource group name where the resource is meant to be created.
   d. Templatefilepath refers to file path of the local system for StorageARM.json.

4. Now add the resource name and click enter.

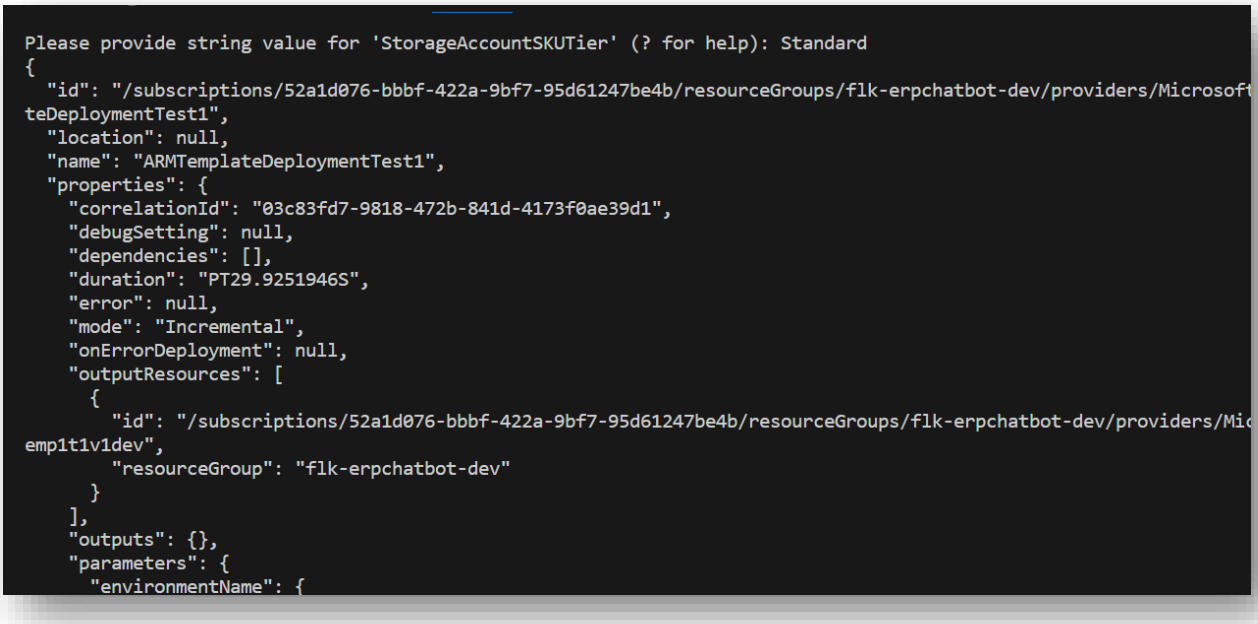


5. It will ask to input the parameter name. Add **dev** and click enter.*

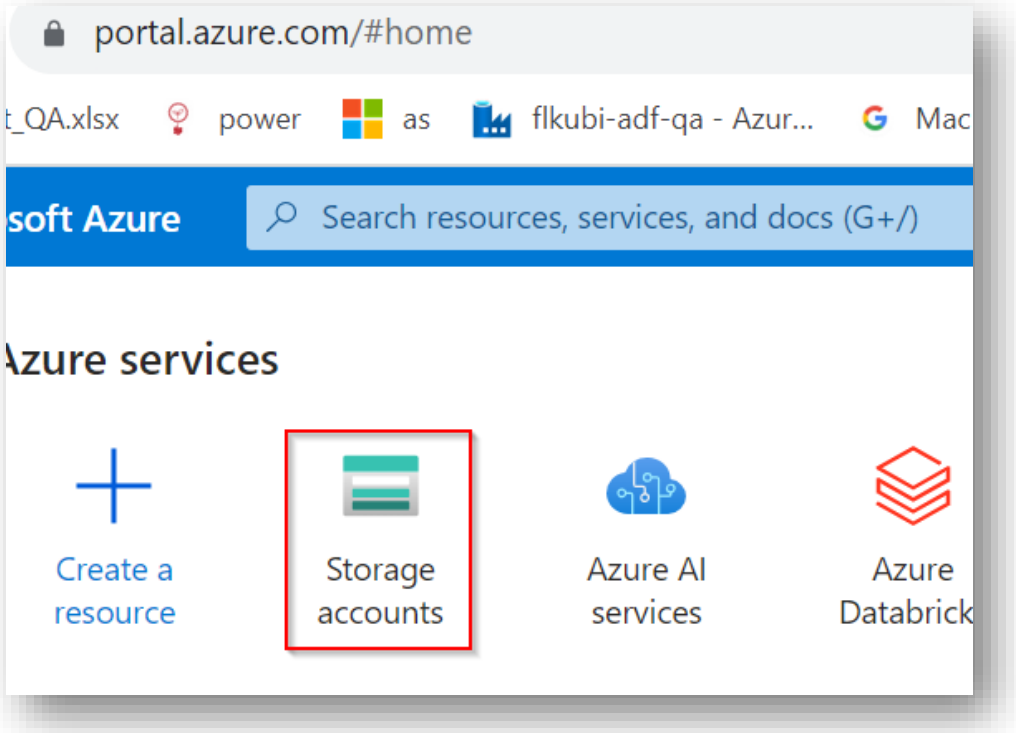
   ***: Enter '?' to see the sample values of the parameters**



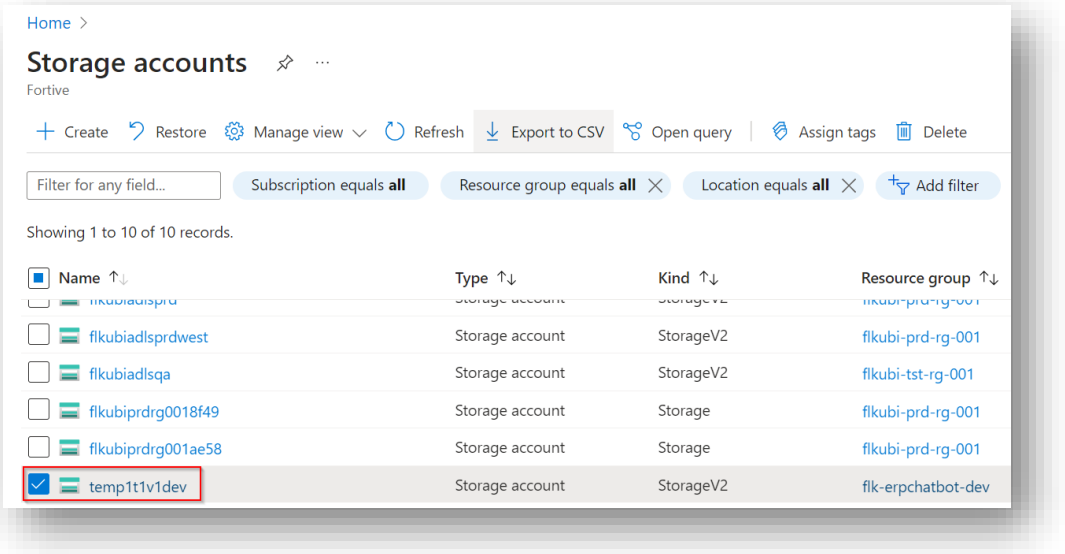
6. Once it is deployed, It will show all details of the created resource.



7. Verify resource creation from portal.
    a. Open Portal.Azure.Com -> Azure AI Services -> Azure Cognitive Search



b.



Resource Parameter Recommendation

Resource	Parameter	Suggestions	Recommendation	Reason
Storage Account	StorageAccountName	NA	The name must be unique across all existing storage account names in Azure. It must be 3 to 24 characters long, and can contain only lowercase letters and numbers.	Try to use numbers in between alphabets to make it unique
Storage Account	EnvironmentName	1. dev 2. qa 3. prod/prd	NA	NA
Storage Account	StorageAccountKind	1. StorageV2 2. Storage	StorageV2	It combines the older blob functionalities with hierarchical name space functions of newly launched Azure Storage.

Storage Account	StorageAccountSKUName	1. Standard_LRS 2. Premium_LRS 3. Standard_GRS 4. Premium_GRS	Standard_LRS -> if data privacy is very high and data can not be stored on global level data centers. Standard_GRS -> to store replica of data on global level	Please refer to the link
Storage Account	StorageAccountSKUTier	1. Standard 2. Premium	Standard	Please refer to the link

ARM Template for Open AI models

Any Open AI based project would require the following resources:

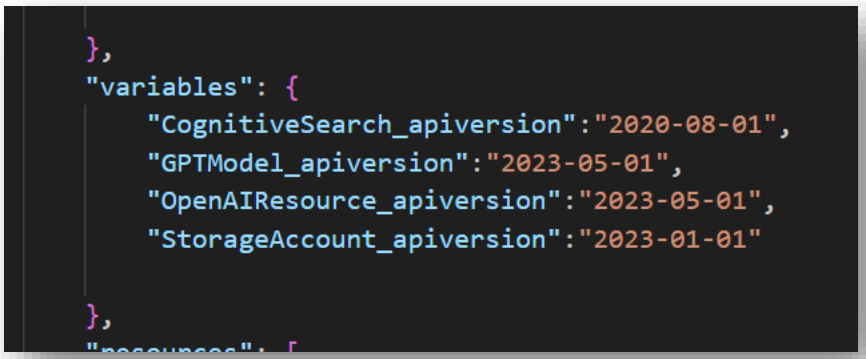
1. Azure Open AI resource
2. Azure Open AI Model Deployment
3. Azure Storage Account
4. Azure Cognitive Search Services

Following file contains the ARM template in dynamic manner.

Pre-requiresites: Please refer to the section [Prerequisites steps to create the ARM template in Visual Studio Code] for installing the pre-requisites.

How to use this template?

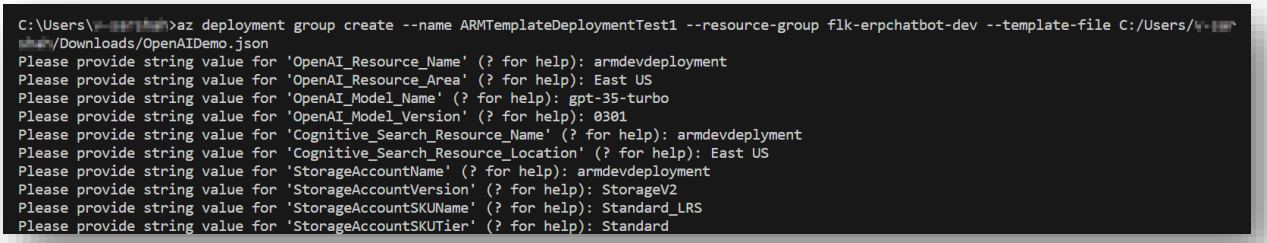
1. Save the file in the system.
    a. Double click on the file and save it in the system with save name.
2. Update following numbered details of the image.
    a. Update the api version details under variables section.



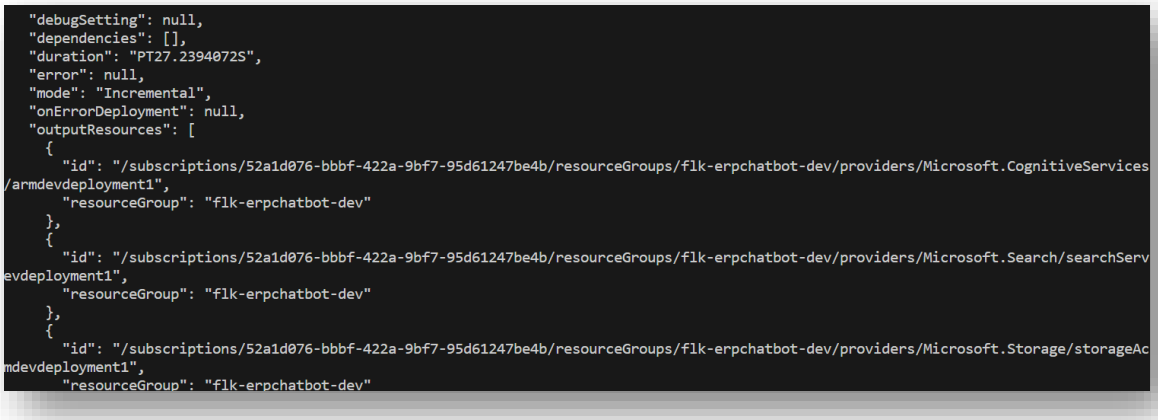
3. Now run the file using following command.
    a. az deployment group create --name <DeploymentName> --resource-group <ResourceGroupName> --template-file <fullpathoffile>
    b. DeploymentName: Provide any deployment name to deploy.
    c. ResourceGroupName: Enter resource group name where all the resources will be created
    d. Provide full path of OpenAIDemo.json.
4. Command will ask for following parameter names

    Note: Use '?' to get sample values for the parameters.

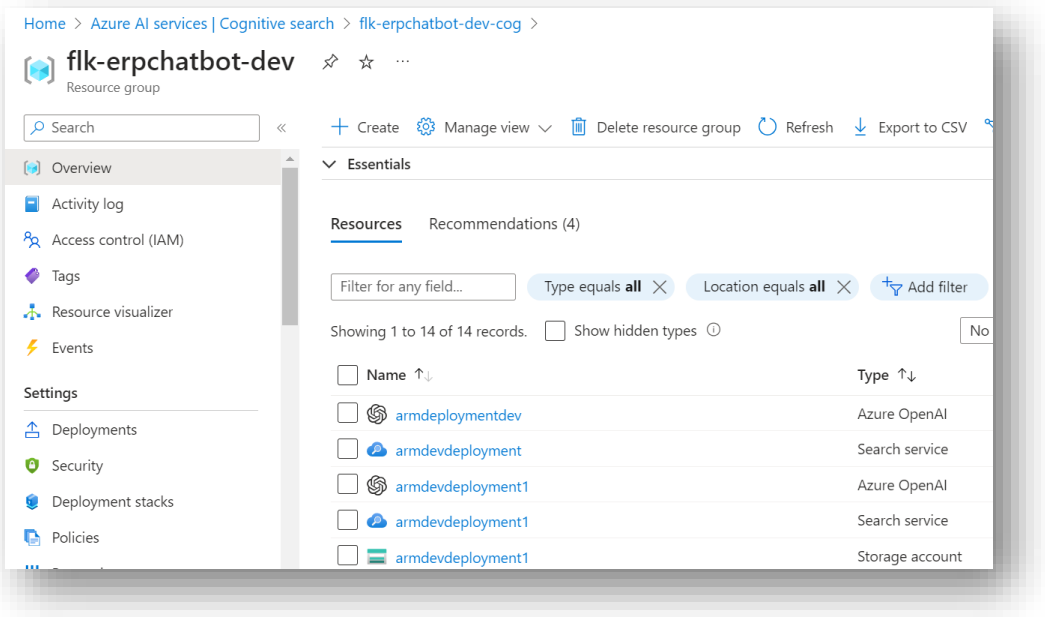
a. OpenAI_Resource_Name: Provide name of the open ai resource
b. OpenAI_Resource_Area: Provide the area associated with the resource. i.e., East US, East US2, West US etc)
c. OpenAI_Model_Name: Provide LLM model which you want to create (i.e., gpt-35-turbo, gpt-35-turbo-16k)
d. OpenAI_Model_Version: Provide version of the Open AI models (i.e., 0301,0613)
e. Cognitive_Search_Resource_Name: Provide cognitive search resource name.
f. Cognitive_Search_Resource_Location: Provide location for the Cognitive Search resource. (i.e., East US, West US)
g. StorageAccountName: Provide storage account name.
h. StorageAccountVersion: Provide kind of storage account. (i.e., StorageV2)
i. StorageAccountSKUName: Provide SKU Name for the storage account (i.e., Standard_LRS, Premium_LRS etc.)
j. StorageAccountSKUTier: Provide SKU tier for the storage account. (i.e., Standard, Premium)
k. Please refer to the following image for the sample values entered.



l. Once entered, it will start the deployment. Once deployment is completed, it will show screen like shown below with resourceIDs of the resources.



m. All created resources can be viewed under your selected resource group in portal.azure.com.



How did we create this template?

Follow these steps:

1. The template was created in the visual studio using ARM template extension.
    - Please refer to the [Prerequisites steps to create the ARM template in Visual Studio Code] to install it.
2. Now, we used command like arm-storage for storage account template. And update the values based on documentation.
3. We added Azure Open AI template using template generation method and updated the values using Microsoft documentation.
4. We also added Cognitive Search resources deployment sample template from the given documentation and updated the values in it.
5. We then parameterized the names of the resources, and created variables for api version. Please refer to the section [Parameterize the ARM Template] for more details on how to create parameters/variables.

Resource Parameter Recommendation

Resource	Parameter	Suggestions	Recommendation	Reason
Open AI Resource	OpenAI_Resource_Name	NA	Your resource name can only include alphanumeric characters and hyphens and can't start or end with a hyphen.	NA
Open AI Resource	OpenAI_Resource_Area	1. East US 2. West US 3. East US2	Keep area same as area of resource group under which the resource is to be created.	NA
Open AI Resource	OpenAI_Model_Name	1. gpt-35-turbo 2. gpt-35-turbo-16k 3. gpt-35-turbo-instruct	gpt-35-turbo.	Most of the Questions and Answers will be under 4000 words/tokens. If requirement is higher then choose 16k model.
Open AI Resource	OpenAI_Model_Version	1. 0301, 0613 (for gpt-35-turbo) 2. 0613 (for gpt-35-turbo-16k) 2. 0914 (for gpt-35-instruct)	Select the latest one, higher one based on requirement.	NA
Azure Cognitive Search	Cognitive_Search_Resource_Name	NA	Service name must only contain lowercase letters, digits or dashes, cannot use dash as the first two or last one characters, cannot contain consecutive dashes, and is limited between 2 and 60 characters in length.	NA

Azure Cognitive Search	Cognitive_Search_Resource_Location	1. East US 2. West US 3. East US2	Keep area same as area of resource group under which the resource is to be created.	NA
Storage Account	StorageAccountName	NA	The name must be unique across all existing storage account names in Azure. It must be 3 to 24 characters long, and can contain only lowercase letters and numbers.	Try to use numbers in between alphabets to make it unique
Storage Account	StorageAccountVersion	1. StorageV2 2. Storage	StorageV2	It combines the older blob functionalities with hierarchical name space functions of newly launched Azure Storage.
Storage Account	StorageAccountSKUName	1. Standard_LRS 2. Premium_LRS 3. Standard_GRS 4. Premium_GRS	Standard_LRS -> if data privacy is very high and data can not be stored on global level data centers. Standard_GRS -> to store replica of data on global level	Please refer to the link
Storage Account	StorageAccountSKUTier	1. Standard 2. Premium	Standard	Please refer to the link

ARM Template for ML/DL models

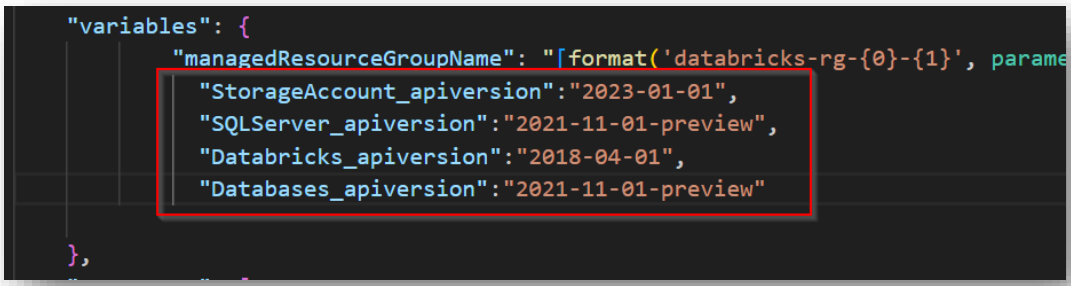
Any ML/DL based project would require the following resources:

1. Azure Databricks/Azure ML
2. Azure Storage
3. Azure Synapse (if required)

Following file contains the ARM template in dynamic manner.

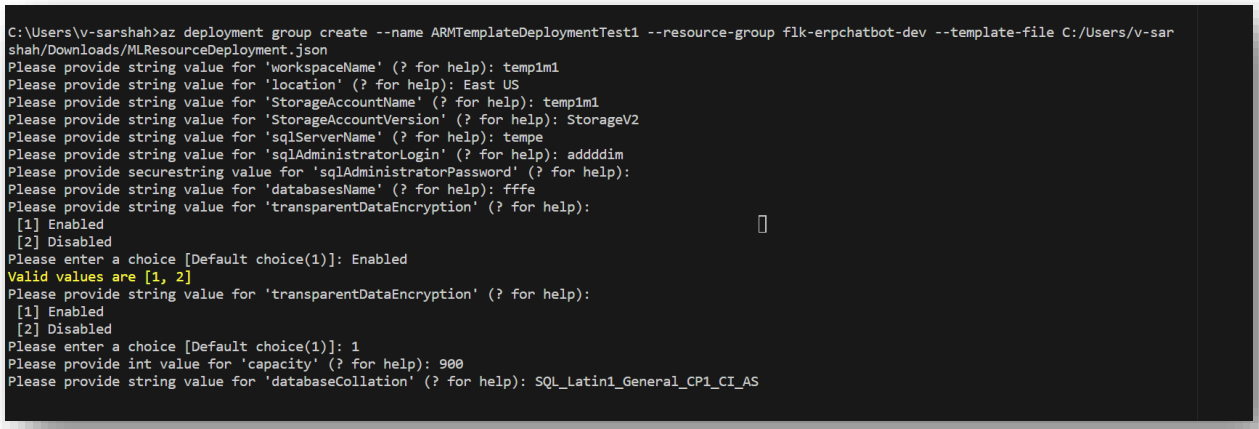
How to use this template?

1. Save the file in the system.
    a. Double click on the file and save it in the system with save name.
2. Update following numbered details of the image.
    a. Update the api version details under variables section.

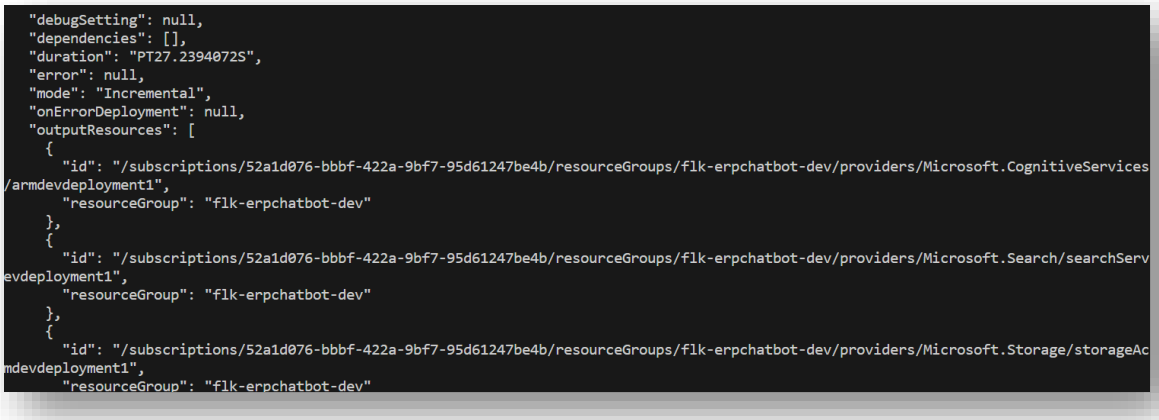


3. Now run the file using following command.
    a. az deployment group create --name <DeploymentName> --resource-group <ResourceGroupName> --template-file <fullpathoffile>
    b. DeploymentName: Provide any deployment name to deploy.
    c. ResourceGroupName: Enter resource group name where all the resources will be created
    d. Provide full path of OpenAIDemo.json.
4. Command will ask for following parameter names
    a. workspaceName: Provide Databricks workspace name
    b. StorageAccountName: Provide storage account name.
    c. StorageAccountVersion: Provide kind of storage account. (i.e., Storage, StorageV2)

d. sqlAdministratorLogin: Provide user id for SQL database. Based on provided user id it will create new user id using that name.
e. sqlAdministratorPassword: Provide password for the sql database login. It will set the password for the above given user id.
f. databasesName: Provide SQL database name.
g. transperantDataEncryption: It allows to enable the encryption. Enter either 1 or 2.
h. capacity: Capacity refers to the DTU. Minimum allowed value for it is 900 and maximum allowed is 54000.
i. databaseCollation: Enter values for database collation. i.e., SQL_Latin1_General_CP1_CI_AS
j. Please refer to the following image for the sample values entered.



k. Once entered, it will start the deployment. Once deployment is completed, it will show screen like shown below with resourceIDs of the resources.



l. All created resources can be viewed under your selected resource group in portal.azure.com.

How did we create this template?

Follow these steps:

1. The template was created in the visual studio using ARM template extension.
    - Please refer to the [Prerequisites steps to create the ARM template in Visual Studio Code] to install it.
2. Now, we used command like arm-storage for storage account template. And update the values based on documentation.
3. We added Azure Databricks template from the given documentation and updated the values in it.
4. We also added Azure SQL database deployment template from the Microsoft Documentation and updated the values in it..
5. e then parameterized the names of the resources, and created variables for api version. Please refer to the section [Parameterize the ARM Template] for more details on how to create parameters/variables.

Resource Parameter Recommendation

Resource	Parameter	Suggeestions	Recommendation	Reason
Azure Databricks	Workspace Name	NA	Keep <workspacename>-<EnvironmentName> convention.	NA
Azure Databricks	Pricing Tier	1. Standard 2. Premium 3. Trial	Standard	If Standard is cost optimized tier. If Role based access is required then go for Premium.
Azure Databricks	Location	1. East US 2. East US2 3. West US 4. West US2	Keep the location same as resource group's location	NA

Storage Account	StorageAccountName	NA	The name must be unique across all existing storage account names in Azure. It must be 3 to 24 characters long, and can contain only lowercase letters and numbers.	Try to use numbers in between alphabets to make it unique
Storage Account	StorageAccountVersion	1. StorageV2 2. Storage	StorageV2	It combines the older blob functionalities with hierarchical name space functions of newly launched Azure Storage.
Storage Account	StorageAccountSKUName	1. Standard_LRS 2. Premium_LRS 3. Standard_GRS 4. Premium_GRS	Standard_LRS -> if data privacy is very high and data can not be stored on global level data centers. Standard_GRS -> to store replica of data on global level	Please refer to the link
Storage Account	StorageAccountSKUTier	1. Standard 2. Premium	Standard	Please refer to the link
SQL Server	sqlServerName	NA	Keep <workspacename>-<EnvironmentName> convention.	NA
SQL Server	sqlAdministratorLogin	NA	Enter proper user id for SQL server authentication. It should not be admin.	NA
SQL Server	sqlAdministratorPassword	NA	Enter password containing Uppercase, Lowercase, Symbols, Numbers having	

			length of minimum 8 and maximum 64	
SQL Server	databasesName	NA	Keep <databasename>-<EnvironmentName> convention. With maximum length of 63. It should contain only alphanumerics and hyphens.	
SQL Server	transparentDataEncryption	1. Enabled 2. Disabled	1. Enabled	Recommended for data safty.
SQL Server	capacity	From 900 to 54000	Select based on requirement.	Please refer to the link
SQL Server	databaseCollation	SQL_Latin1_General_CP1_CI_AS	SQL_Latin1_General_CP1_CI_AS	NA

Existing ARM Templates (MLOps) Mapping with Portal Parameters

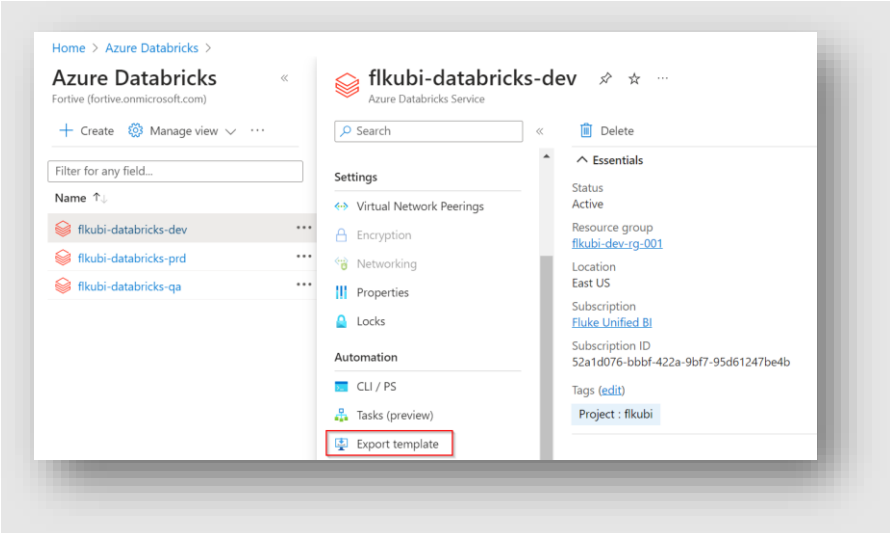
For MLOps, Statistical/Deep Learning/Azure AI services based model majorly uses following resources.

1. Azure Databricks
2. Azure Storage
3. Azure Cognitive Search Services
4. Azure Open AI Services.
5. Azure Web App Services
6. Azure Data Factory Services (if data-pull is integrated)

Azure Databricks

Azure Databricks repository can not be maintained using ARM templates. As the repository contains notebooks and other codes, which can not be maintained using ARM template directly. ARM template can provide option to deploy a ADB workspace but information regarding notebooks are not captured in it.

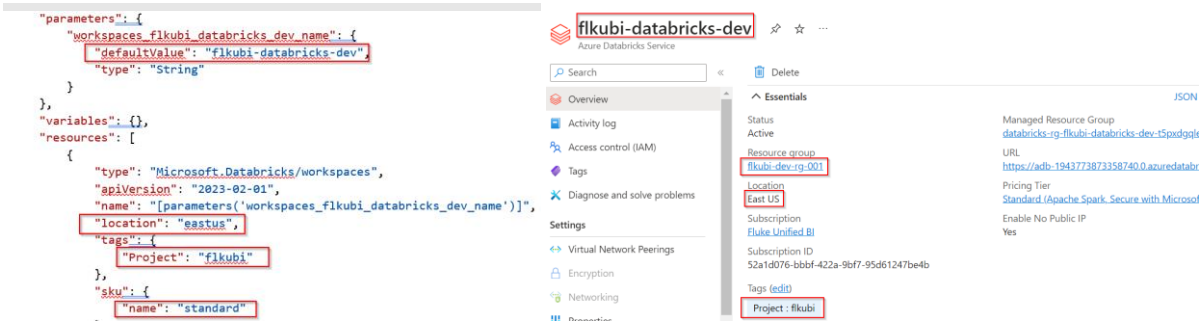
Azure Databricks ARM template can be exported like below



Sample ARM template for Databricks is given below

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploy-
mentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "workspaces_flkubi_databricks_dev_name": {
            "defaultValue": "flkubi-databricks-dev",
            "type": "String"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Databricks/workspaces",
            "apiVersion": "2023-02-01",
            "name": "[parameters('workspaces_flkubi_databricks_dev_name')]",
            "location": "eastus",
            "tags": {
                "Project": "flkubi"
            },
            "sku": {
                "name": "standard"
            },
            "properties": {
                "managedResourceGroupId": "[concat('/subscriptions/52a1d076-bbbf-
422a-9bf7-95d61247be4b/resourceGroups/databricks-rg-', parameters('work-
spaces_flkubi_databricks_dev_name'), '-t5pxdgqlejimc')]",
                "parameters": {
                    "enableNoPublicIp": {
                        "type": "Bool",
                        "value": true
                    },
                    "natGatewayName": {
                        "type": "String",
                        "value": "nat-gateway"
                    },
                    "prepareEncryption": {
                        "type": "Bool",
                        "value": false
                    },
                    "publicIpName": {
                        "type": "String",
                        "value": "nat-gw-public-ip"
                    },
                    "requireInfrastructureEncryption": {
```

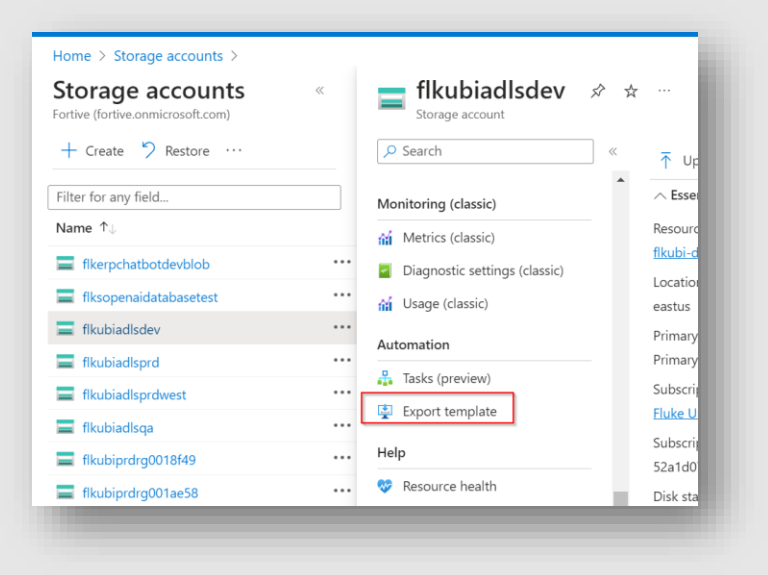
ARM Template Explanation



```
"parameters": {
    "workspaces_flkubi_databricks_dev_name": {
        "defaultValue": "flkubi-databricks-dev",
        "type": "String"
    }
},
"variables": {},
"resources": [
    {
        "type": "Microsoft.Databricks/workspaces",
        "apiVersion": "2023-02-01",
        "name": "[parameters('workspaces_flkubi_databricks_dev_name')]",
        "location": "eastus",
        "tags": {
            "Project": "flkubi"
        },
        "sku": {
            "name": "standard"
```

Azure Storage

Similar to Azure Databricks, Azure Storage as a resource can be deployed using ARM template. But the files that it contains, it can not be deployed or tracked via ARM templates.

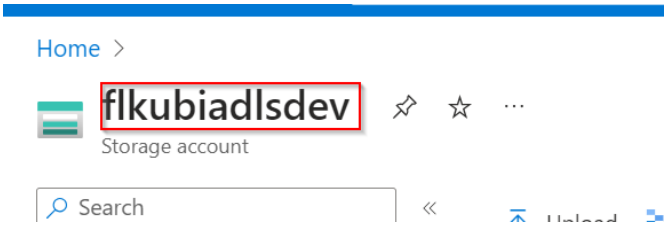


Sample storage ARM template is attached below.

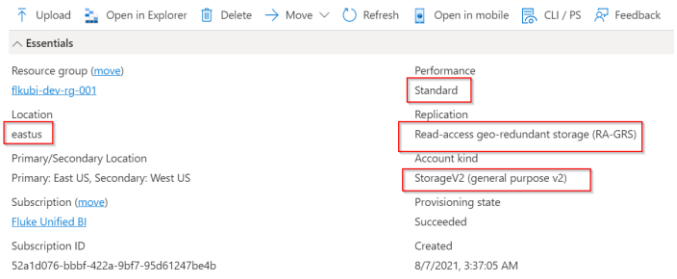
```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccounts_flkubiadlsdev_name": {
      "defaultValue": "flkubiadlsdev",
      "type": "String"
    },
    "factories_flkubi_adf_dev_externalid": {
      "defaultValue": "/subscriptions/52a1d076-bbbf-422a-9bf7-95d61247be4b/resourceGroups/flkubi-dev-rg-001/providers/Microsoft.DataFactory/factories/flkubi-adf-dev",
      "type": "String"
    },
    "virtualNetworks_flkubi_mgd_vnet_001_externalid": {
      "defaultValue": "/subscriptions/52a1d076-bbbf-422a-9bf7-95d61247be4b/resourceGroups/flkubi-mgd-rg-001/providers/Microsoft.Network/virtualNetworks/flkubi-mgd-vnet-001",
      "type": "String"
    },
    "virtualNetworks_flukebi_use_vnet_01_externalid": {
      "defaultValue": "/subscriptions/52a1d076-bbbf-422a-9bf7-95d61247be4b/resourceGroups/flukebi-net-hub-001/providers/Microsoft.Network/virtualNetworks/flukebi-use-vnet-01",
      "type": "String"
    }
  },
  "variables": {},
  "resources": [
    {
```

ARM Template Explanation

```
"parameters": {
  "storageAccounts_flkubiadlsdev_name": {
    "defaultValue": "flkubiadlsdev",
    "type": "String"
  },
  "factories_flkubi_adf_dev_externalid": {
    "defaultValue": "/subscriptions/52a1d076-bbbf-422a-9bf7-95d61247be4b/resourceGroups/flkubi-dev-rg-001/providers/Microsoft.DataFactory/factories/flkubi-adf-dev",
    "type": "String"
  },
```

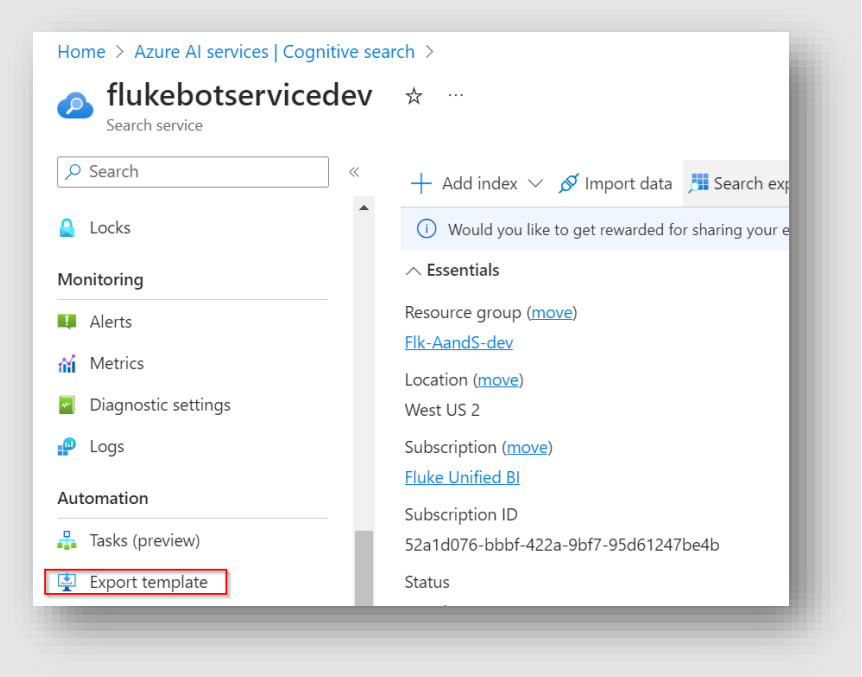



```
"type": "Microsoft.Storage/storageAccounts",
"apiVersion": "2023-01-01",
"name": "[parameters('storageAccounts_flkubiadlsdev_name')]",
"location": "eastus",
"sku": {
  "name": "Standard_RAGRS",
  "tier": "Standard"
},
"kind": "StorageV2",
```



Azure Cognitive Search Services

Azure Cognitive Search Services resource template can also be exported in the same way as shown below.

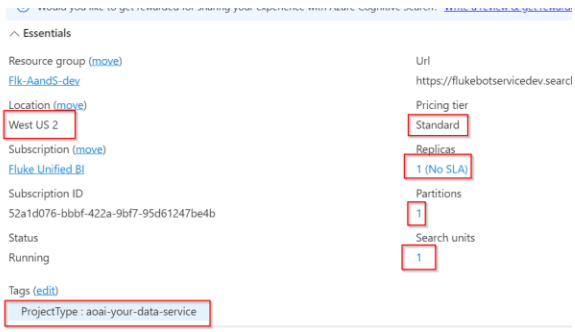


Sample ARM template for Azure Cognitive Search Services is attached below.

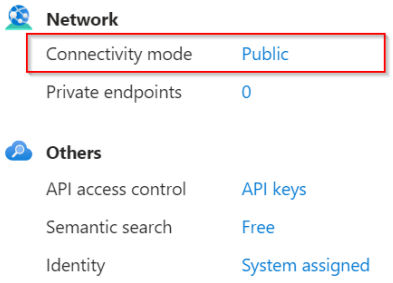
```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "searchServices_flukebotservicedev_name": {
            "defaultValue": "flukebotservicedev",
            "type": "String"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Search/searchServices",
            "apiVersion": "2022-09-01",
            "name": "[parameters('searchServices_flukebotservicedev_name')]",
            "location": "West US 2",
            "tags": {
                "ProjectType": "aoai-your-data-service"
            },
            "sku": {
                "name": "standard"
            },
            "identity": {
                "type": "SystemAssigned"
            },
            "properties": {
                "replicaCount": 1,
                "partitionCount": 1,
                "hostingMode": "Default",
                "publicNetworkAccess": "Enabled",
                "networkRuleSet": {
                    "ipRules": []
                },
                "encryptionWithCmk": {
                    "enforcement": "Unspecified"
                },
                "disableLocalAuth": false,
                "authOptions": {
                    "apiKeyOnly": {}
                }
            }
        }
    ]
}
```

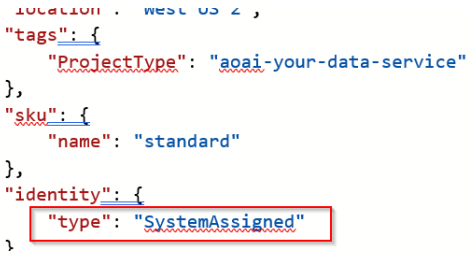
ARM Template Explanation

```
"resources": [
    {
        "type": "Microsoft.Search/searchServices",
        "apiVersion": "2022-09-01",
        "name": "[parameters('searchServices_flukebotservicedev_name')]",
        "location": "West US 2",
        "tags": {
            "ProjectType": "aoai-your-data-service"
        },
        "sku": {
            "name": "standard"
        },
        "identity": {
            "type": "SystemAssigned"
        },
        "properties": {
            "replicaCount": 1,
            "partitionCount": 1,
            "hostingMode": "Default",
            "publicNetworkAccess": "Enabled",
            "networkRuleSet": {
                "ipRules": []
```

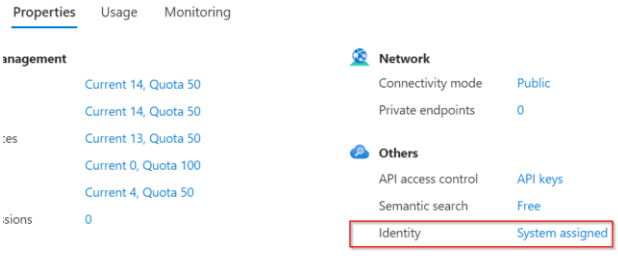


```
    },
    "properties": {
        "replicaCount": 1,
        "partitionCount": 1,
        "hostingMode": "Default",
        "publicNetworkAccess": "Enabled",
        "networkRuleSet": {
            "ipRules": []
        },
        "encryptionWithCmk": {
            "enforcement": "Unspecified"
        },
        "disableLocalAuth": false,
        "authOptions": {
            "apiKeyOnly": {}
        }
```



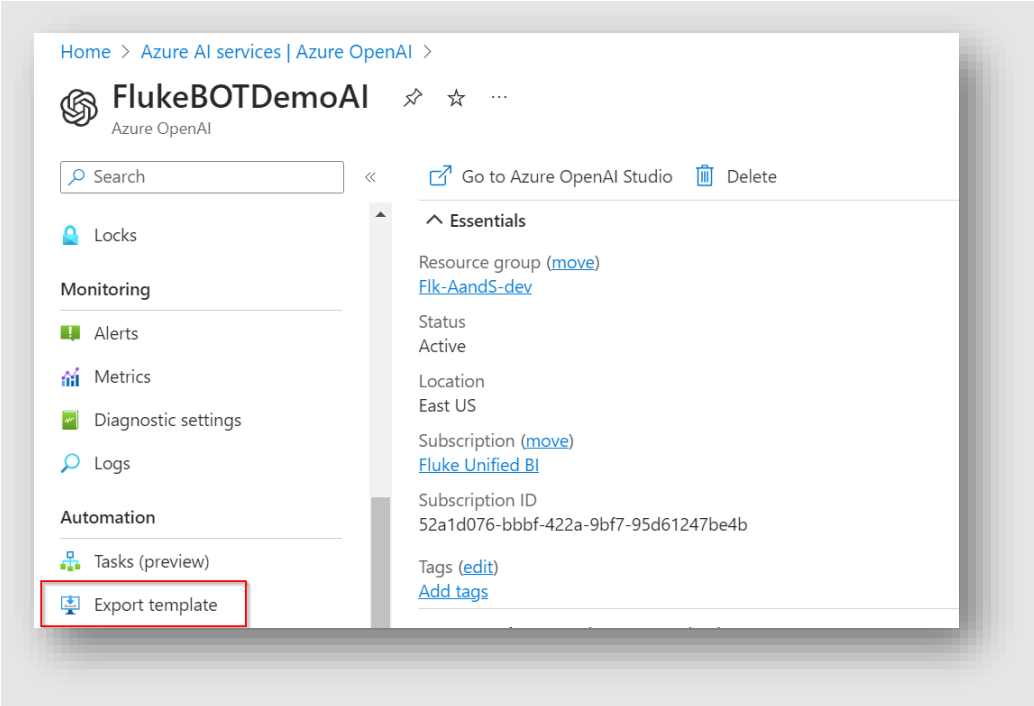


```
"location": "West US 2",
"tags": {
    "ProjectType": "aoai-your-data-service"
},
"sku": {
    "name": "standard"
},
"identity": {
    "type": "SystemAssigned"
}
```



Azure Open AI Services

Azure Open AI resource ARM template can be exported from the website. It contains details about Open AI resource along with pre-deployed models. ARM template can be deployed to a particular environment and pre-deployed GPT model deployment can also be provisioned with same name.

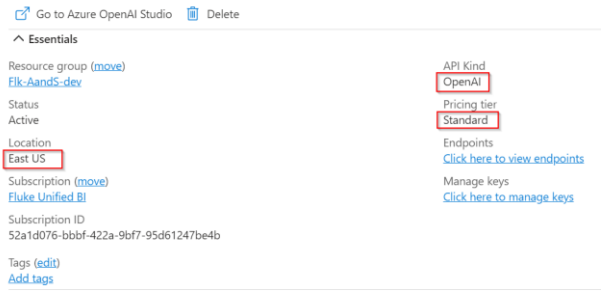


Sample Open AI ARM template is attached below.

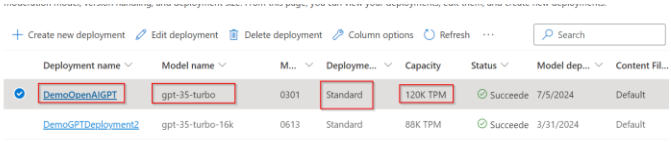
```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "accounts_FlukeBOTDemoAI_name": {
            "defaultValue": "FlukeBOTDemoAI",
            "type": "String"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.CognitiveServices/accounts",
            "apiVersion": "2023-05-01",
            "name": "[parameters('accounts_FlukeBOTDemoAI_name')]",
            "location": "eastus",
            "sku": {
                "name": "S0"
            },
            "kind": "OpenAI",
            "properties": {
                "customSubDomainName": "flukebotdemoai",
                "networkAcls": {
                    "defaultAction": "Allow",
                    "virtualNetworkRules": [],
                    "ipRules": []
                },
                "publicNetworkAccess": "Enabled"
            }
        },
        {
            "type": "Microsoft.CognitiveServices/accounts/deployments",
            "apiVersion": "2023-05-01",
            "name": "[concat(parameters('accounts_FlukeBOTDemoAI_name'), '/DemoGP
TDeployment2')]",
            "dependsOn": [
                "[resourceId('Microsoft.CognitiveServices/accounts', parameters('
accounts_FlukeBOTDemoAI_name'))]"
            ],
            "sku": {
                "name": "Standard",
                "capacity": 88
            },
            "properties": {
```

ARM Template Explanation

```
"resources": [
    {
        "type": "Microsoft.CognitiveServices/accounts",
        "apiVersion": "2023-05-01",
        "name": "[parameters('accounts_FlukeBOTDemoAI_name')]",
        "location": "eastus",
        "sku": {
            "name": "S0"
        },
        "kind": "OpenAI",
        "properties": {
            "customSubDomainName": "flukebotdemoai"
            "networkAcls": {
                "defaultAction": "Allow",
                "virtualNetworkRules": [],
                "ipRules": []
            },
            "publicNetworkAccess": "Enabled"
        }
    },
    {
```

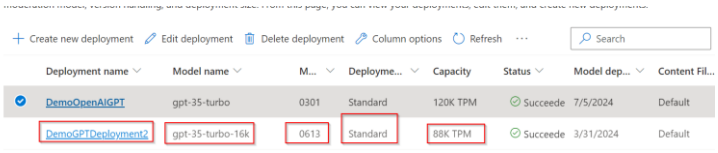



```
        {
            "type": "Microsoft.CognitiveServices/accounts/deployments",
            "apiVersion": "2023-05-01",
            "name": "[concat(parameters('accounts_FlukeBOTDemoAI_name'), '/DemoOpenAIGPT')]",
            "dependsOn": [
                "[resourceId('Microsoft.CognitiveServices/accounts', parameters('accounts_FlukeBOTDemoAI_name'))]"
            ],
            "sku": {
                "name": "Standard",
                "capacity": 120
            },
            "properties": {
                "model": {
                    "format": "OpenAI",
                    "name": "gpt-35-turbo",
                    "version": "0301"
                },
                "versionUpgradeOption": "OnceNewDefaultVersionAvailable",
                "raiPolicyName": "Microsoft.Default"
            }
        }
    ]
```




```
    "sku": {
        "name": "Standard",
        "capacity": 88
    },
    "properties": {
```

```
        "model": {
            "format": "OpenAI",
            "name": "gpt-35-turbo-16k",
            "version": "0613"
        },
```



ARM Template Best Practices

Size Limits

1. Limit the size of your ARM template to 4 MB.
2. Each resource definition within the template should not exceed 1 MB.
3. These limits pertain to the final state of the template, including iterative resource definitions, variable and parameter values.
4. The parameter file also has a 4 MB limit.
5. Smaller templates or parameter files may result in errors if the total request size becomes too large.

Other Limitations

1. 256 parameters
2. 256 variables
3. 800 resources, which includes the copy count for resource repetitions.
4. 64 output values
5. 10 unique locations per subscription, tenant, or management group scope.
6. Templates are limited to 24,576 characters in template expressions.

Exceeding Limits with Nested Templates

1. Some of these limits can be exceeded by utilizing nest templates.
2. For more details on using nested templates, refer to "Using linked and nested templates when deploying Azure resources."

Combining Values to Reduce Limits

1. To stay within these limits, you can combine multiple values into an object, reducing the number of parameters, variables, or outputs.

Validating and Testing ARM Template

Azure provides ARM Template Testing Toolkit for testing the ARM templates. It checks the and validates the code, format of the template, variable restriction are followed or not and performs other sanity tests.

Following is the list of different code sanity checks performed by ARM-TTK.

1. Use correct schema
2. Declared parameters must be used
3. Secure parameters can't have hard-coded default
4. Environment URLs can't be hard-coded
5. Location uses parameter
6. Resources should have location
7. VM size uses parameter
8. Min and max values are numbers
9. Artifacts parameter defined correctly
10. Declared variables must be used
11. Dynamic variable should not use concat
12. Use recent API version
13. Use hard-coded API version
14. Properties can't be empty
15. Use Resource ID functions
16. ResourceId function has correct parameters
17. dependsOn best practices
18. Nested or linked deployments can't use debug
19. Admin user names can't be literal value
20. Use latest VM image
21. Use stable VM images
22. Don't use ManagedIdentity extension
23. Outputs can't include secrets

24. Use protectedSettings for commandToExecute secrets
25. Use recent API versions in reference functions
26. Use type and name in resourceId functions
27. Use inner scope for nested deployment secure parameters

All the details can be checked from this documentation.

To test and validate template manually using ARM TTK,

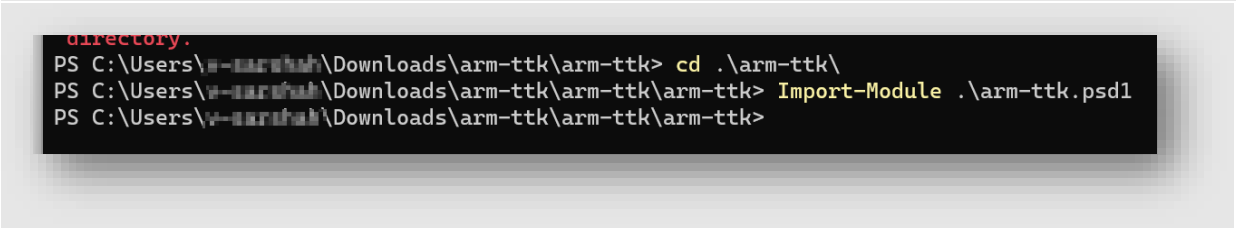
1. Install Power Shell on the system and start the Power Shell.
2. Download the latest zip for the Azure TTK and extract it to a folder.
3. Navigate arm-ttk folder inside tool kit extraction folder.

4. Run following script to unblock the shell from accessing the file scripts.

```
Get-ChildItem *.ps1, *.psd1, *.ps1xml, *.psm1 -Recurse | Unblock-File
```

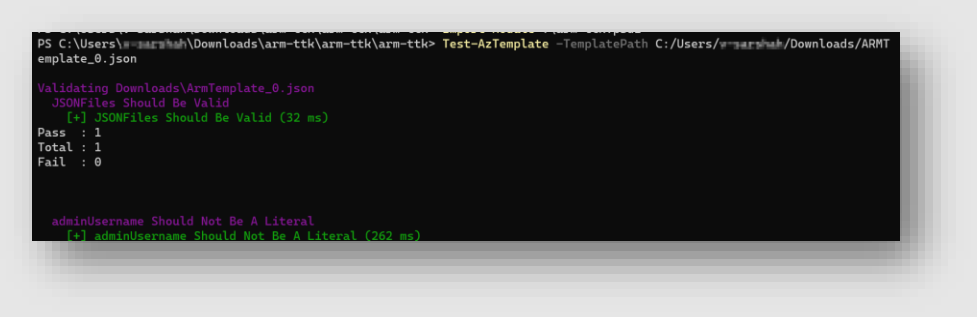
5. Import the module

```
Import-Module .\arm-ttk.psd1
```



6. Run the following command to test the template.

```
Test-AzTemplate -TemplatePath \path\to\template
```



7. Output will be generated like given below.
    a. Pass cases will be green.
    b. Failed cases will be red.
    c. Warnings will be highlighted as Yellow

The output will provide detailed review of the ARM template which can be further analyzed and fixed. Above same steps can be added in the Azure DevOps CI/CD pipeline PowerShell task and the ARM template of the build can be validated in the Azure Pipeline. The original git repo with file present can be referenced for this.

Integrate ARM Template in Azure Pipeline

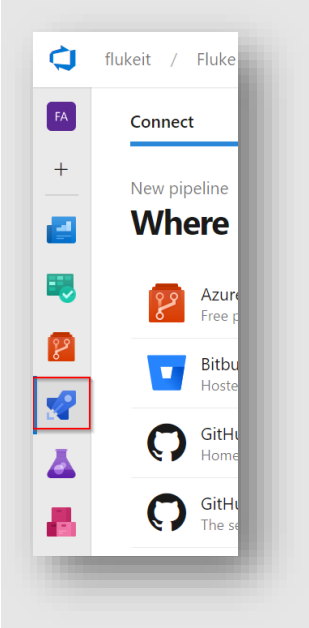
ARM templates can be integrated in Azure CI/CD pipelines for maintenance and versioning of the resource. Some of the resources like Azure Data Factory are fully configured in ARM templates. To integrate the deployment with Azure pipelines, following perquisites needs to be satisfied.

Pre-requisites:

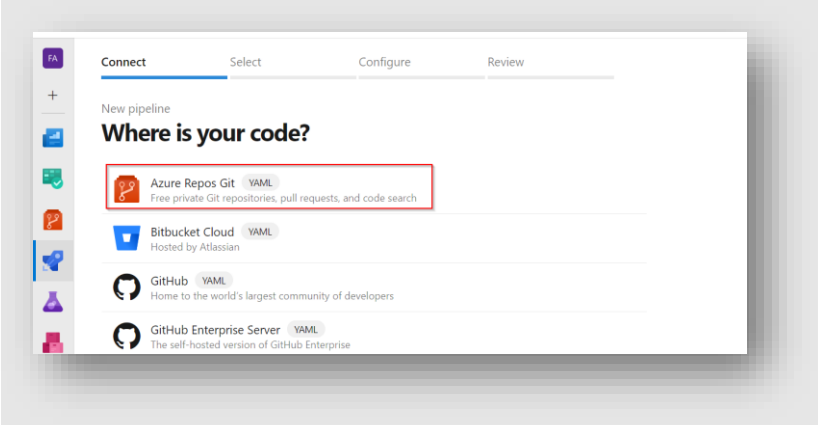
1. Azure repository must created and configured.
2. ARM template must be pushed to repository or power shell command must be used to generate ARM template from the given repo files.
3. ARM template deployment script using power shell must be added in the repository.

Integration:

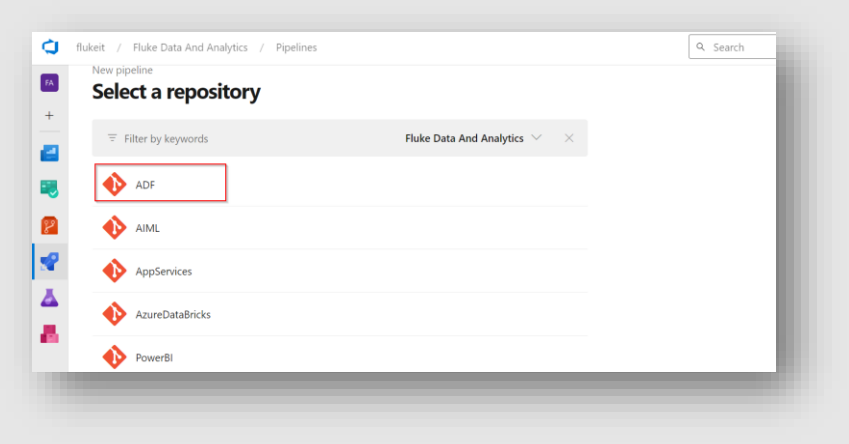
1. Open the pipelines in the Azure DevOps (dev.azure.com).



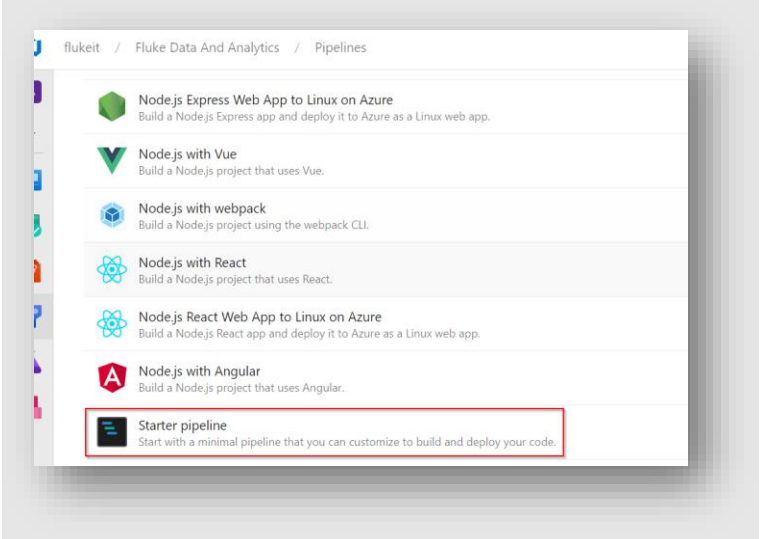
2. In the connect section, select Azure Repository/Azure GIT.



3. Select particular repository having ARM template.



4. Click on Starter pipeline to edit the code manually. It will open up an YAML file having basic code.



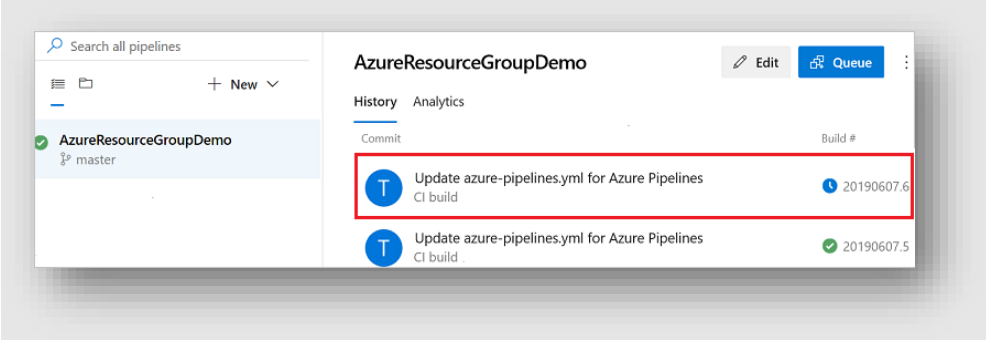
5. ARM template deployment with Power Shell will be used here. Add following sample code.

```
trigger:
- master

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: AzurePowerShell@5
  inputs:
    azureSubscription: 'script-connection'
    ScriptType: 'FilePath'
    ScriptPath: './Deploy-AzTemplate.ps1'
    ScriptArguments: -Location 'centralus' -ResourceGroupName 'demogroup' -
TemplateFile templates\mainTemplate.json
    azurePowerShellVersion: 'LatestVersion'
```

6. Update the subscription name , script argument names based on your script.
7. Save and queue the run of DevOps pipeline.



8. For detailed instructions, please refer to the Microsoft Documentation.

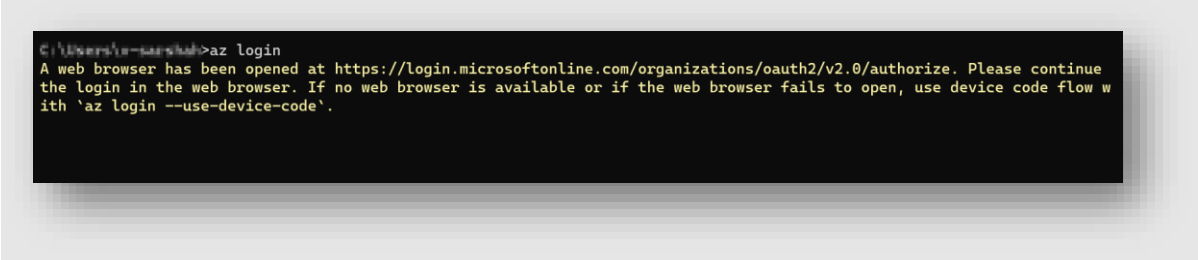
How to Deploy ARM Templates

ARM templates can be deployed with multiple ways.

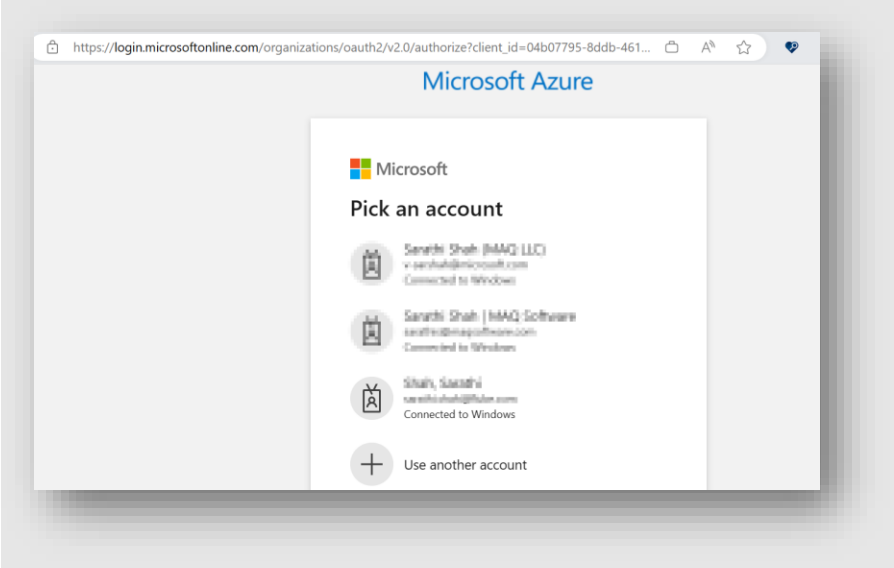
1. Azure CLI
2. Azure Portal – Custom Deployment
3. Azure Power Shell
4. Python Code
5. Rest API
6. Cloud Shell
7. Deploy-to-Azure-Button

Deployment of ARM Template Using Azure CLI

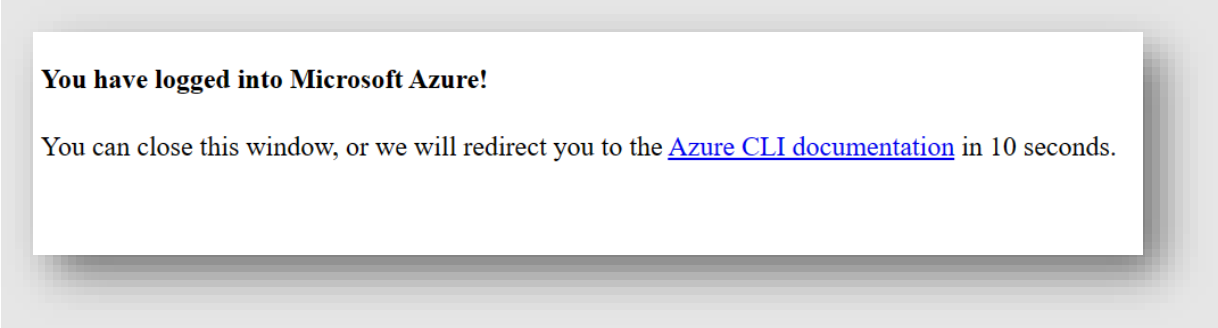
1. Once the ARM template file is ready, save it with appropriate name in the System.
    a. If Azure CLI is not installed, please refer to this Link for Azure CLI installation.
    b. An .msi files needs to be downloaded and executed to install the Azure CLI on windows.
    c. Installation can be verified by running 'az' command in the CMD.
2. Now open the Command Prompt.
3. Now run the command az login in the the CMD.



4. Login with proper ID-Password from this open prompt window.



5. After successful login it will show following message.



6. Now to connect to particular subscription, run `az account set --subscription SubscriptionName`
7. Create a resource group under which the template needs to created. For creating a Resource Group, run the following command.

```
az group create \
  --name myResourceGroup \
  --location 'Central US'
```

8. Once Resource Group is created, start the deployment of the resource. Run the following command to deploy the resource.

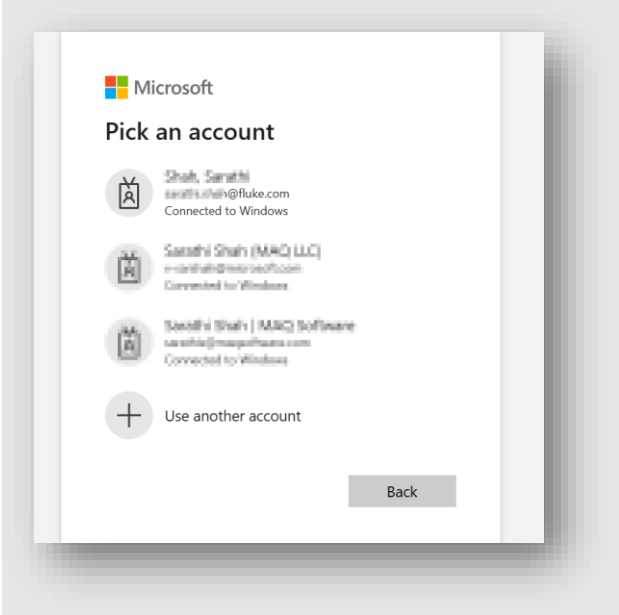
```
templateFile="{provide-the-path-to-the-template-file}"
az deployment group create \
  --name blanktemplate \
  --resource-group myResourceGroup \
  --template-file $templateFile
```

9. **Once the deployment is succeeded, the screen will show up like following.**

Deploy ARM Template using Power Shell

1. Once the ARM template file is ready, save it with appropriate name in the System.
2. To deploy using the Azure Power Shell, Power Shell must be installed in the system. If it is not installed in the system, refer the documentation for the installation
3. Open the Power Shell.
4. Connect to Az account using following command.

5. An authentication window will open. Sign to using relevant credentials.



6. Once logged in, set the deployment scope for resource type to be deployed.

a. To deploy to a **resource group**, use New-AzResourceGroupDeployment.

```
New-AzResourceGroupDeployment -ResourceGroupName <resource-group-name> -TemplateFile <path-to-template>
```

b. To deploy to a **subscription**, use New-AzSubscriptionDeployment

```
New-AzSubscriptionDeployment -Location <location> -TemplateFile <path-to-template>
```

c. To deploy to a **management group**, use New-AzManagementGroupDeployment.

```
New-AzManagementGroupDeployment -Location <location> -TemplateFile <path-to-template>
```

d. To deploy to a **tenant**, use New-AzTenantDeployment.

```
New-AzTenantDeployment -Location <location> -TemplateFile <path-to-template>
```

7. Now create a resource group if required.

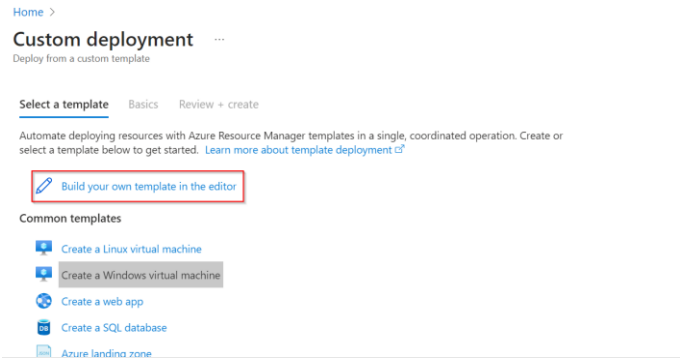
```
New-AzResourceGroup -Name ExampleGroup -Location "Central US"
```

8. Run the following command to deploy the local file.

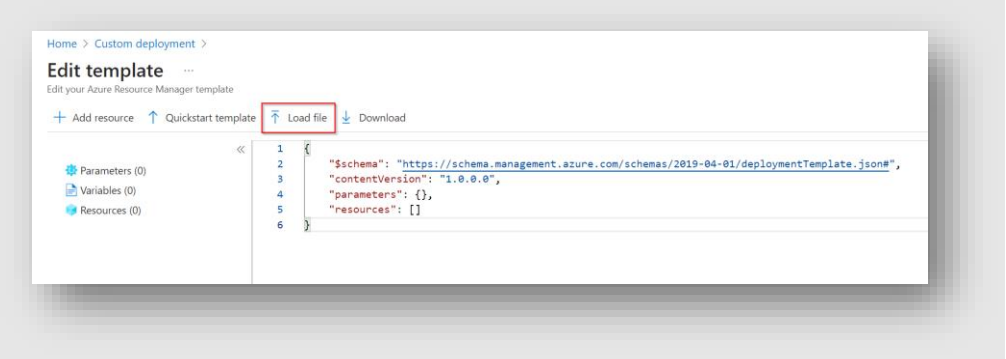
```
New-AzResourceGroupDeployment `
  -Name ExampleDeployment `
  -ResourceGroupName ExampleGroup `
  -TemplateFile <path-to-template>
```

Deploy ARM Template using Azure Portal

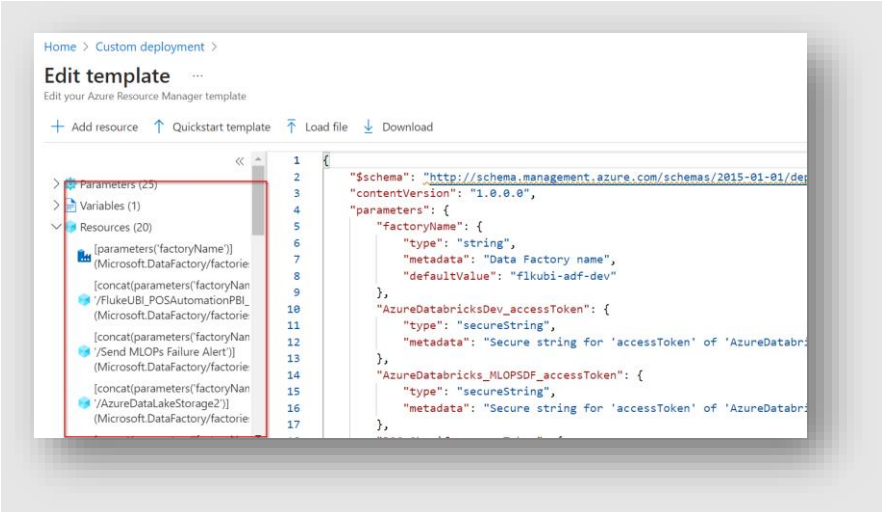
1. To deploy any ARM template via Azure Portal, custom deployment is required.
2. Open the Azure portal and search for Deploy a Custom Template.
3. Click on the Build your own template in the editor



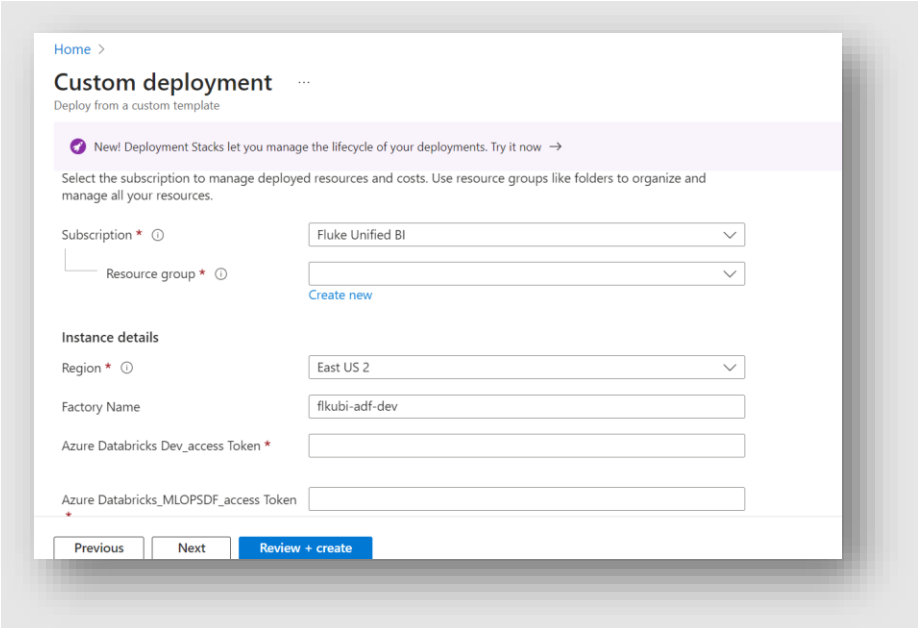
4. Copy your JSON code in the editor shown, or click on the Load File option and select your ARM template JSON file



5. Once the file is selected, all the parameters, resource information can be seen as shown below, click on Save.



6. Now fill the details for Azure Subscription, and other parameters.



7. Once all values are selected, click on Review and Create.

ARM Template Deployment Modes

Incremental and Complete Deployments

1. ARM templates support two deployment types: incremental and complete.
2. These deployment types determine how the Resource Manager service handles resources not defined in the template.

Resource Creation Attempt

1. In both incremental and complete modes, the Resource Manager service attempts to create all resources specified in the template.

Resource Matching and Unchanged State

1. If a resource already exists and matches the template resource, Azure leaves the resource unchanged.
2. Re-running commands from a previous section would not trigger any operations in Azure.

Updating Resource Properties

1. When you modify one or more properties in a resource, Azure updates the resource with the new values.
2. For example, changing the storage SKU from Standard LRS to Standard GRS results in Azure updating the storage account resource.

Deploying Using Complete Mode

Deletion of Unspecified Resources

1. In complete mode, Resource Manager deletes resources that exist in the resource group but are not specified in the template.

Template Condition and REST API Versions

1. If your template includes a resource that isn't deployed due to a condition evaluating to false, the outcome depends on the REST API version used for template deployment.
2. If deploying with a version before 2019-05-10, the resource is not deleted.
3. After 2019-05-10, the resource is no longer available, effectively deleted.
4. In recent versions of Azure PowerShell and Azure CLI, the resource is deleted.

Caution with Complete Mode and Copy Loops

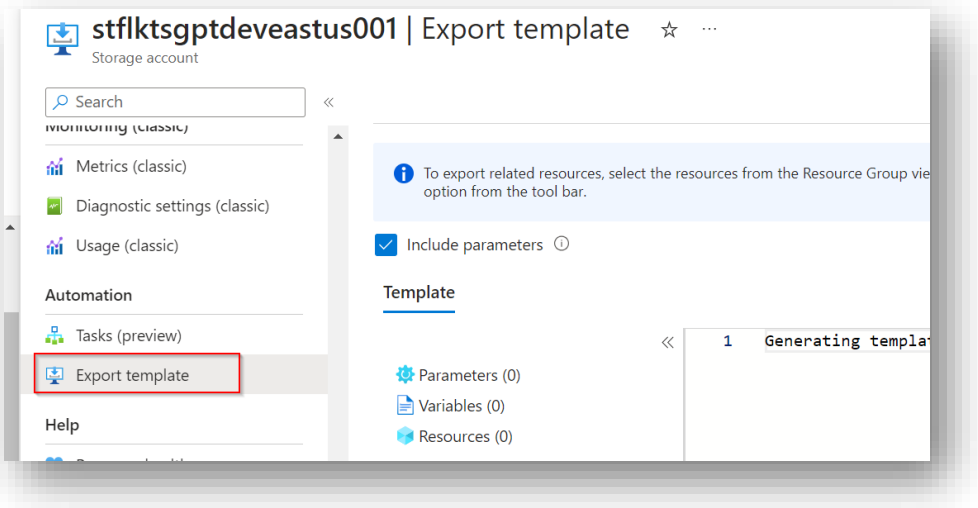
1. When using complete mode in templates with copy loops, exercise caution.
2. After resolving the copy loop, any resources not specified in the template are deleted.

Resource Creation for Fluke Tech Support GPT

Storage Account

Export the Template of Dev Resource

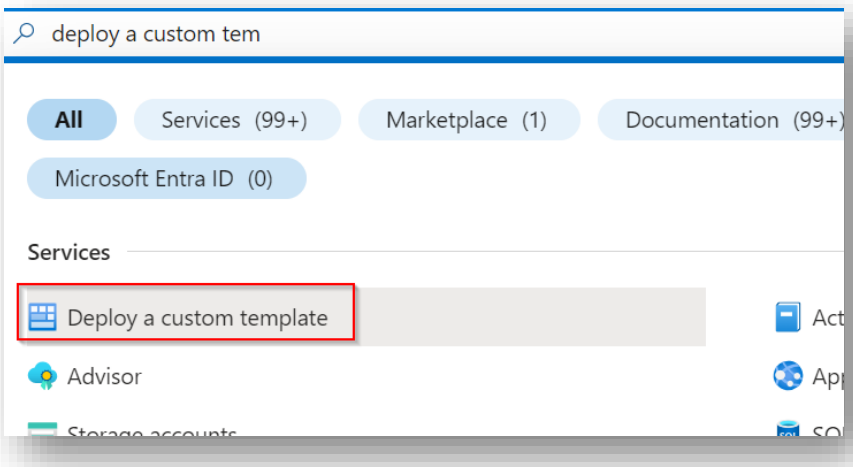
1. Open portal.azure.com
2. Open the storage account and on the menu click on the Export Template option.



3. Replace words related to Dev workspace to UAT / Prod in template and save it in JSON format.

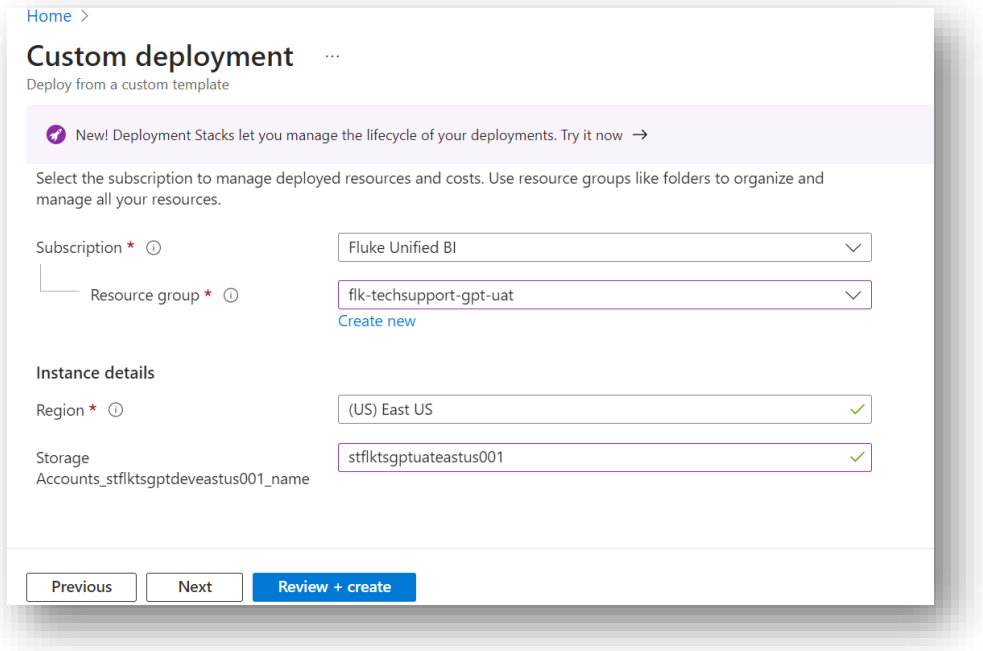
Deploy the Template to UAT Resource Group

1. Go to portal.azure.com and open ‘Deploy a custom template’



2. Click on Build your own template in the editor.

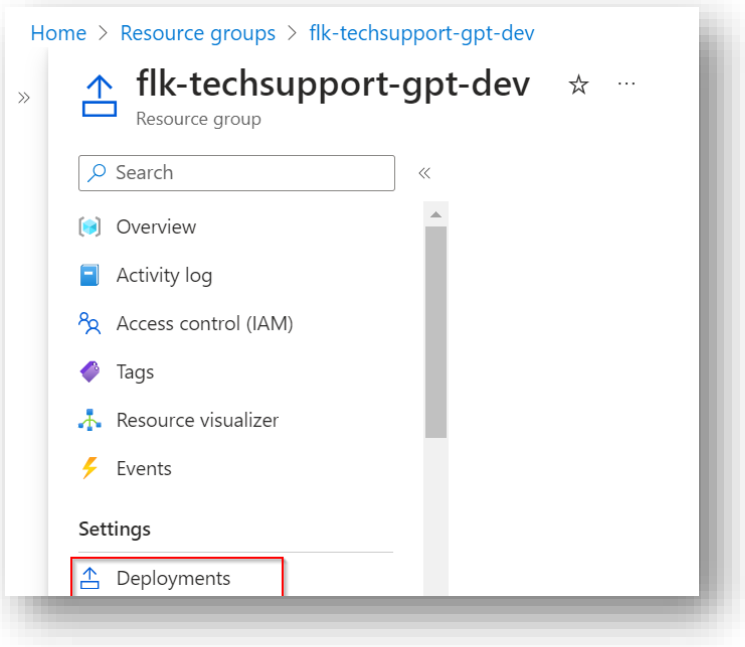
3. Paste the JSON content in the editor and click on save.
4. Select/Update the required parameter values and click on deploy.



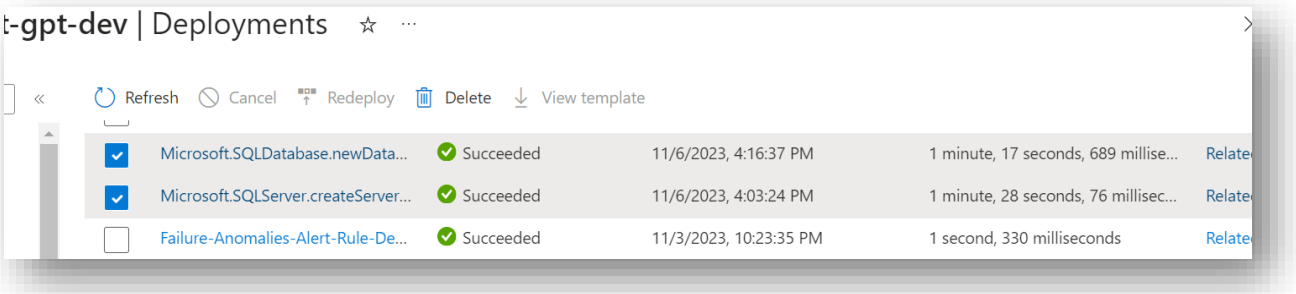
Azure SQL Server And Azure SQL DB

Exporting the ARM Templates

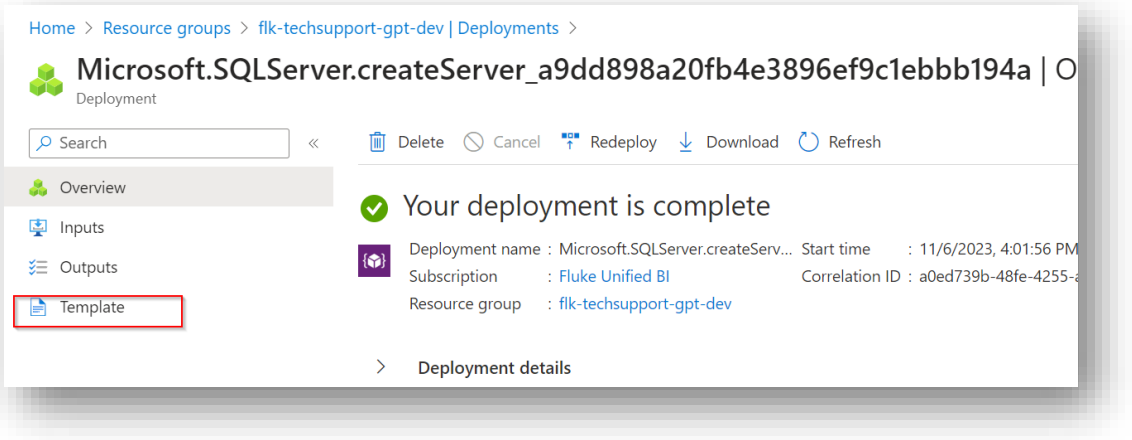
1. For getting the ARM template for this, we will refer to the deployment made while creating the resource on Dev.
2. Go to portal.azure.com, and open the relevant resource group.
3. Click on the Deployment option from the menu on the left as shows below in the image.



4. Now find the exact deployment for Azure SQL DB and Azure SQL Server.



5. Open the SQL Server deployment and click on the template to generate the template



6. Update all “dev” keywords to uat to make it refer to UAT environment and resources.
    a. Update the resource name accordingly.
    b. If any dependsOn statement is used in the ARM template, make sure to verify that that resource for which dependency is there, exists on UAT environment or is getting created in the same ARM template.
7. Save the resource template with .json name.

Follow the same method and save the Azure SQL DB ARM Template.

Deployment of the Resource (Azure SQL Server + Azure SQL DB)

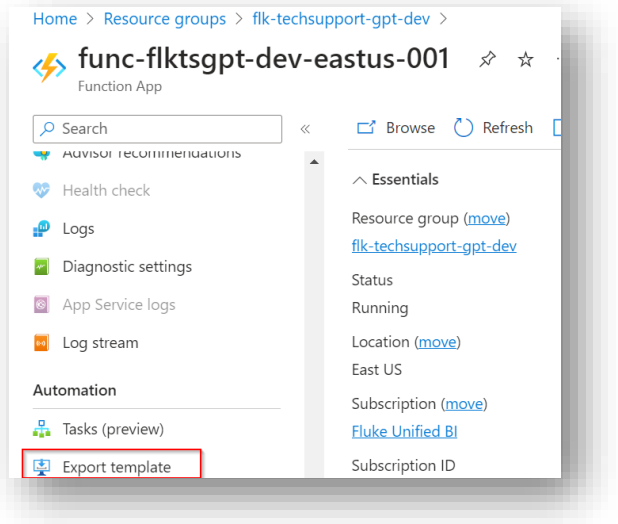
Please refer to the section [Deploy the Template to UAT Resource Group] for deploying the saved resource group.

1. Deploy the Azure SQL Server first.
2. Once SQL Server is deployed, deploy the Azure SQL DB.

Azure Function App

Export the Template:

1. Open portal.azure.com
2. Open the Function App resource and on the menu click on the Export Template option.



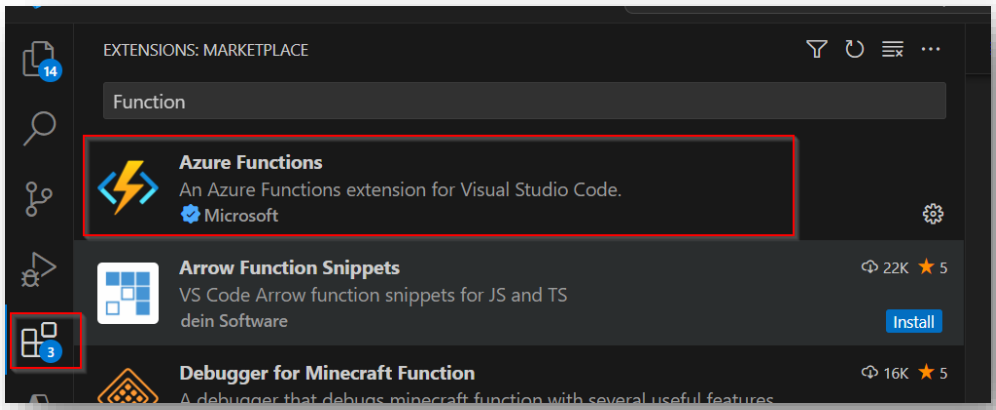
3. Replace words related to Dev workspace to UAT / Prod in template and save it in JSON format.
    a. Verify resource group name and other things are aligned or not.
4. Save the file to .json format.

Deploy the Template

Please refer the to the section [Deploy the Template to UAT Resource Group] for the template deployment.

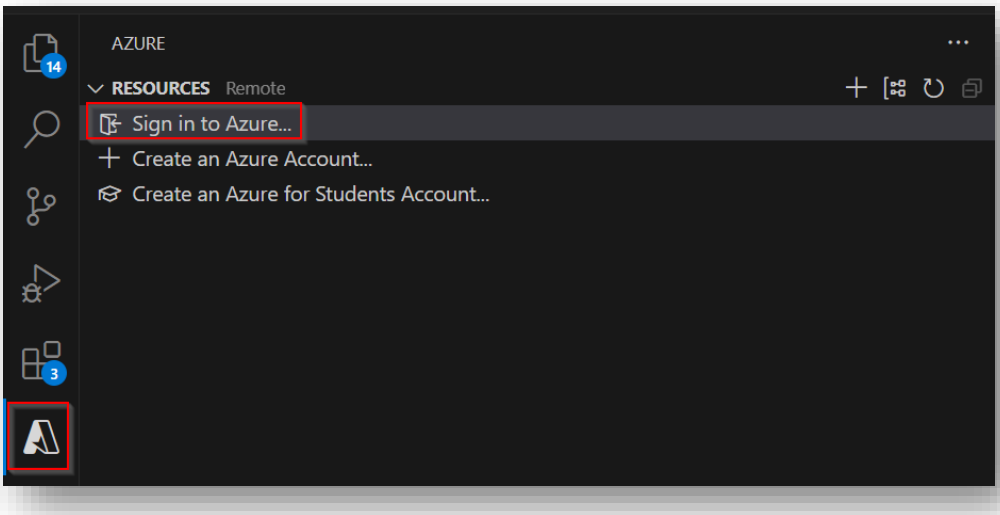
Function App Internal Resource/Function Creation

1. Function creation requires deployment from Visual Studio Code or Azure CLI Functions.
2. For Visual Studio installation, please refer to the steps mention in section [Prerequisites steps to create the ARM template in Visual Studio Code].
3. Install the Azure Functions extension.
    a. Go the the extension in Visual Studio Code and search for Function App.

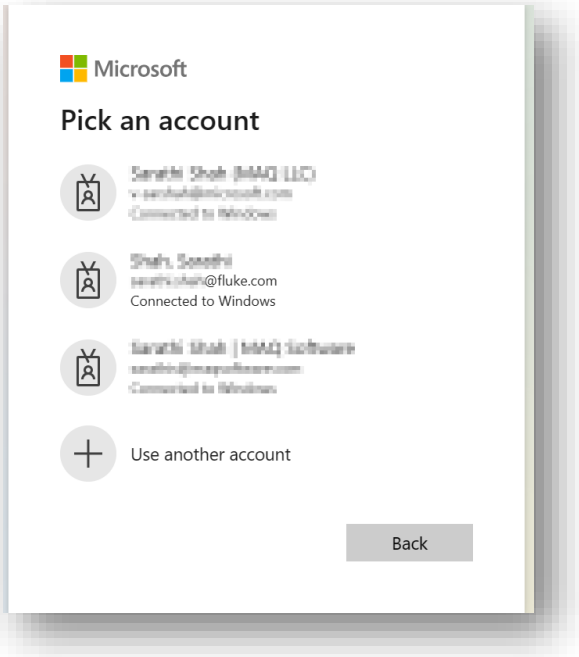


b. Install the extension.

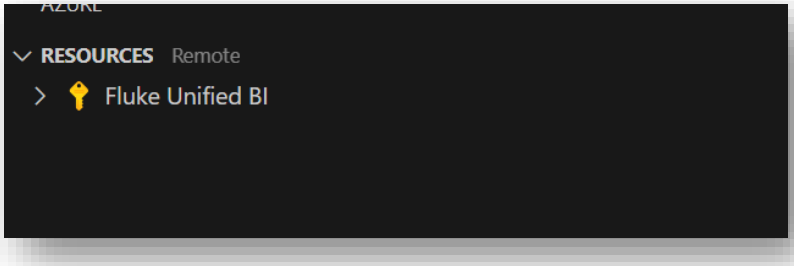
4. Now, click on Azure Icon, and select Sign in to Azure.



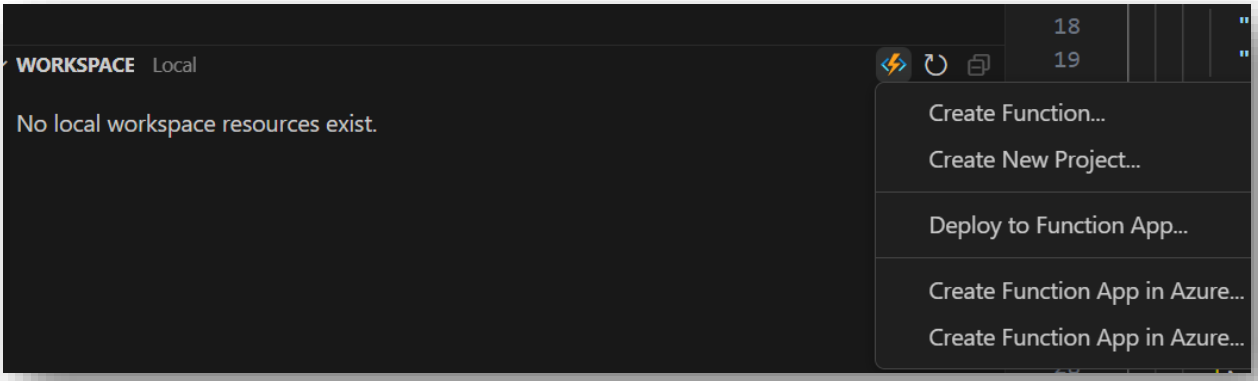
5. Authenticate with the credentials which have access to the resource group where deployment is to be done.



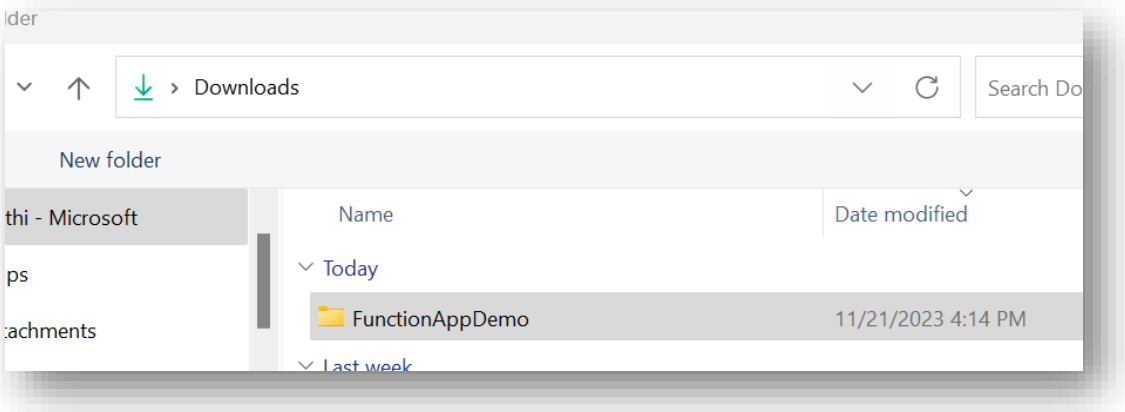
6. Once logged in successfully, all the accessible subscription will be visible.



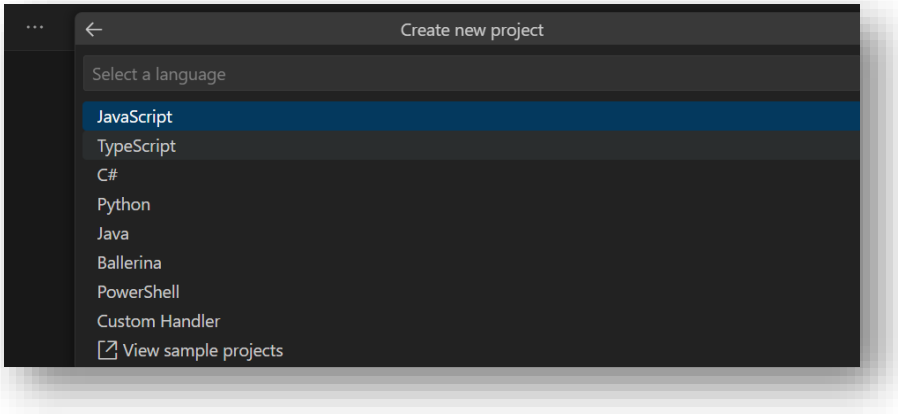
7. Now, click on the function icon as shown below and select create a new function.



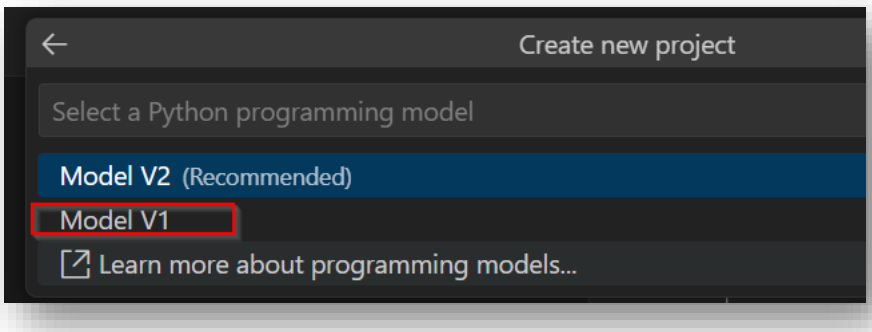
8. Select the folder where you want to create the function.



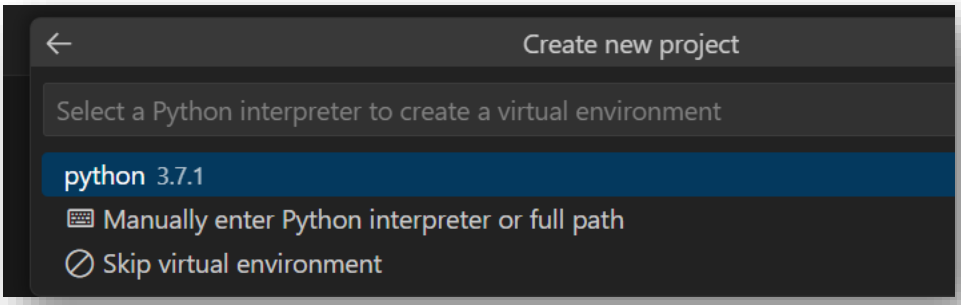
9. Now select the language of the Function App.
    a. The language can be checked from by viewing the function app files.



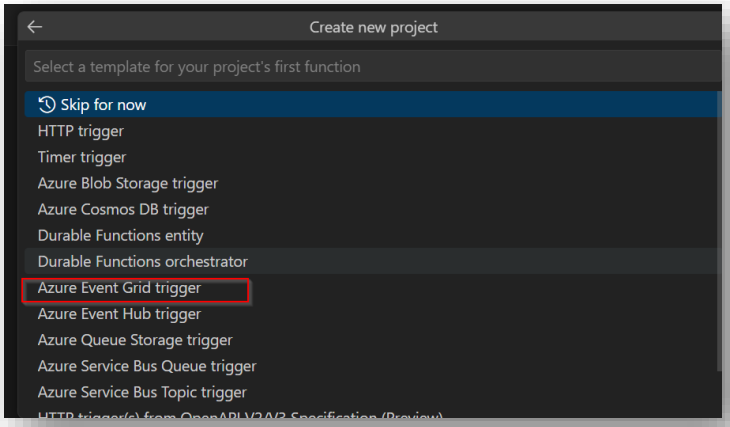
10. Now select the Model in which the original Function App was created.
    a. For both the models, the app structure is different.
    b. We have selected Model V1 as original app function on dev resource was created using Model V1.



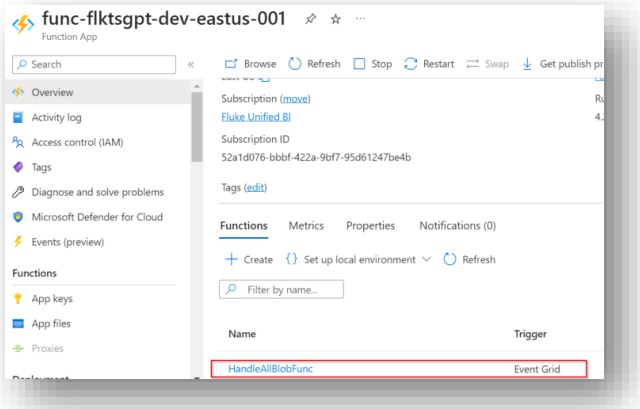
11. Select the python environment.



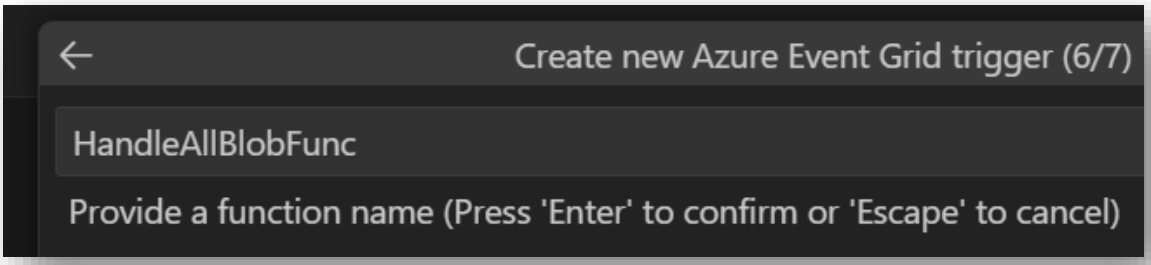
12. Select the trigger for the function app.
    a. We have selected the trigger based on the Dev resource observation.



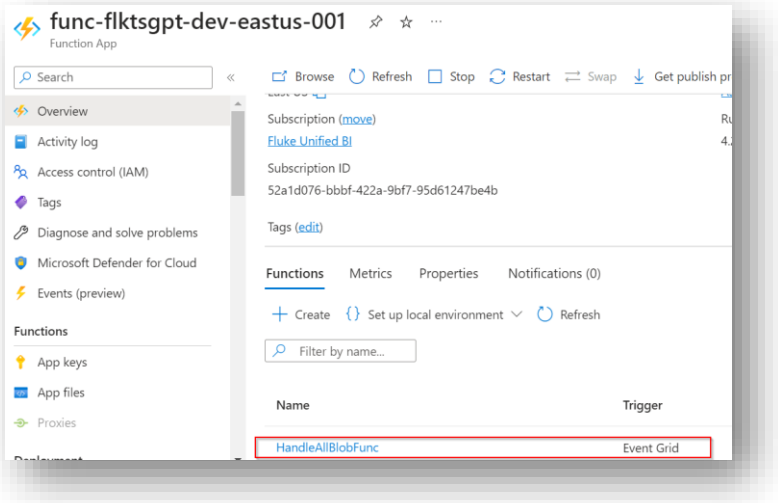
    b. Following image shows the attached trigger.
        i. To check the trigger, click on the Function listed in the Function App.



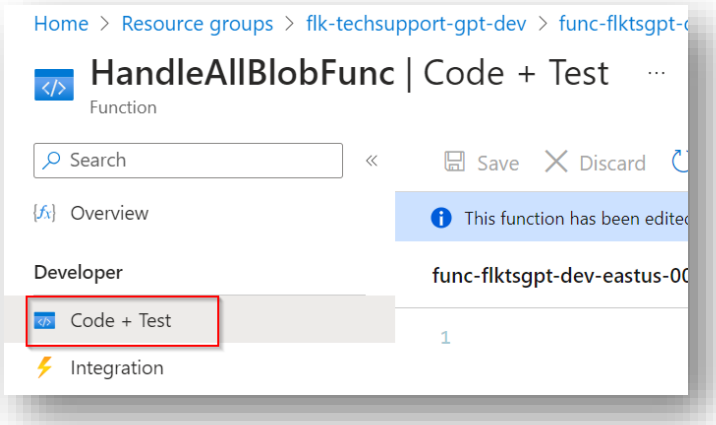
13. Add the function name.



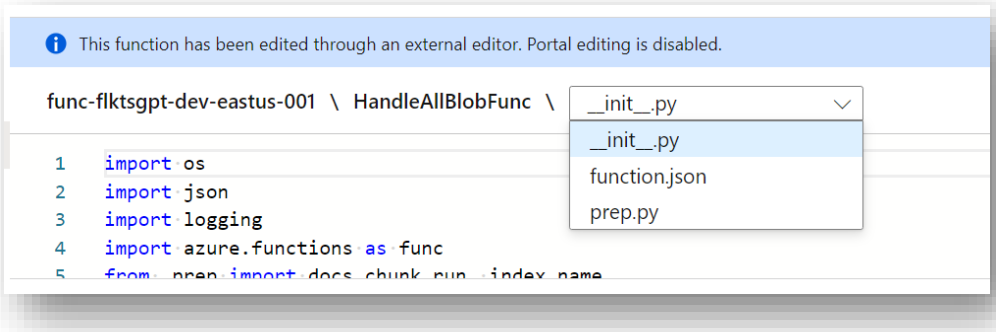
14. Click on Open it in a new window.
15. Now copy the function.json file, __init__.py file code from Dev Function app.
    a. To get the code, click on the Function App.



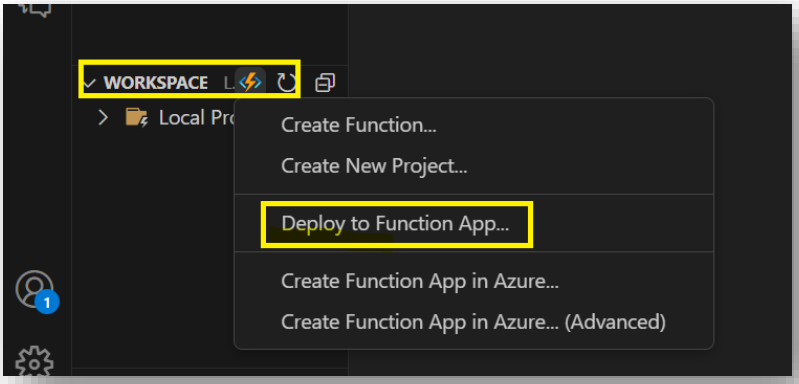
    b. Click on Code + Test



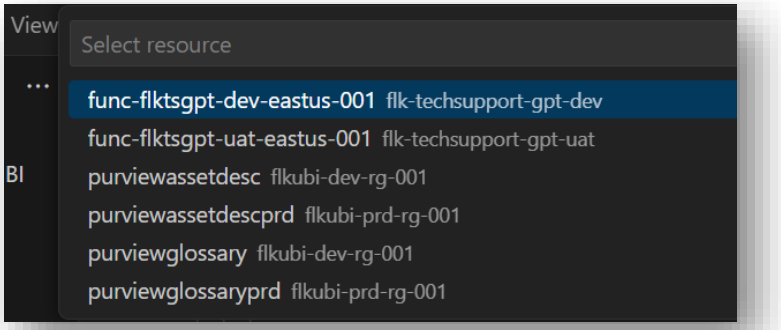
16. Copy the relevant file's code with the same name in visual studio.



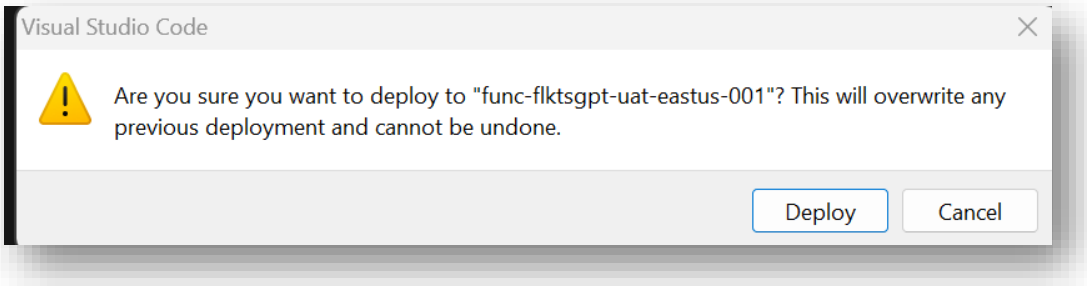
17. Now, click on the Deploy to Azure as shows in the image below.



18. Sign into Azure If required, as explained in steps 4 & 5.
19. Select the Resource Group.



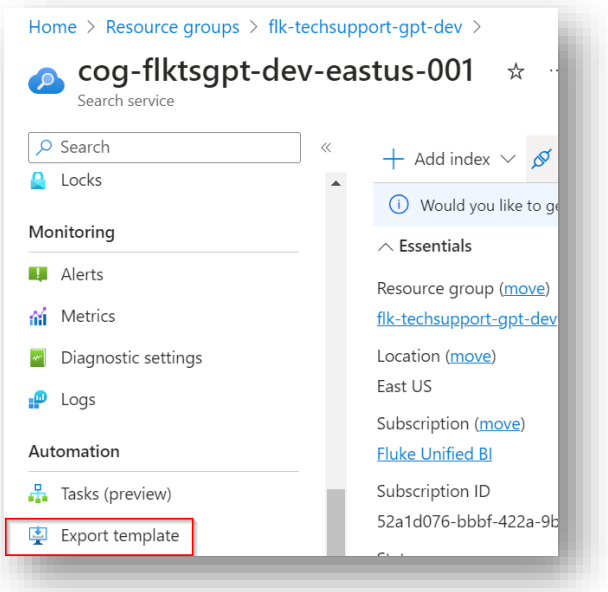
20. Click on deploy



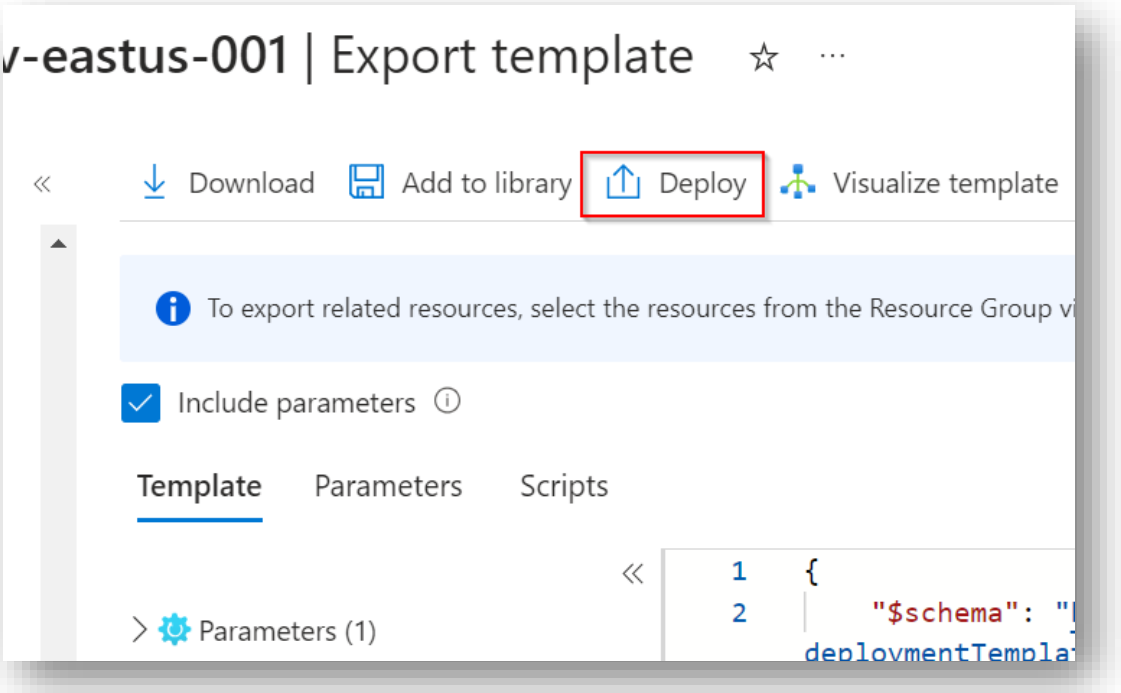
Azure Cognitive Search Services

Export the Template:

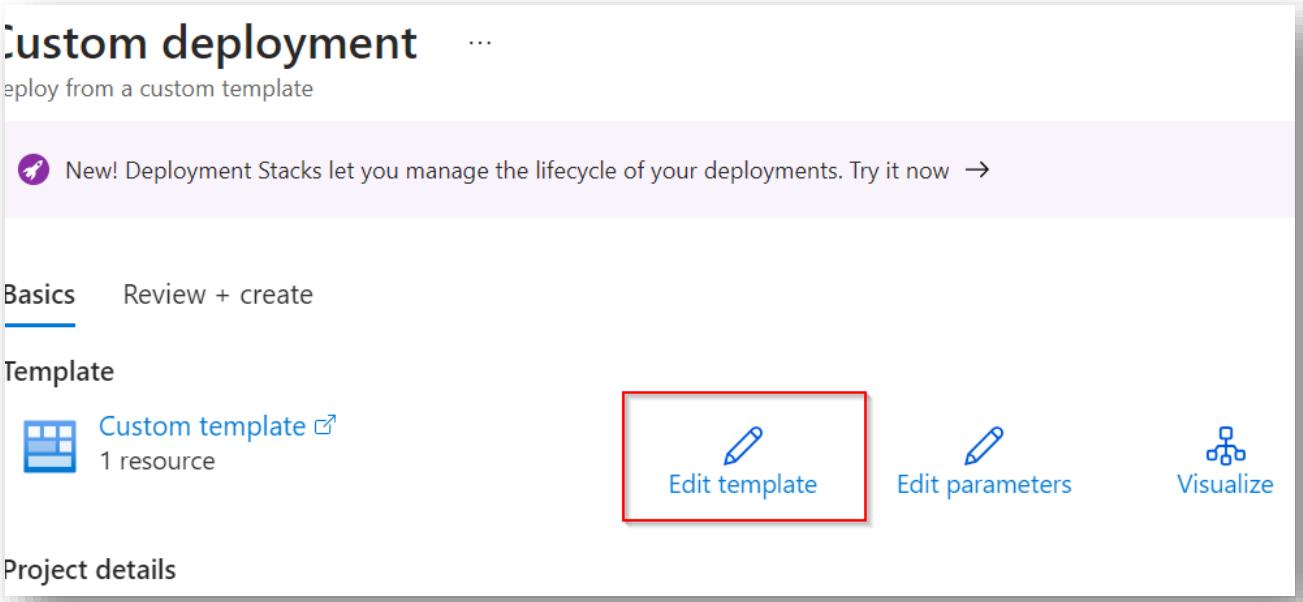
1. Open portal.azure.com and go the Cognitive Search resource.



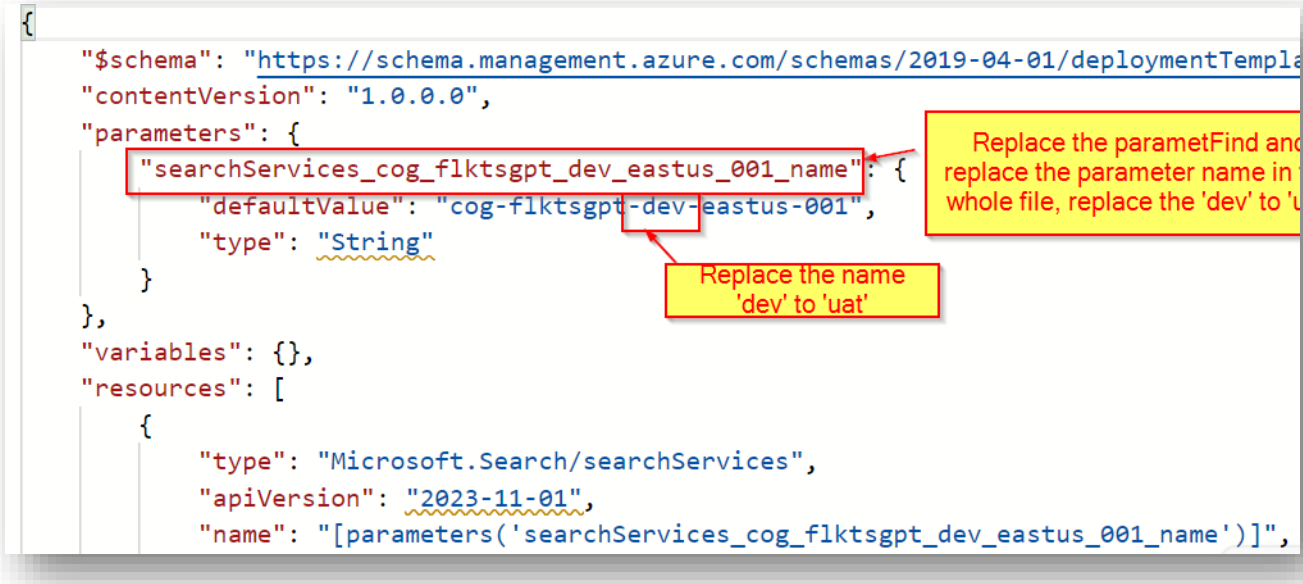
2. Click on Deploy.



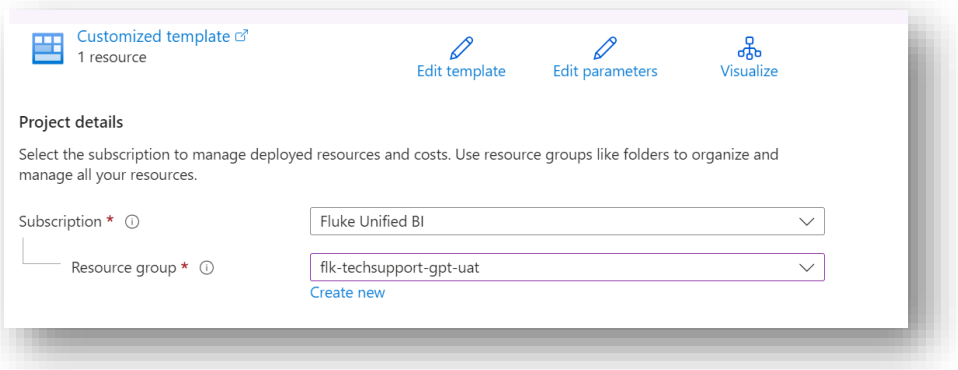
3. Click on the Edit template.



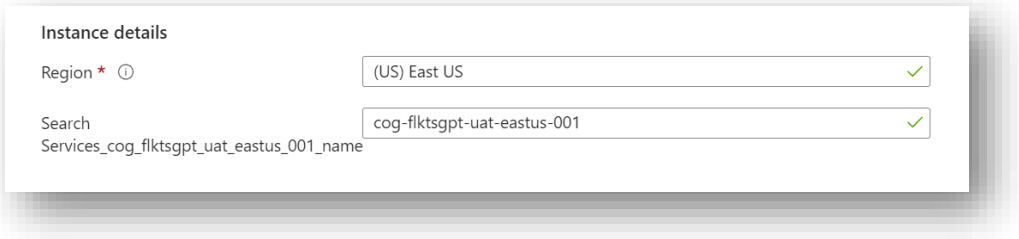
4. Replace all ‘dev’ keyword to ‘uat’ for UAT naming conventions and click on save.



5. Now once selected, select the Subscription, Resource Group for the deployment



6. Now verify the resource name.



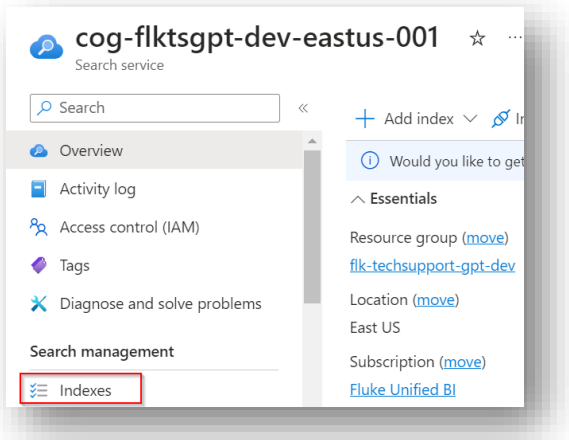
7. Click on Review + Create, and Create the resource.

Migrating Azure Cognitive Search Index

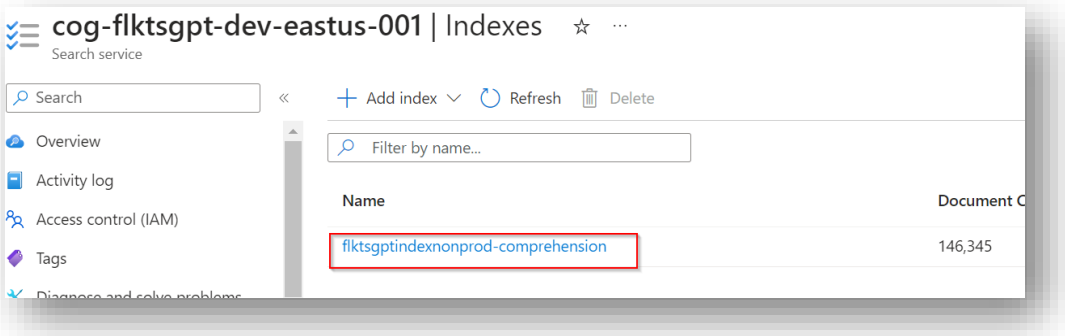
Azure Cognitive Search Index is not considered as resource and therefore not included in the ARM template.

Follow the steps to migrate the index from Dev to UAT manually.

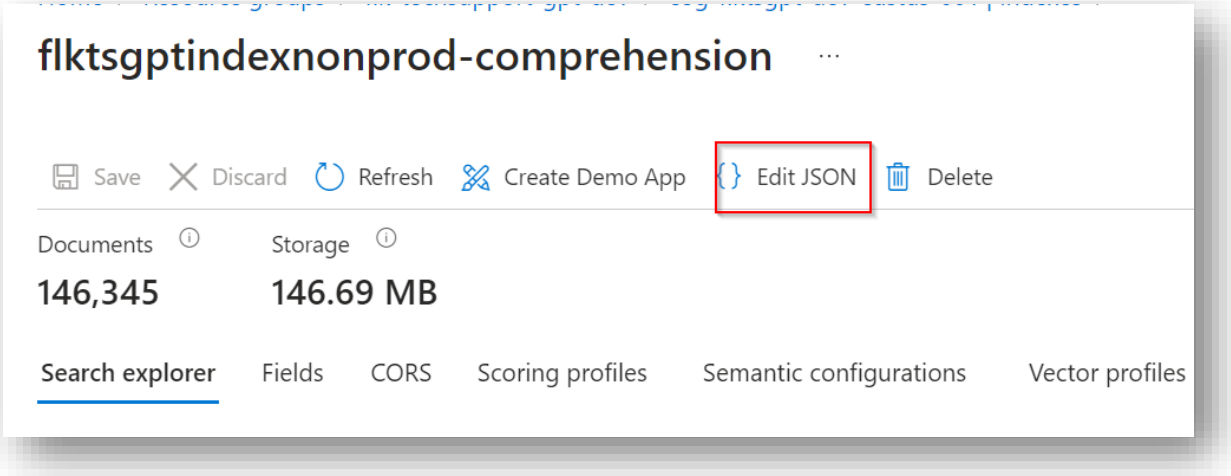
1. In the portal.azure.com, Under the Azure Cognitive Search Dev Resource, go to the Index.



2. Open the index created on the dev resource.



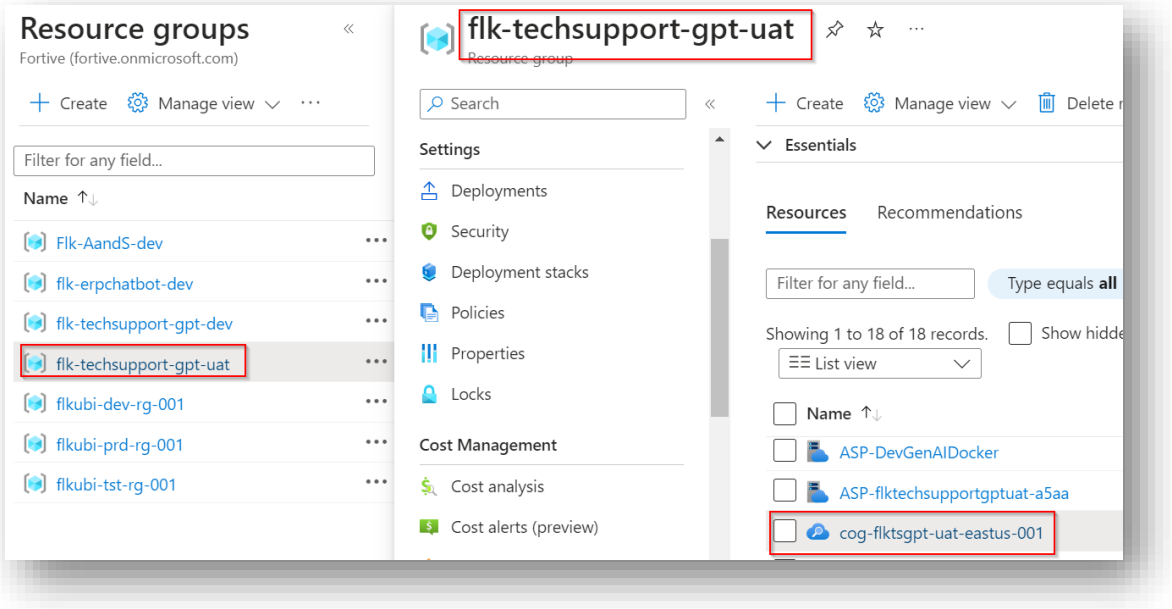
3. Click on the Edit JSON.



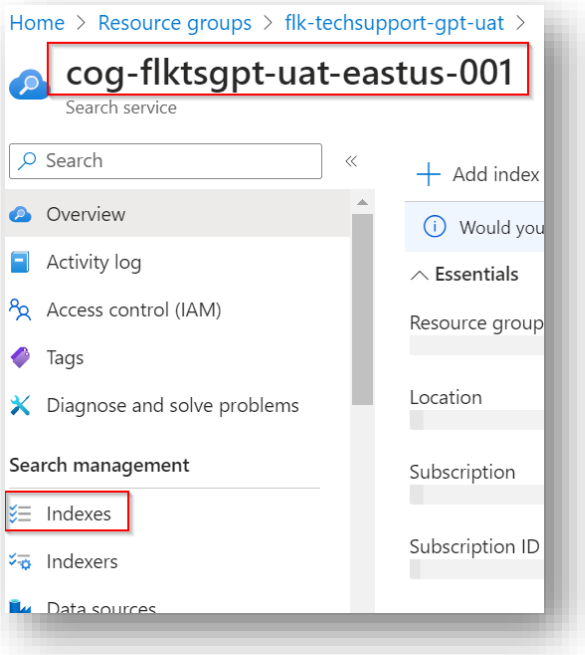
4. Save the JSON definition to a .json file.
5. Open the file and remove '@odata' related lines. Please refer to the image below. There can be more lines.

Deploy the Index Manually

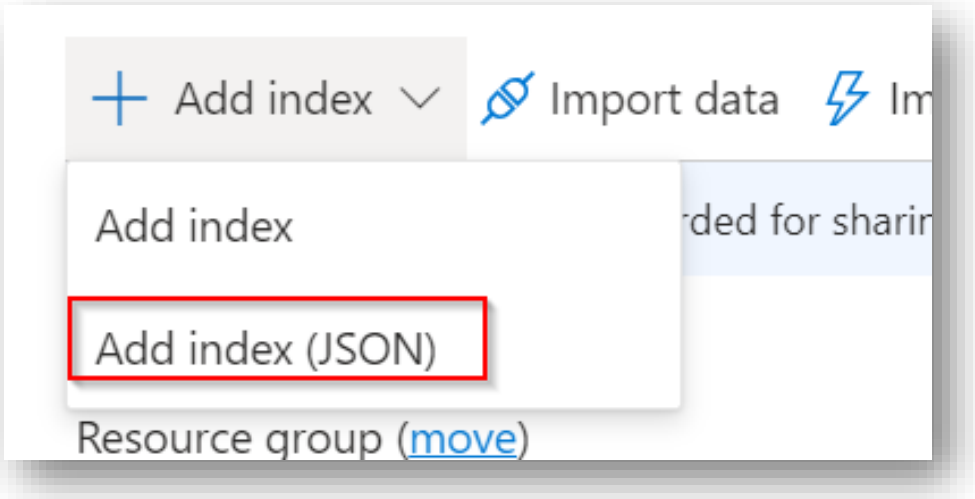
1. Under the portal.azure.com, open the UAT Cognitive Search Service resource.



2. Click on the index from the left menu.



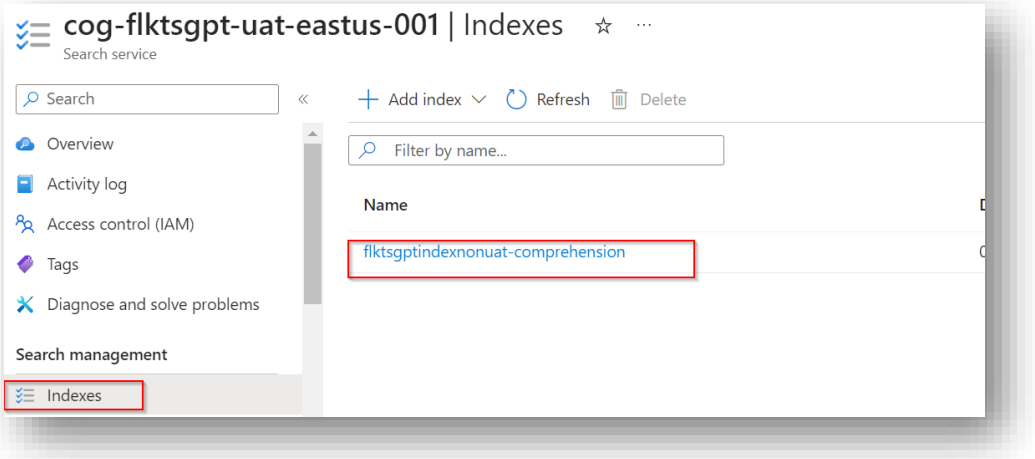
3. Click on the Add Index -> Add Index (JSON).



4. Copy the saved definition to the Index creation editor.

5. Remove the red highlighted parts of the JSON content from the editor. Please refer to the image above.
6. Once removed, it will enable save button at the bottom of the window. Click on Save.

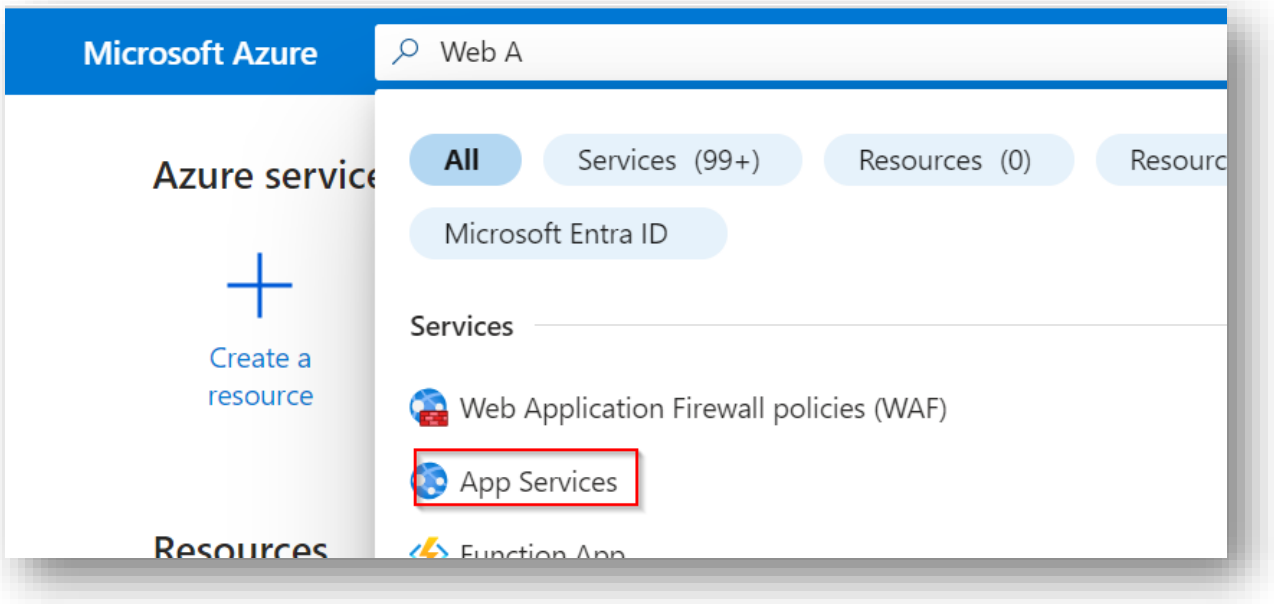
7. Verify the index created from the Index tab of the Cognitive Search Resource.



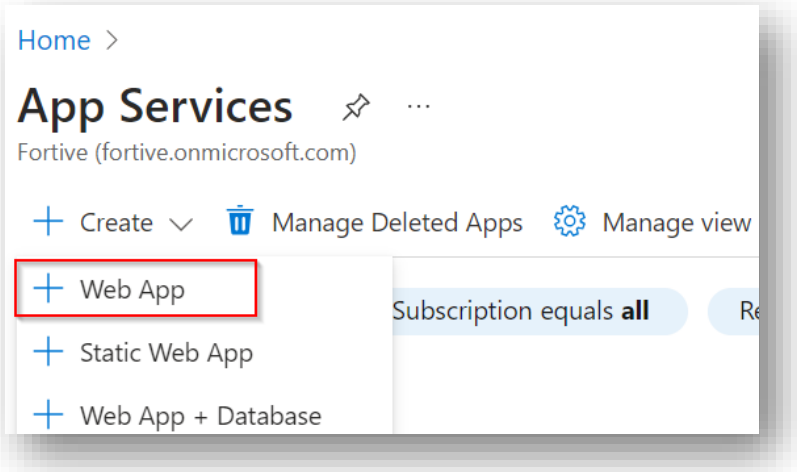
Web Application

Deployment of the Web Application happened based on creation only, which registers a web app.

1. Open portal.azure.com, and go to App Services.

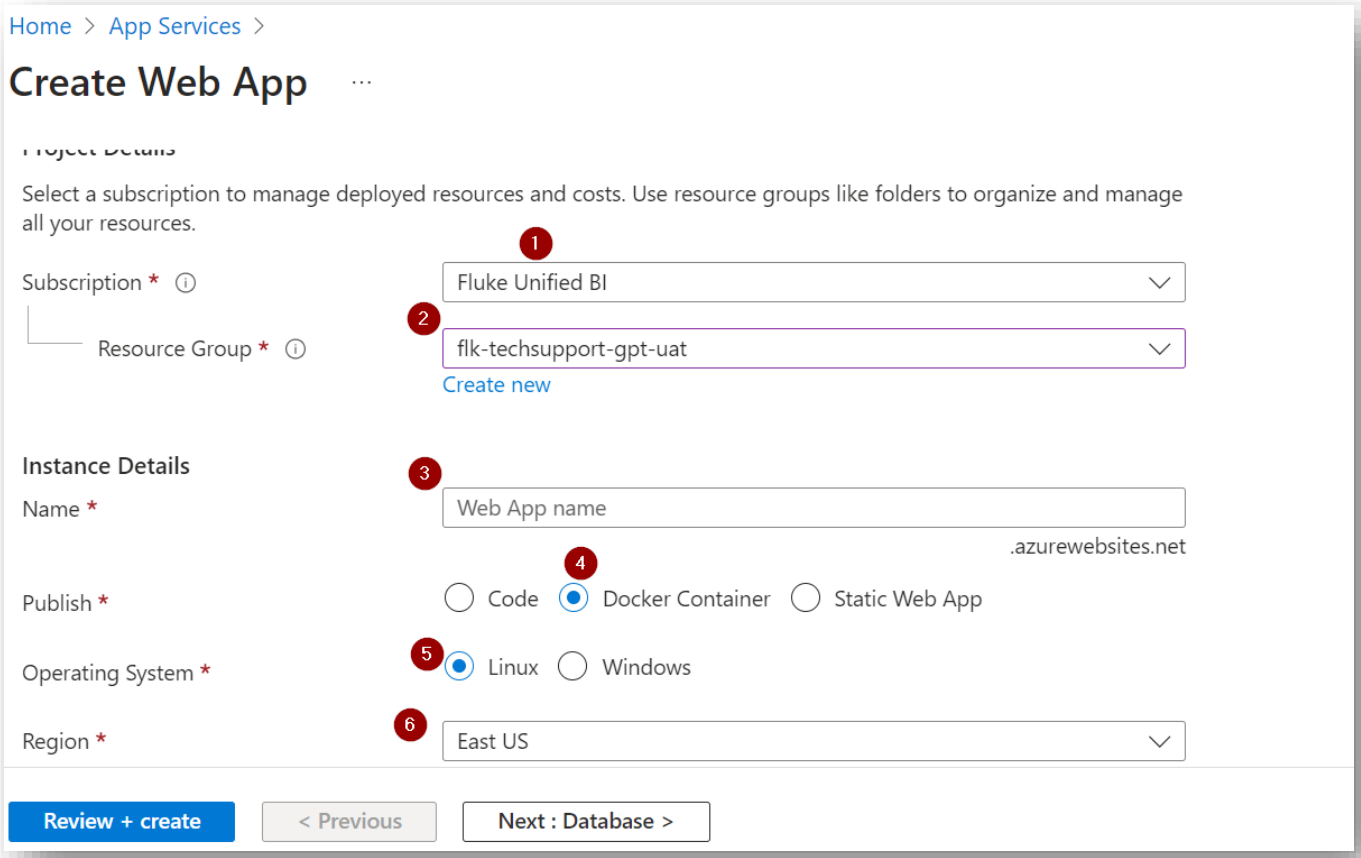


2. Click on Create-> Create Web App

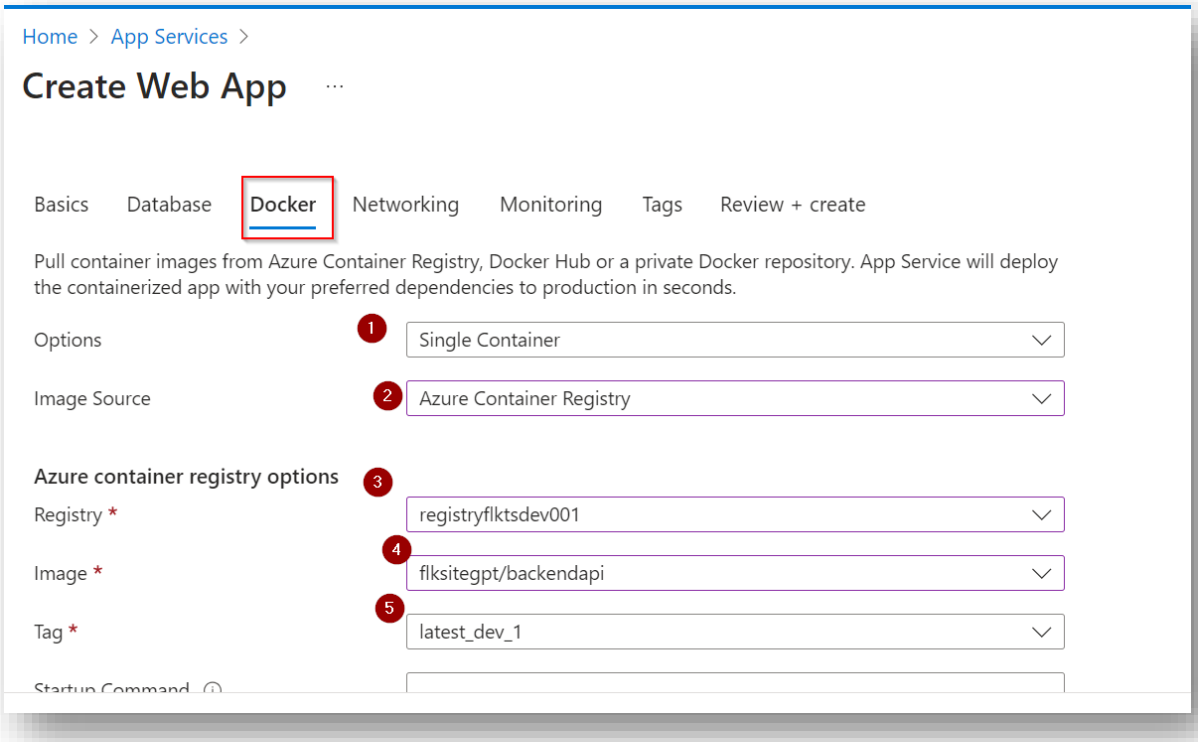


3. Enter the following details.
    a. Select the subscription for the resource.
    b. Select the resource group.
    c. Enter the Web app name.
    d. Select the Docker container. (As a container is published on dev in the Azure Container Registry).

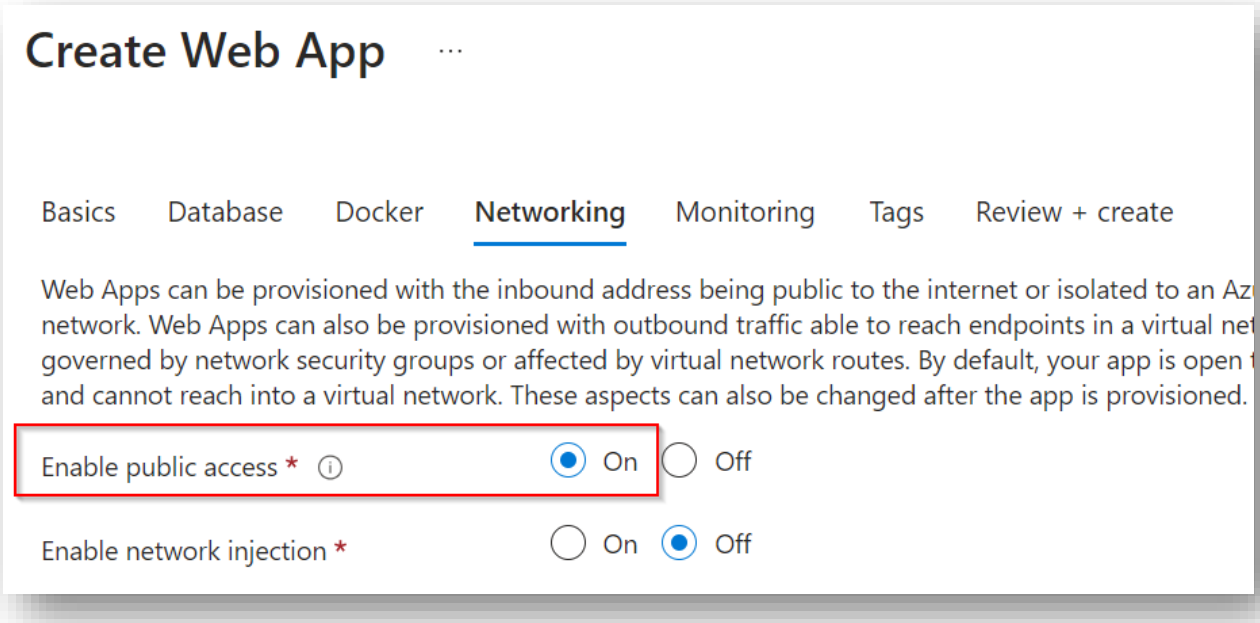
e. Select Linux as operating system.
f. Select same region as Resource Group's region.



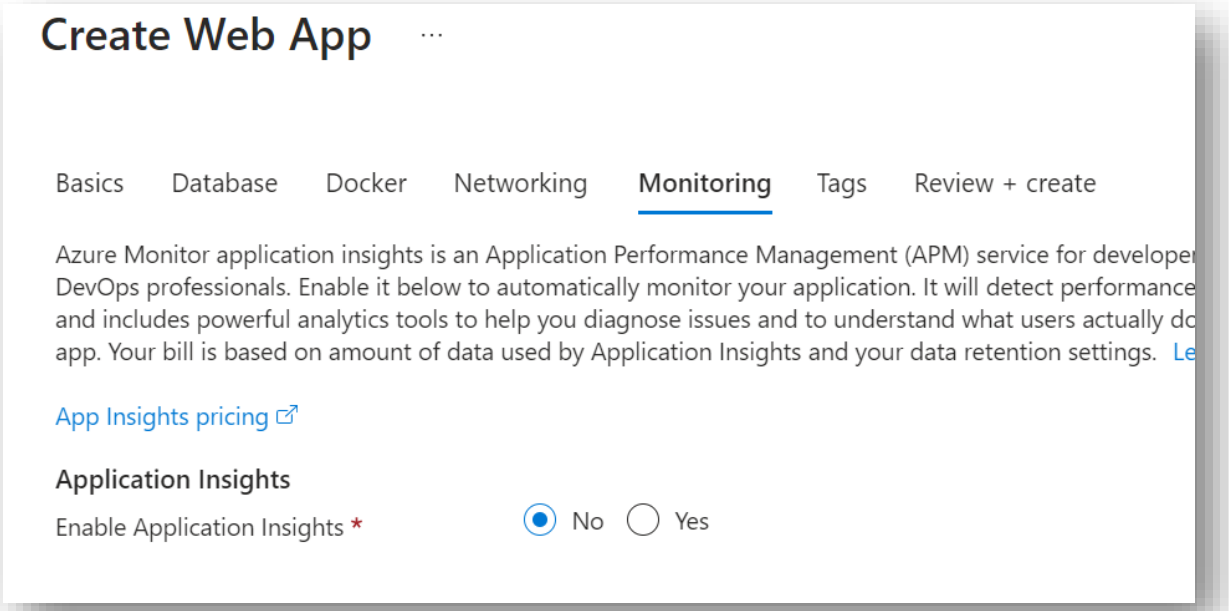
4. Under the Docker section enter the following details.
   a. Select Single Container.
   b. Select Azure Container Registry (Dev resource used ACR)
   c. Select the registry.
      i. As Docker image is a machine from where web app is deploed, we have used the same image of the docker as on dev.
   d. Select the Dev docker image.
   e. Click on next.



5. Under the Networking tab, Ensure that the public access is enabled.



6. Keep No on Enable Application Insights if you have a App Insight resource already created. Otherwise select yes, it will create a new resource on UAT.

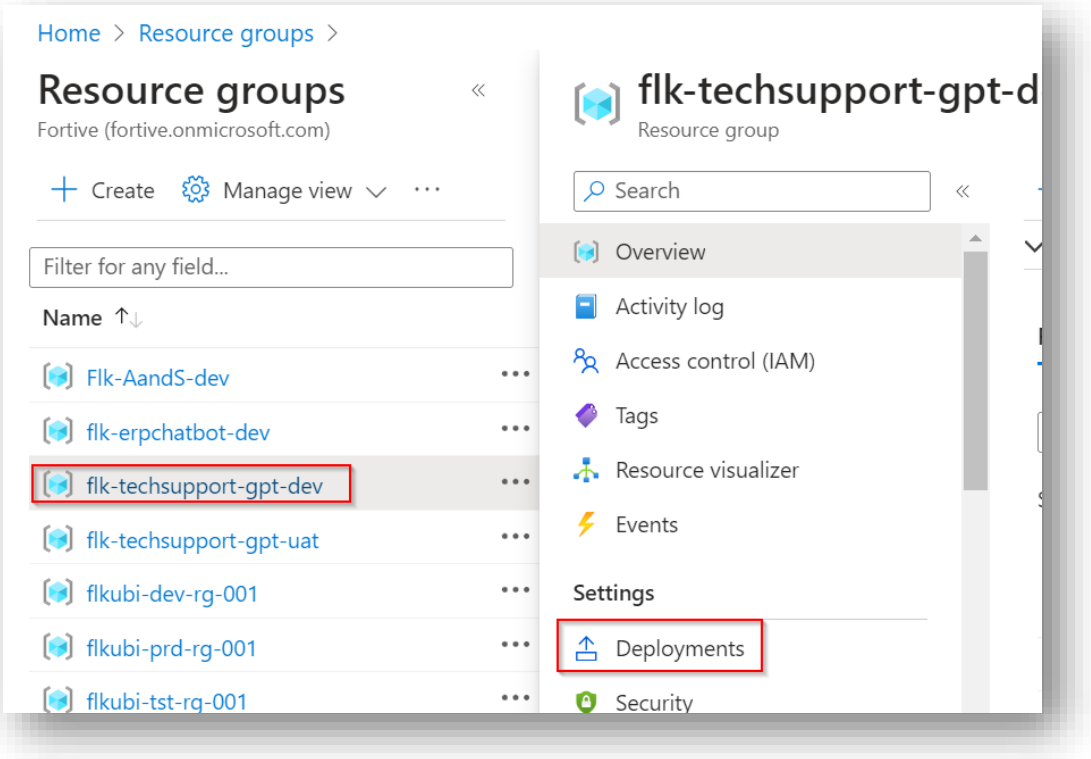


7. Click on review + create.
8. Verify the Web Application under the App Service tab opened in step 1.

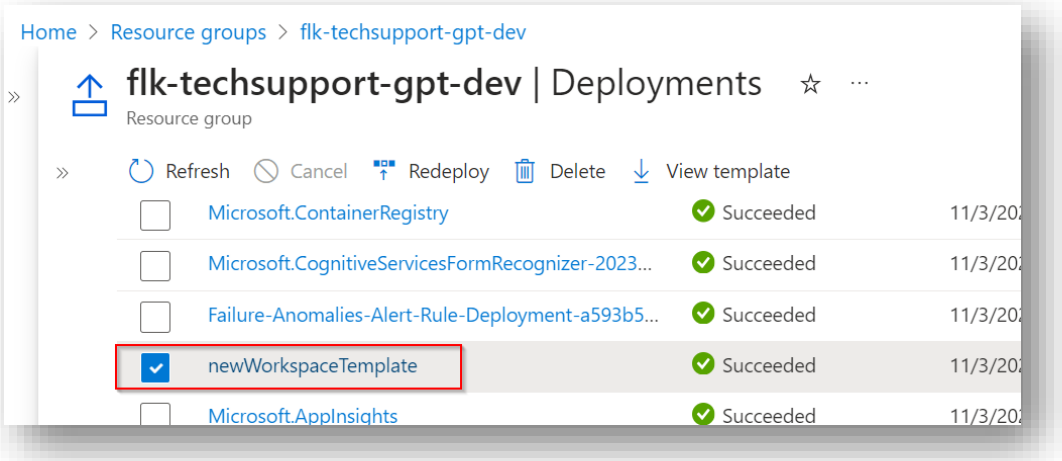
Log Workspace Creation

Export the Template + Deployment

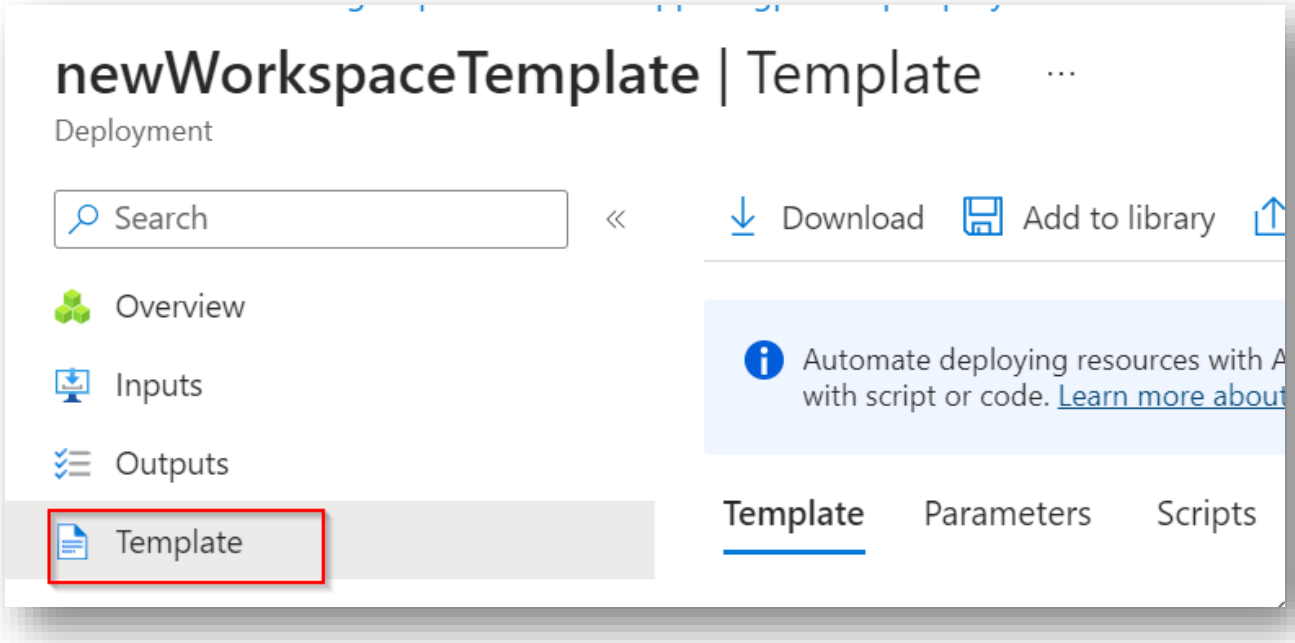
1. Open portal.azure.com and go the Dev resource group, select the Deployments.



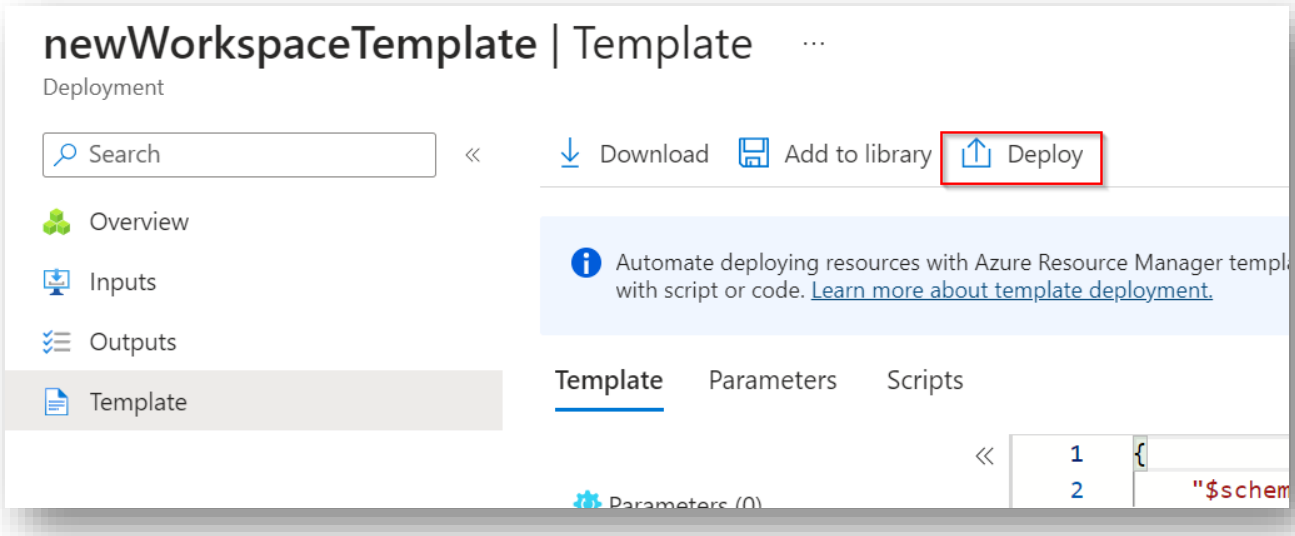
2. Select the deployment of the Log Workspace.



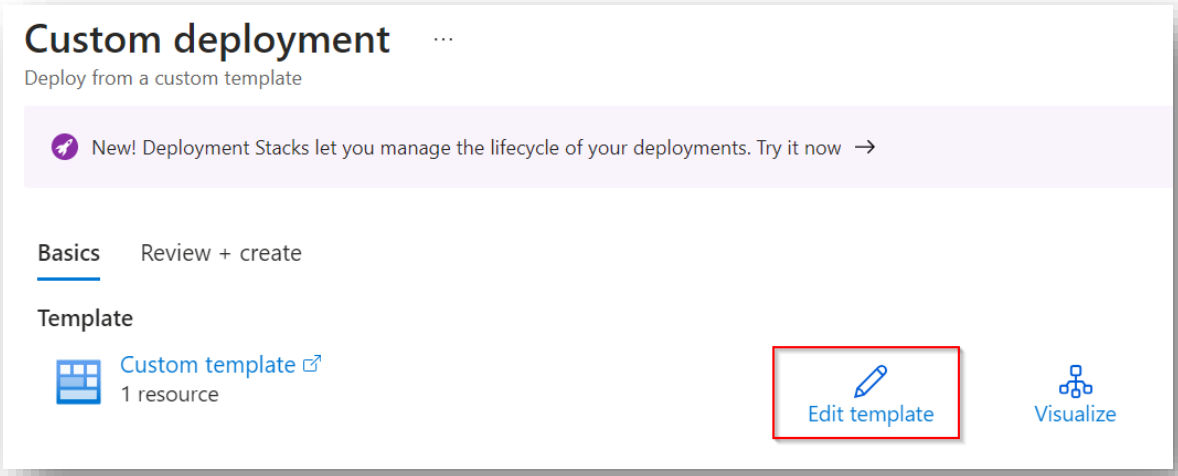
3. Click on Template to generate the deployed template.



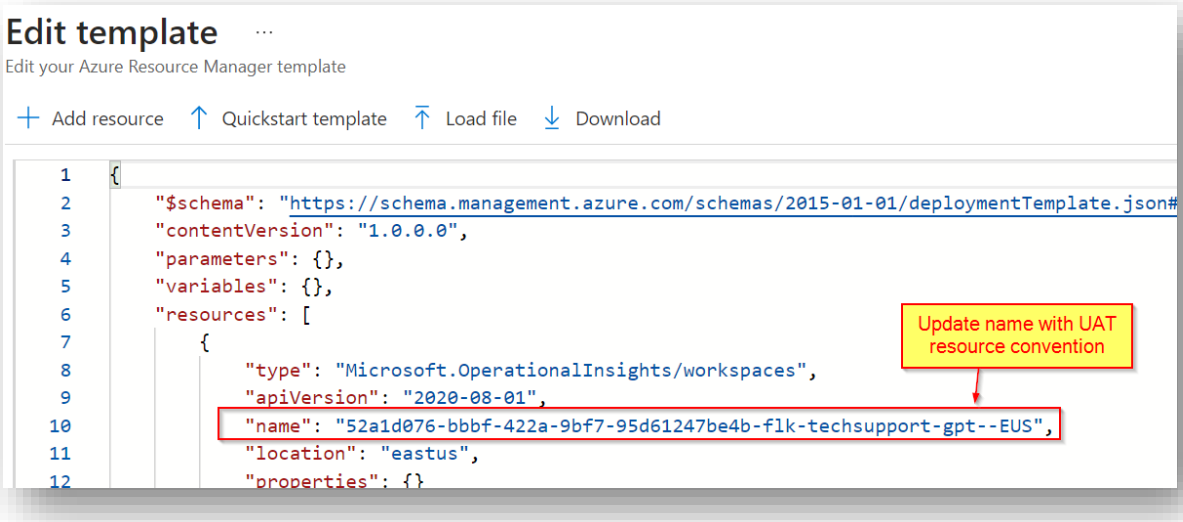
4. Click on the deploy above. It will open a Custom Deployment window.



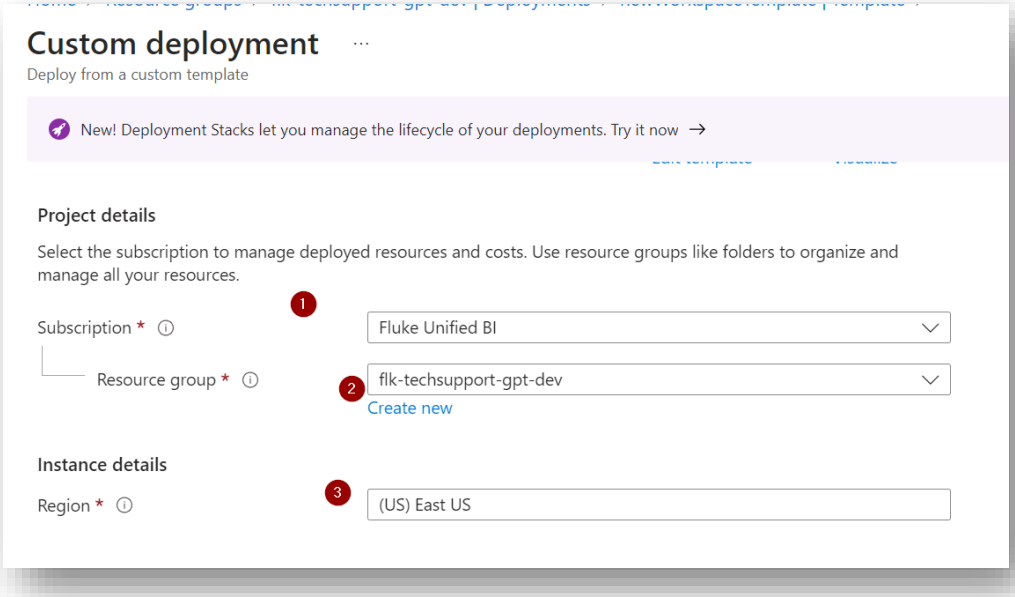
5. Click on Edit template.



6. In the opened editor, update the resource name with UAT environment conventions and click on save.



7. Update the following details in the opened window.
    a. Select the subscription.
    b. Select the resource group where the deployment is to be made.
    c. Select the region for the resource.

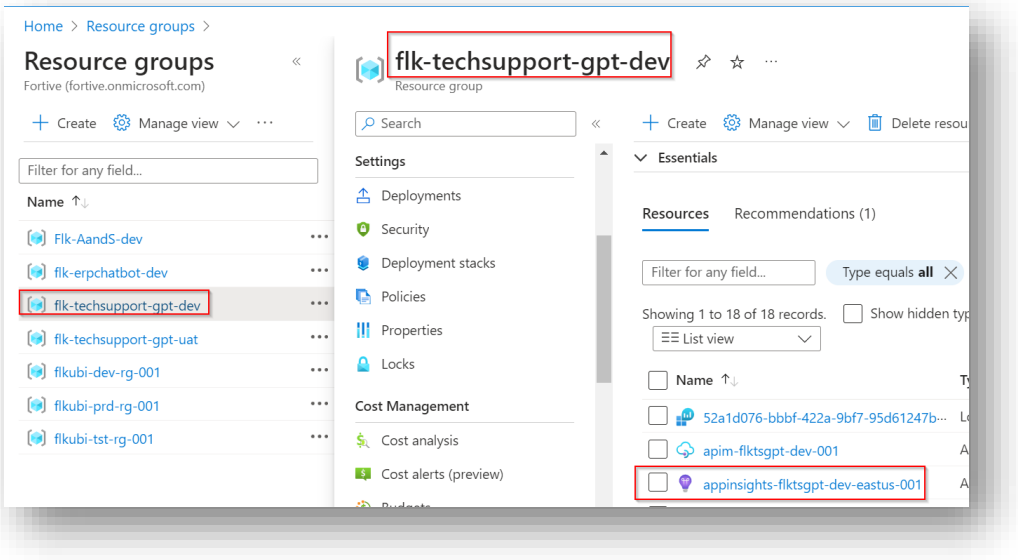


8. Click on next.
9. Click on Review + Create.

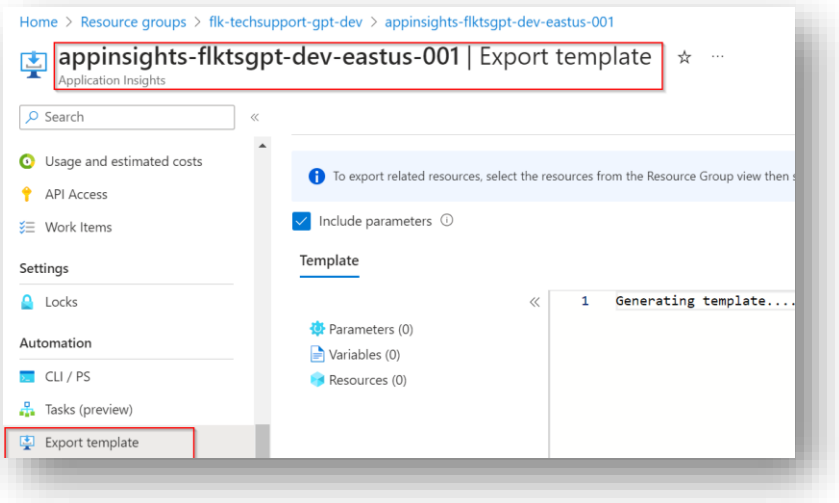
Application Insights

Export the Template

10. Open portal.azure.com and go the Dev resource group.
    a. Select the Application Insights resource.



11. In the Application Insights resource, on the left menu, click on the Export Template option

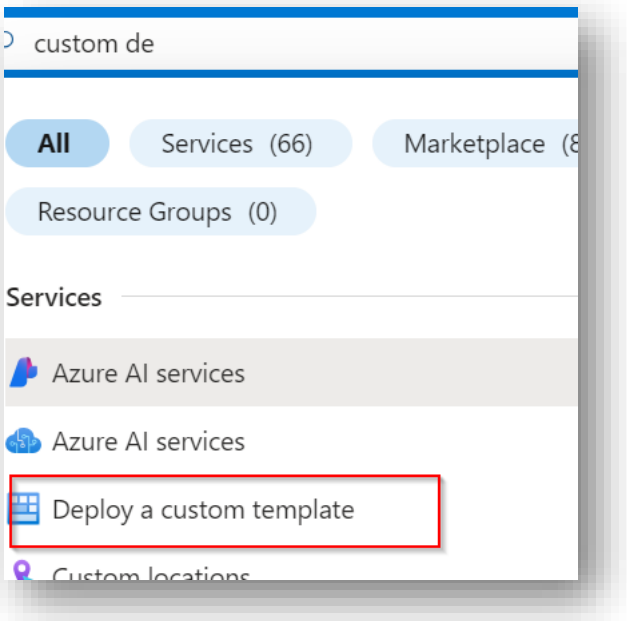


12. Copy the JSON content of the resource and save it in .JSON file.
13. Open the .JSON file in file editor.
    a. Replace the parameter name having ‘dev’ to ‘uat’ across the file

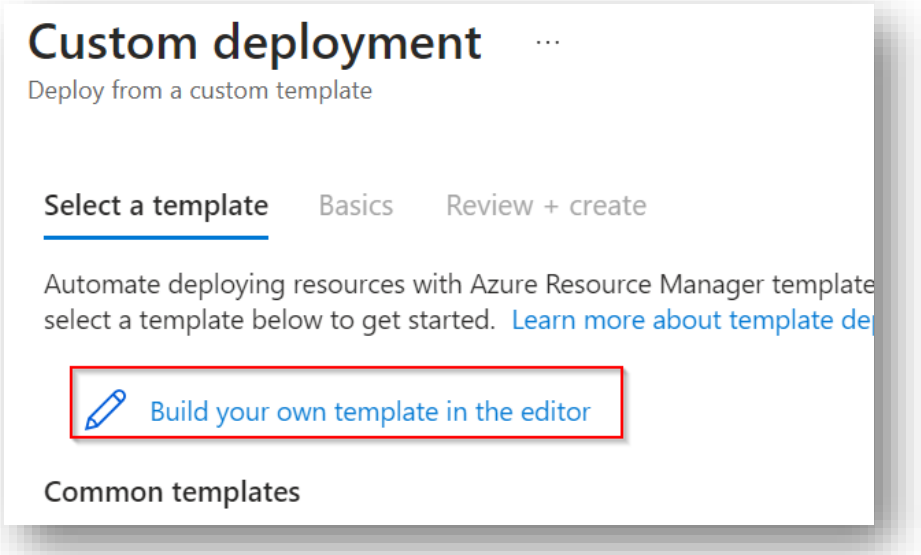
b. Replace all ‘dev’ occurrence to ‘uat’
c. Save the file.

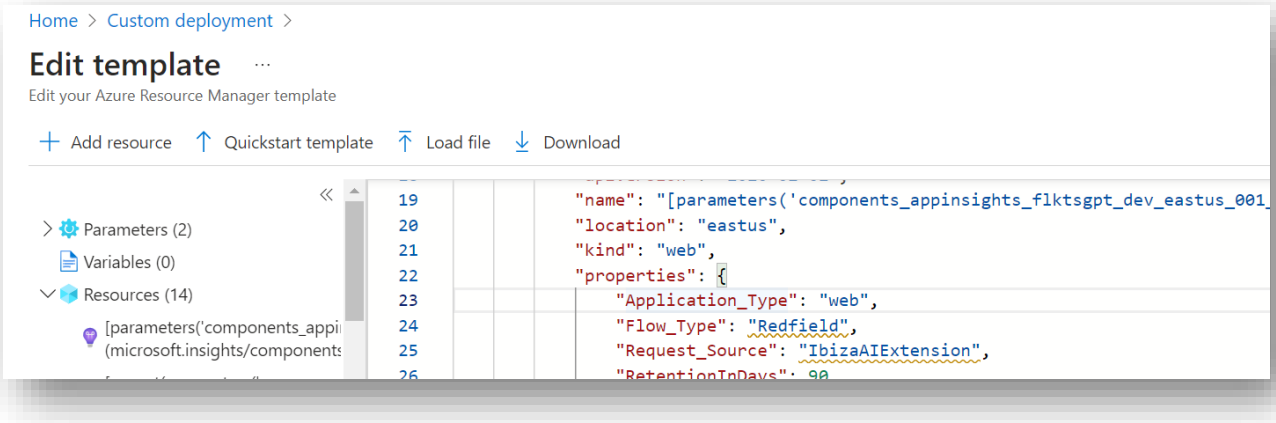
Deploy the Application Insights

1. Open the portal.azure.com and search the Custom Deployment.

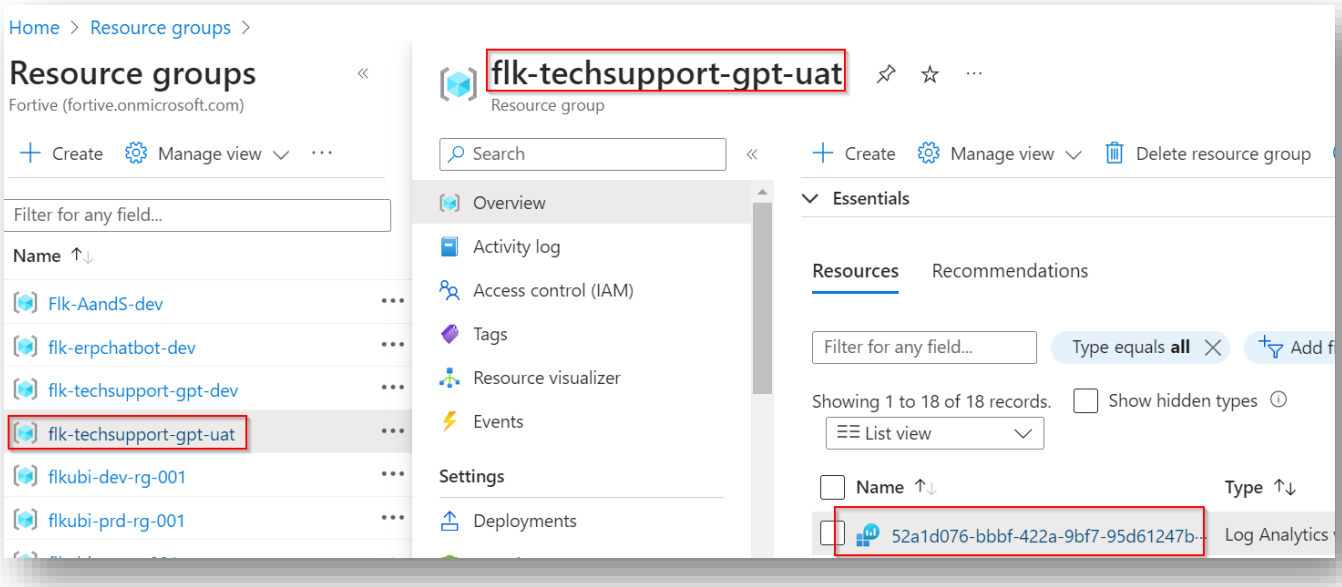


2. Now click on Edit and paste the saved template in the editor, and click on save.

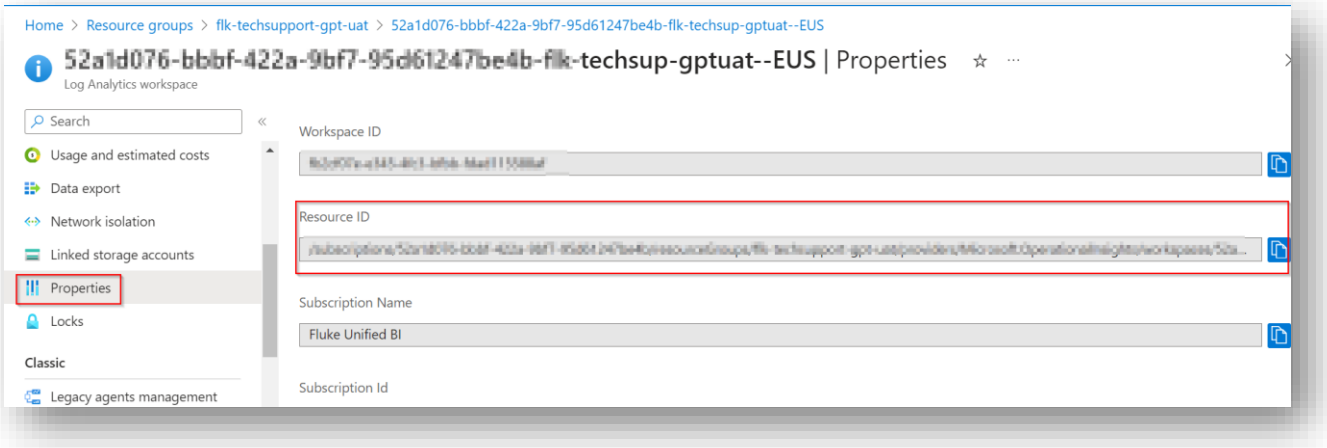




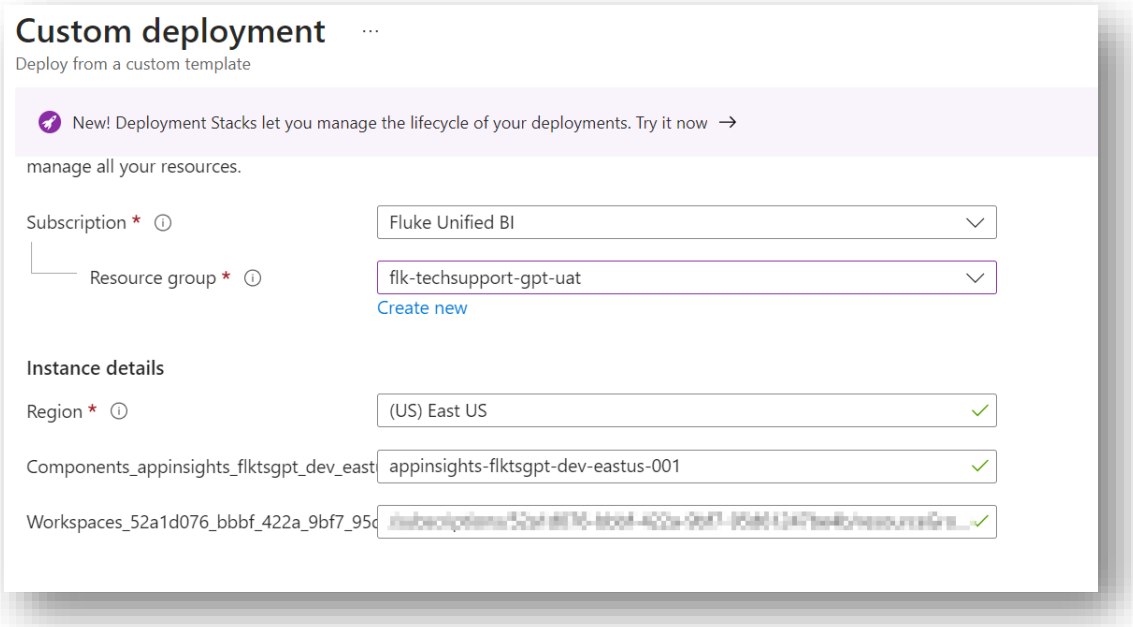
3. Now update the Parameter values in for the ARM template.
    a. Update the name of the resource.
    b. Copy the Resource ID of the Log Workspace created above.
        i. Open the Log Workspace resource in portal.azure.com.



        ii. On the left menu for properties, select the ResourceID.



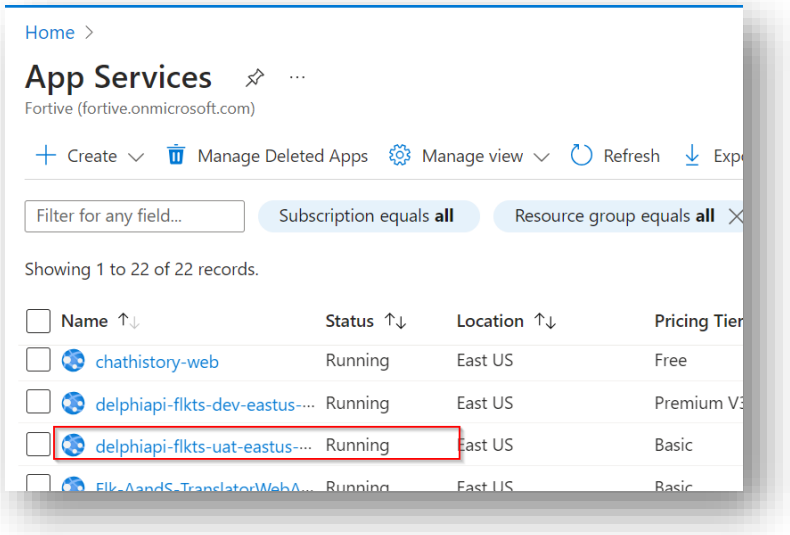
Once the values are added, click on Review + Create.



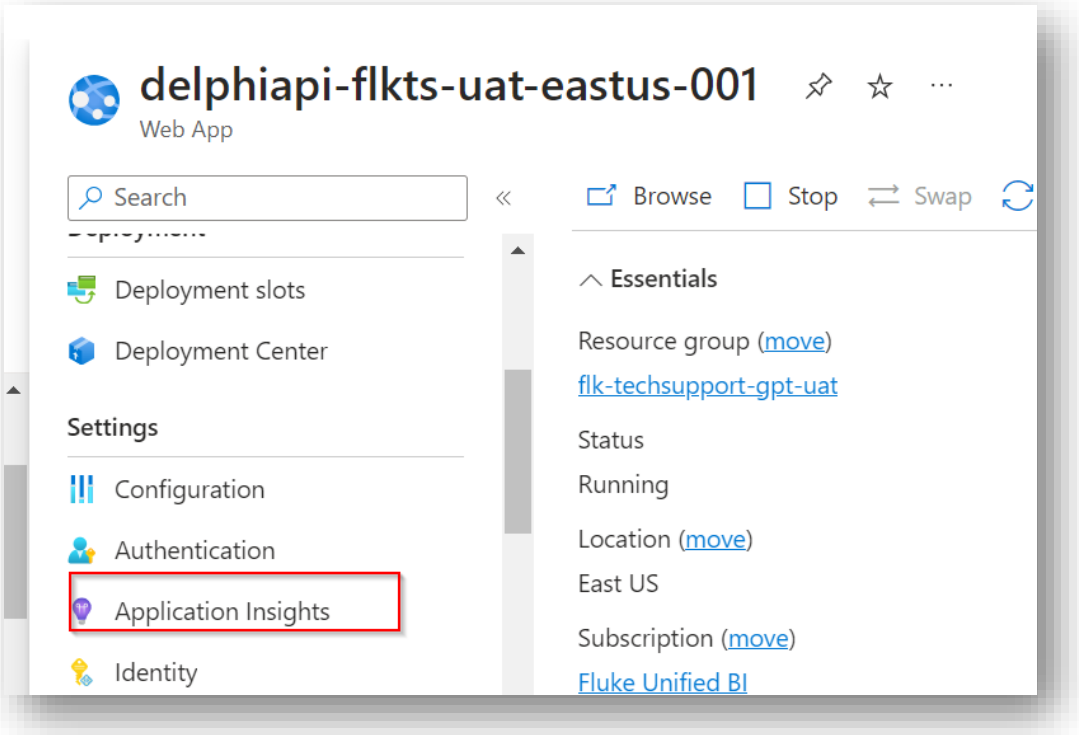
Create all Application Insights resources similar way.

Enable Web App/Other Resources with Application Insights

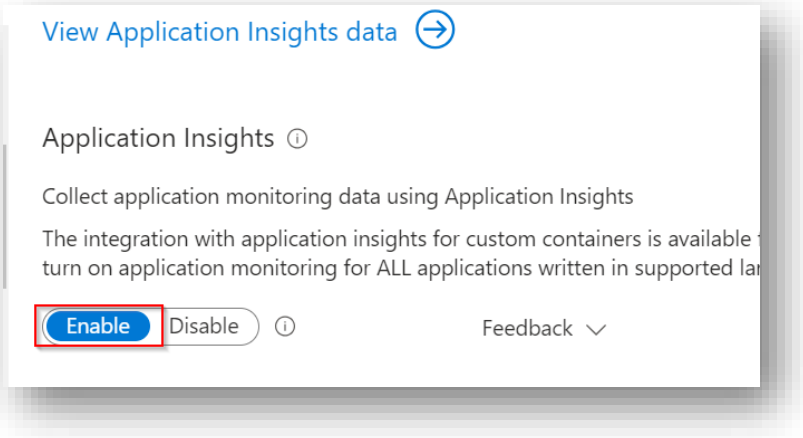
1. Go to portal.azure.com and open App Services.
    a. Select the required Web App resource.



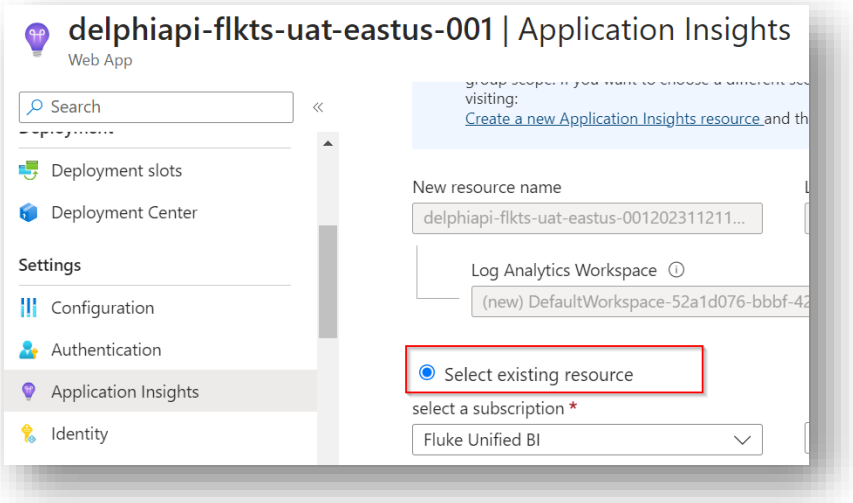
2. On the left menu, select the Application Insights.

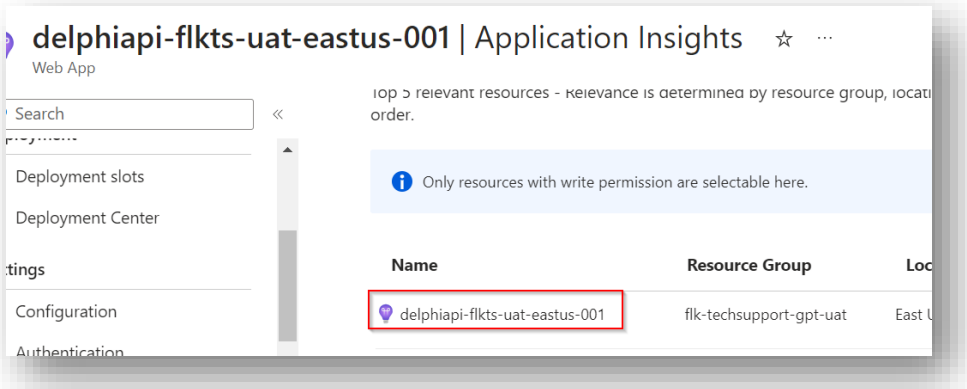


3. Click on Enable Application Insights resource.



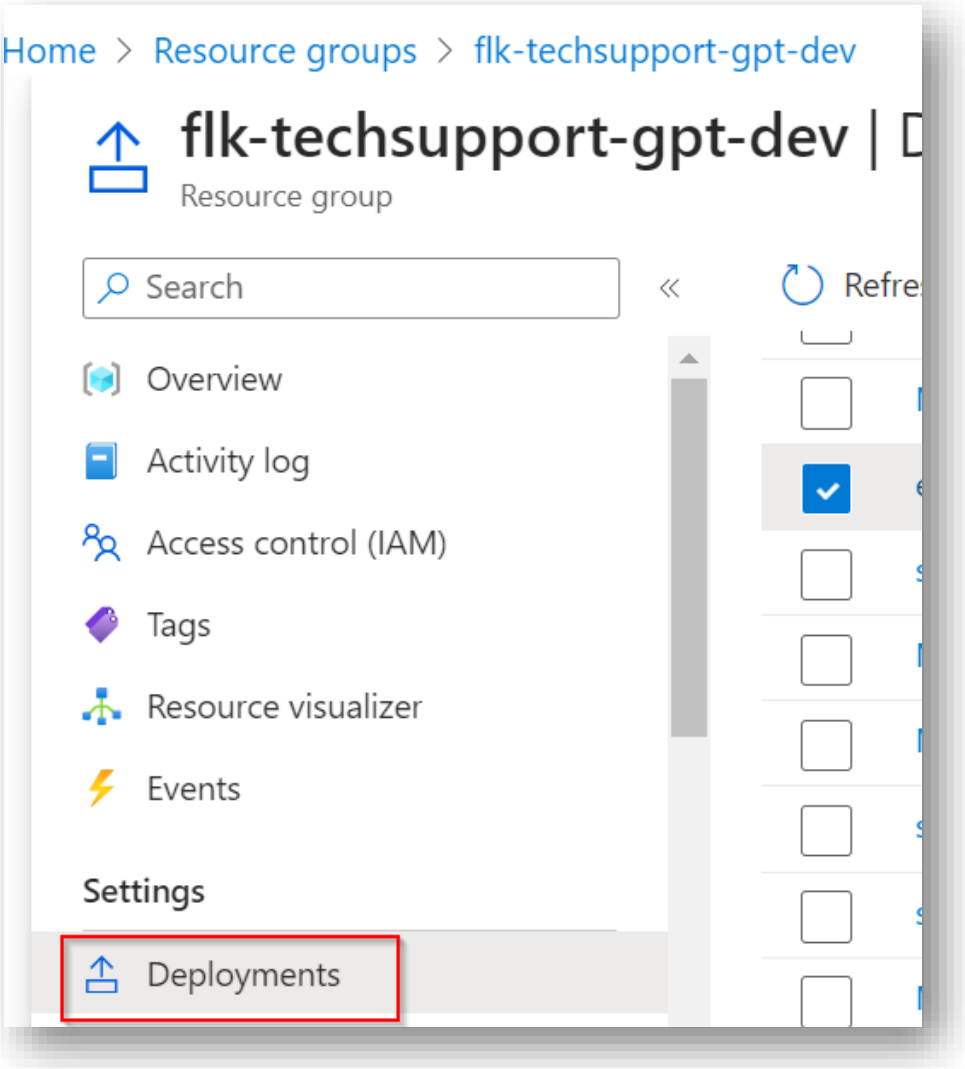
4. Click on the Select Existing Resource and select the relevant Application Insights resource.
    a. Click on Apply.



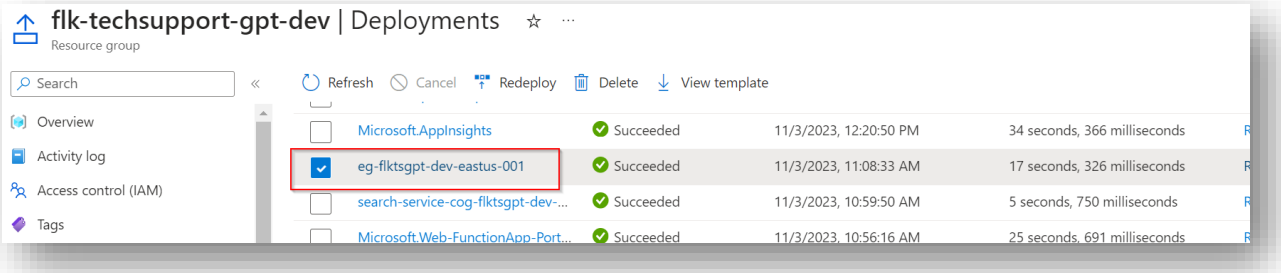


Event System Grid (Trigger for Function App)

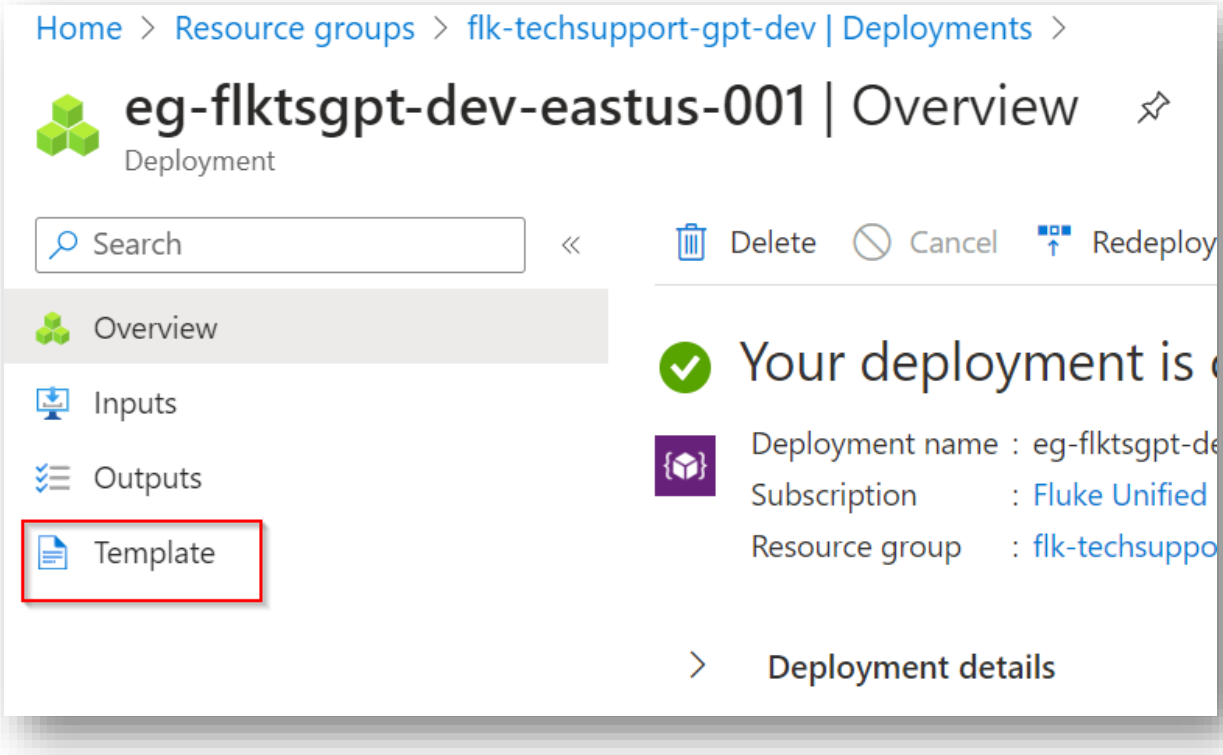
1. Open portal.azure.com, and open resource group.
    a. Click on the Deployments.



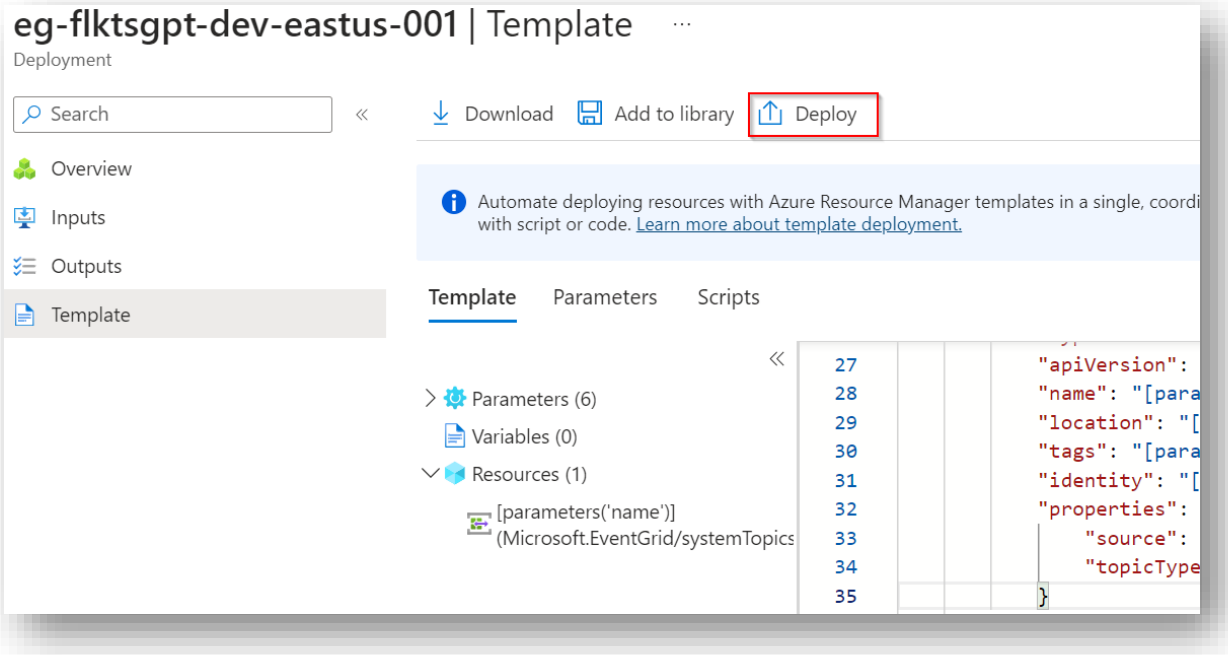
2. Locate the Event grid resource deployment and open that.



3. Click on Template to generate the template for the resource.

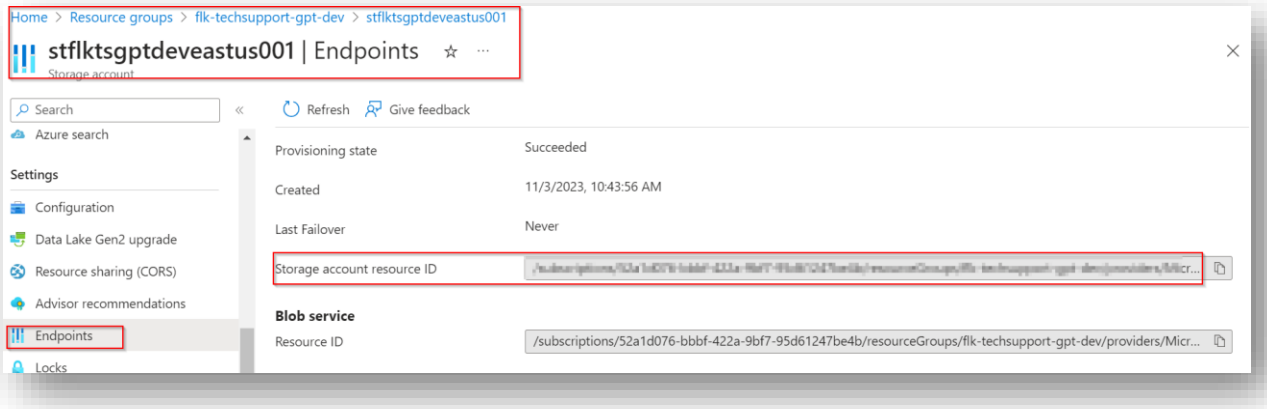


4. Now click on the deploy. It will open one Custom Deployment window.

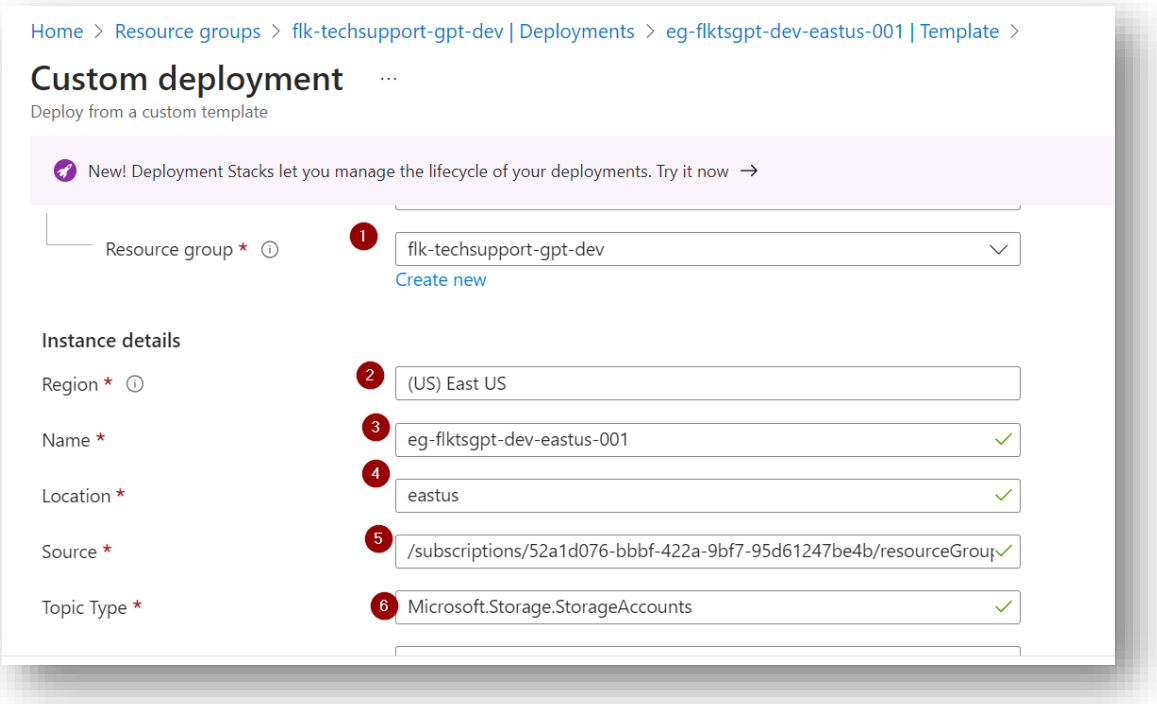


5. Update the following values.
    a. Select the resource group.

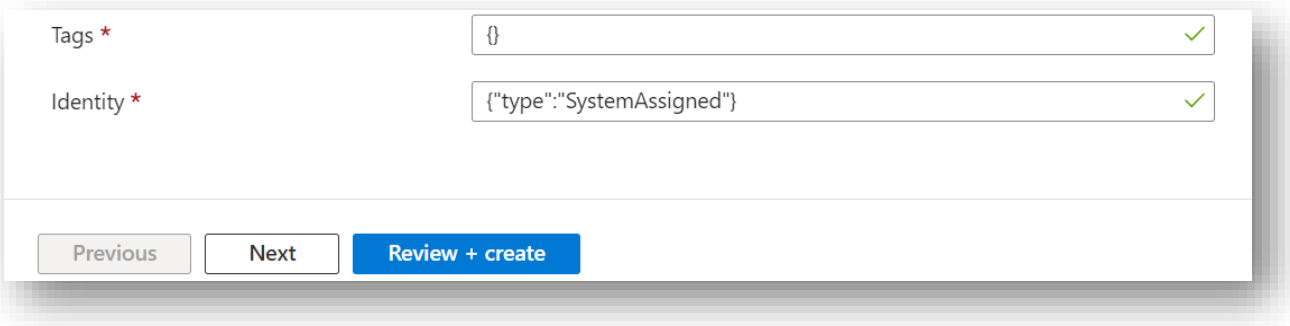
b. Select the region (Same as resource group region)
c. Provide proper name to the resource.
d. Add location of the resource (if it is not loaded bydefault after selecting above three things)
e. Add resource ID of the storage account.



i. To obtain the Resource ID, go the the storage account
ii. Click on the End Points on the left window.
iii. Copy the resource ID.

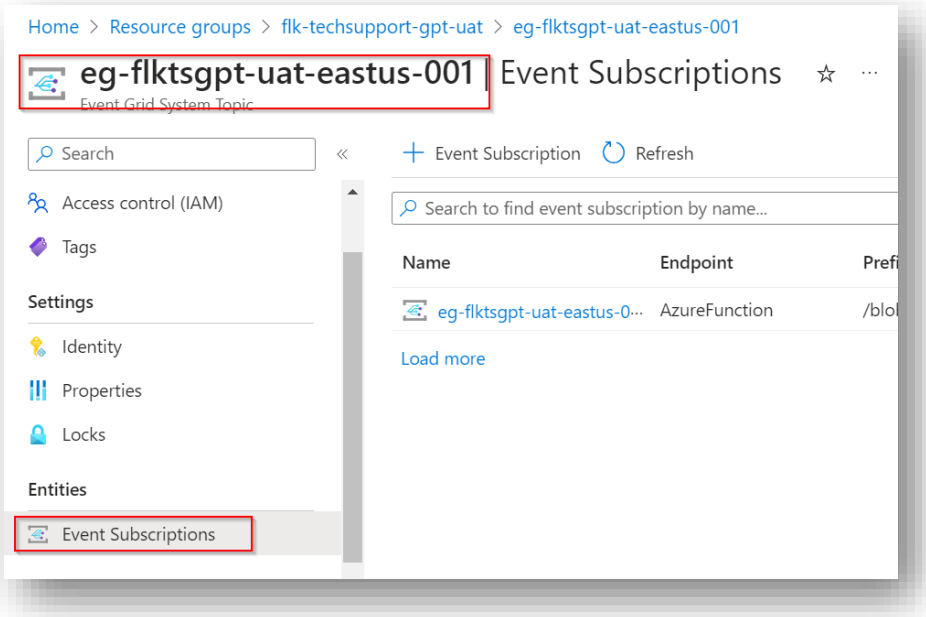


6. Keep the Identity as System Assigned & click on Review + Create.



Add Event Subscription

1. Go to Event Grid Subscription resources under portal.azure.com
    a. Select Event Subscription.

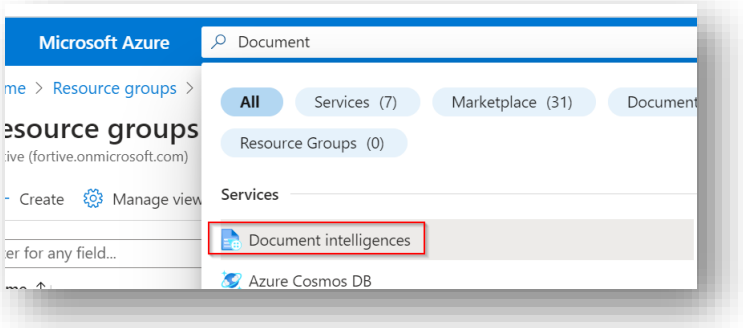


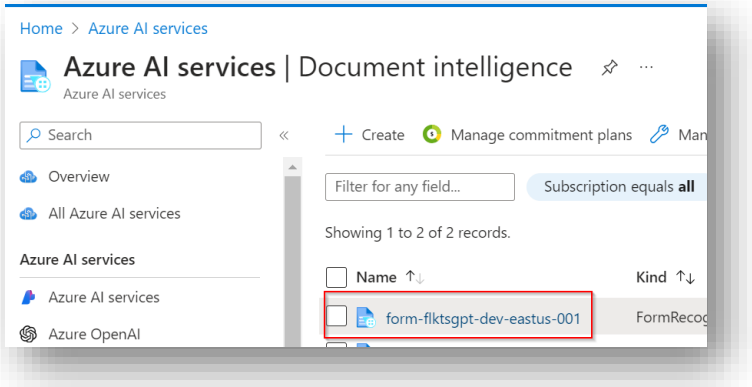
2. Click on + Event Subscription.
3. Copy the same properties from Dev resource.
4. Click on Create.

Document Intelligence: Form Recognizer

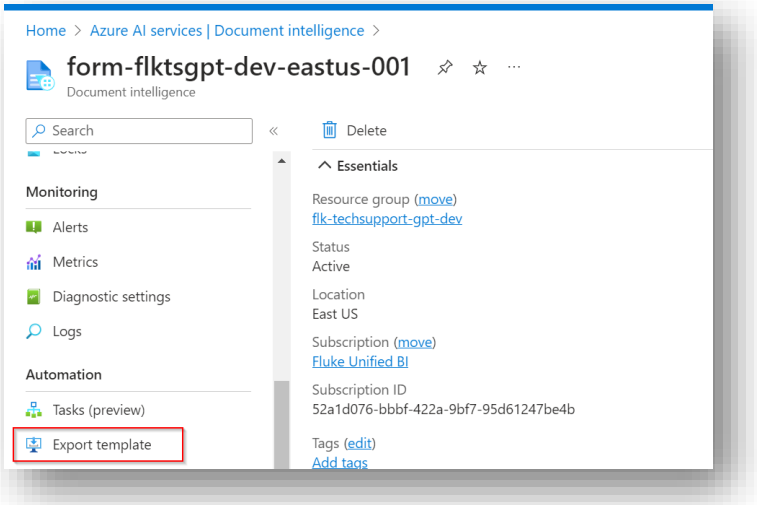
Export the ARM Template

1. Go to portal.azure.com and open the dev resource for Form Recognizer.

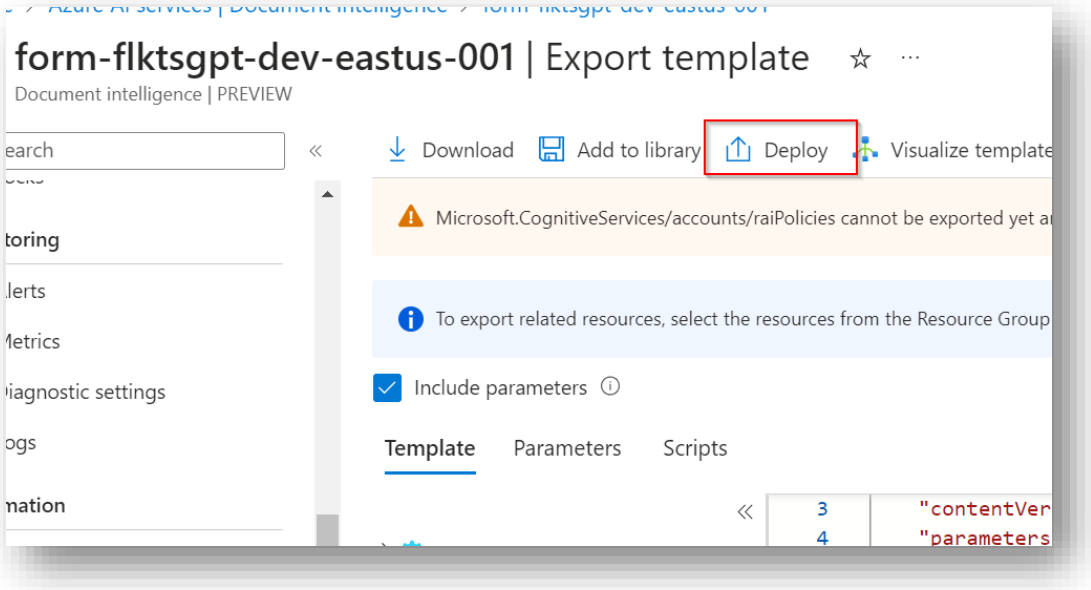




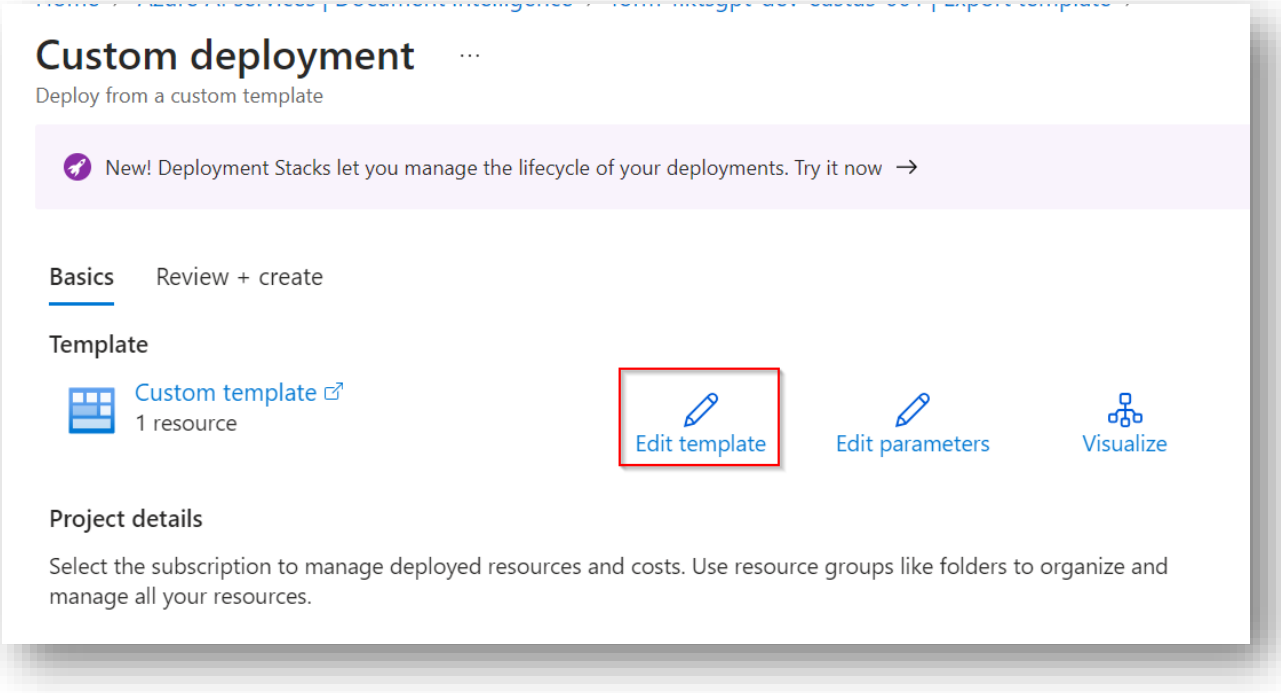
2. On the left menu, click on the Export Template.



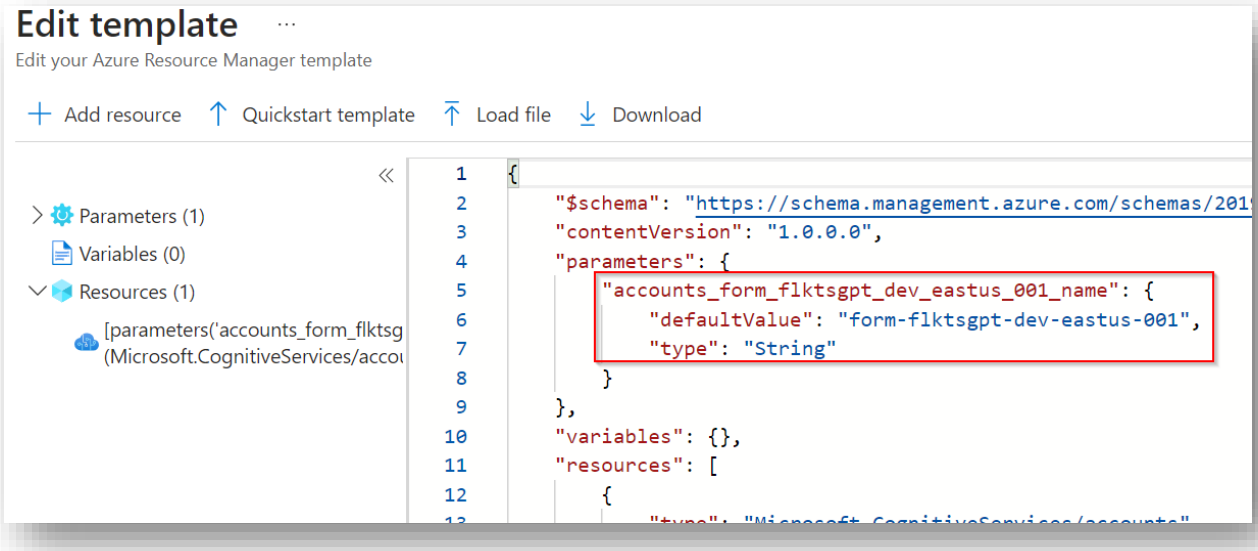
3. Click on the Deploy option. It will open one Custom Deployment window.



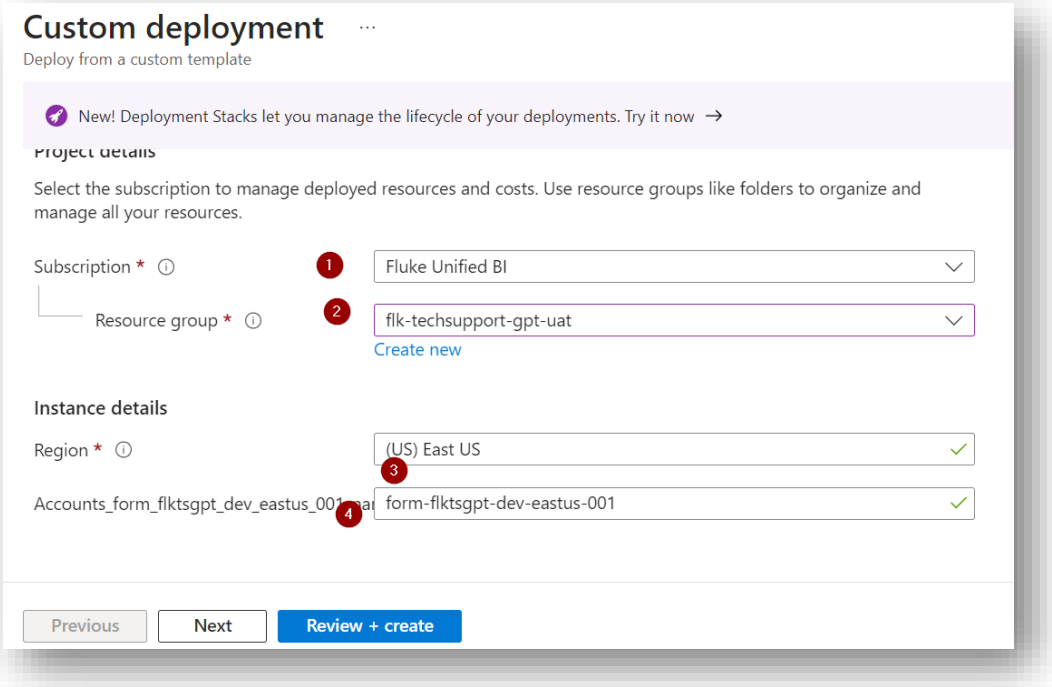
4. In the opened window, click on the Edit Template.



5. In the editor pane, now update the parameters and resource name.
    a. Replace keyword ‘dev’ to ‘uat’ everywhere in the JSON.
    b. Click on Save.



6. Update/verify the Parameters values in the deployment pane.
    a. Select proper subscription value.
    b. Select target resource group.
    c. Select the region of the resource. (Should be same as Resource Group)
    d. Verify the Name of the resource.



7. Click on Review + Create.

Appendix A - Structure of ARM Templates

ARM templates follows a JSON like structure, as given below.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "languageVersion": "",
  "contentVersion": "",
  "apiProfile": "",
  "definitions": { },
  "parameters": { },
  "variables": { },
  "functions": [ ],
  "resources": [ ], /* or "resources": { } with languageVersion 2.0 */
  "outputs": { }
}
```

Refer the following tables for the details of each component

Element Name	Required	Description
$schema	Yes	Location of the JavaScript Object Notation (JSON) schema file that describes the version of the template language. The version number you use depends on the scope of the deployment and your JSON editor.
languageVersion	No	Language version of the template. To view the enhancements of languageVersion 2.0.
contentVersion	Yes	Version of the template (such as 1.0.0.0). You can provide any value for this element. Use this value to document significant changes in your template. When deploying resources using the template, this value can be used to make sure that the right template is being used.
apiProfile	No	An API version that serves as a collection of API versions for resource types. Use this value to avoid having to specify API versions for each resource in the template. When you specify an API profile version and don't specify an API version for the resource type, Resource Manager uses the API version for that resource type that is defined in the profile
definitions	No	Schemas that are used to validate array and object values. Definitions are only supported in languageVersion 2.0.
parameters	No	Values that are provided when deployment is executed to customize resource deployment.

variables	No	Values that are used as JSON fragments in the template to simplify template language expressions.
functions	No	User-defined functions that are available within the template.
resources	Yes	Resource types that are deployed or updated in a resource group or subscription.
outputs	No	Values that are returned after deployment.

Definition

In the definitions section of the template, specify the schemas used for validating array and object values. Definitions can only be used with languageVersion 2.0.

```
"definitions": {
  "<definition-name": {
    "type": "<data-type-of-definition>",
    "allowedValues": [ "<array-of-allowed-values>" ],
    "minValue": <minimum-value-for-int>,
    "maxValue": <maximum-value-for-int>,
    "minLength": <minimum-length-for-string-or-array>,
    "maxLength": <maximum-length-for-string-or-array>,
    "prefixItems": <schema-for-validating-array>,
    "items": <schema-for-validating-array-or-boolean>,
    "properties": <schema-for-validating-object>,
    "additionalProperties": <schema-for-validating-object-or-boolean>,
    "discriminator": <schema-to-apply>,
    "nullable": <boolean>,
    "metadata": {
      "description": "<description-of-the-type-definition>"
    }
  }
}
```

Parameters

In the parameters section of the template, you specify which values you can input when deploying the resources. You're limited to 256 parameters in a template. You can reduce the number of parameters by using objects that contain multiple properties.

The available properties for a parameter are:

```
"parameters": {
  "<parameter-name>" : {
    "type" : "<type-of-parameter-value>",
    "defaultValue": "<default-value-of-parameter>",
    "allowedValues": [ "<array-of-allowed-values>" ],
    "minValue": <minimum-value-for-int>,
    "maxValue": <maximum-value-for-int>,
    "minLength": <minimum-length-for-string-or-array>,
    "maxLength": <maximum-length-for-string-or-array>,
    "prefixItems": <schema-for-validating-array>,
    "items": <schema-for-validating-array-or-boolean>,
    "properties": <schema-for-validating-object>,
    "additionalProperties": <schema-for-validating-object-or-boolean>,
    "discriminator": <schema-to-apply>,
    "nullable": <boolean>,
    "metadata": {
      "description": "<description-of-the parameter>"
    }
  }
}
```

Variables

In the variables section, you construct values that can be used throughout your template. You don't need to define variables, but they often simplify your template by reducing complex expressions. The format of each variable matches one of the data types. You're limited to 256 variables in a template.

The following example shows the available options for defining a variable:

```
"variables": {
  "<variable-name>": "<variable-value>",
  "<variable-name>": {
    <variable-complex-type-value>
  },
  "<variable-object-name>": {
    "copy": [
      {
        "name": "<name-of-array-property>",
        "count": <number-of-iterations>,
        "input": <object-or-value-to-repeat>
      }
    ]
  },
  "copy": [
    {
      "name": "<variable-array-name>",
      "count": <number-of-iterations>,
      "input": <object-or-value-to-repeat>
    }
  ]
}
```

Functions

Within your template, you can create your own functions. These functions are available for use in your template. Typically, you define complicated expressions that you don't want to repeat throughout your template. You create the user-defined functions from expressions and functions that are supported in templates.

When defining a user function, there are some restrictions:

- The function can't access variables.
- The function can only use parameters that are defined in the function. When you use the parameters function within a user-defined function, you're restricted to the parameters for that function.
- The function can't call other user-defined functions.
- The function can't use the reference function.
- Parameters for the function can't have default values.

```
"functions": [
  {
    "namespace": "<namespace-for-functions>",
    "members": {
      "<function-name>": {
        "parameters": [
          {
            "name": "<parameter-name>",
            "type": "<type-of-parameter-value>"
          }
        ],
        "output": {
          "type": "<type-of-output-value>",
          "value": "<function-return-value>"
        }
      }
    }
  }
]
```

Resource Group Deletion Consideration

1. Resources in the resource group specified during the deployment operation are subject to deletion if you deploy to multiple resource groups in a template.
2. Resources in secondary resource groups are not deleted when using complete mode.

Reference Links

Microsoft documentation: Link