Azure Storage
Intro to Azure Data Factory

Lecture 06

Deep Azure @ McKesson

Zoran B. Djordjević

Azure Storage Explorer

- Azure Storage Explorer (Preview) is a standalone app that enables you to easily work with Azure Storage data on Windows, macOS, and Linux. In this article, you learn the various ways of connecting to and managing your Azure storage accounts.
- You can download and install Azure Storage Explorer from this URL:

https://azure.microsoft.com/en-us/features/storage-explorer/

Storage Explorer provides several ways to connect to storage accounts.

Connect to storage accounts associated with your Azure subscriptions.

Connect to storage accounts and services that are shared from other Azure subscriptions.

Connect to and manage local storage by using the Azure Storage Emulator.

In addition, you can work with storage accounts in global and national Azure:

Connect to an Azure subscription: Manage storage resources that belong to your Azure subscription.

Work with local development storage: Manage local storage by using the Azure Storage Emulator.

Attach to external storage: Manage storage resources that belong to another Azure subscription or that are under national Azure clouds by using the storage account's name, key, and endpoints.

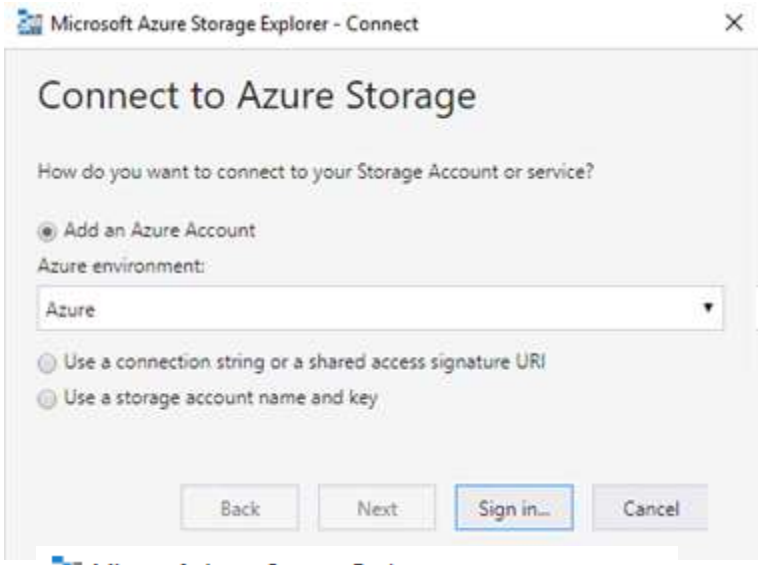
Attach a storage account by using an SAS: Manage storage resources that belong to another Azure subscription by using a shared access signature (SAS).

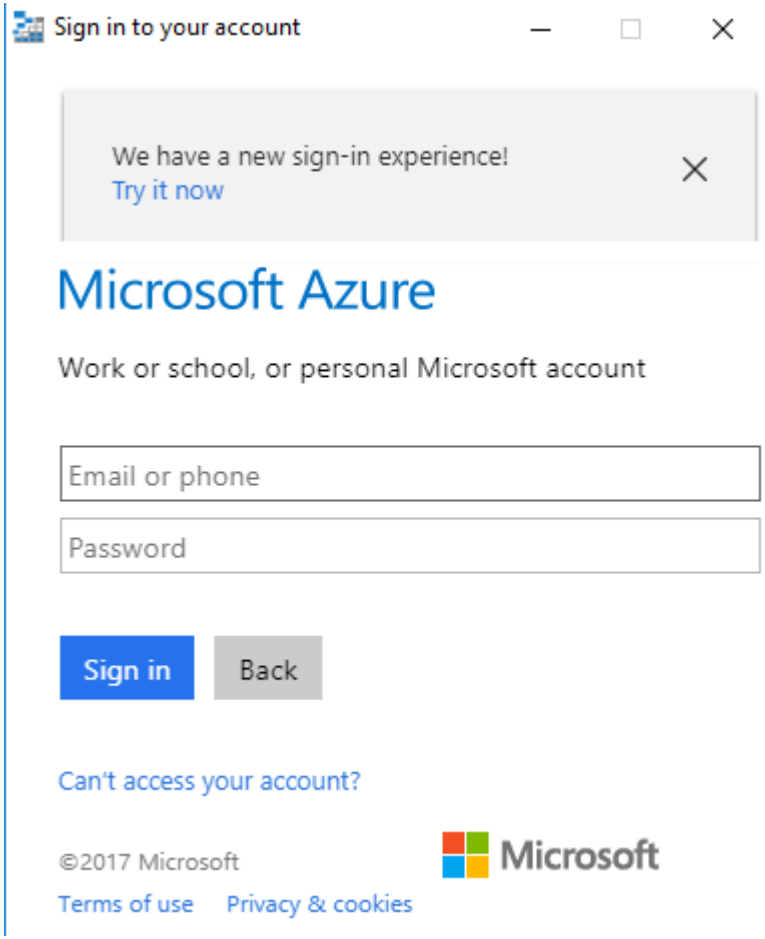
Attach a service by using an SAS: Manage a specific storage service (blob container, queue, or table) that belongs to another Azure subscription by using an SAS.

Connect to an Azure Cosmos DB account by using a connection string: Manage Cosmos DB account by using a connection string.

Sign In

- When you start ACE for the first time you will be asked to Sign In:





Work with Local Development Storage

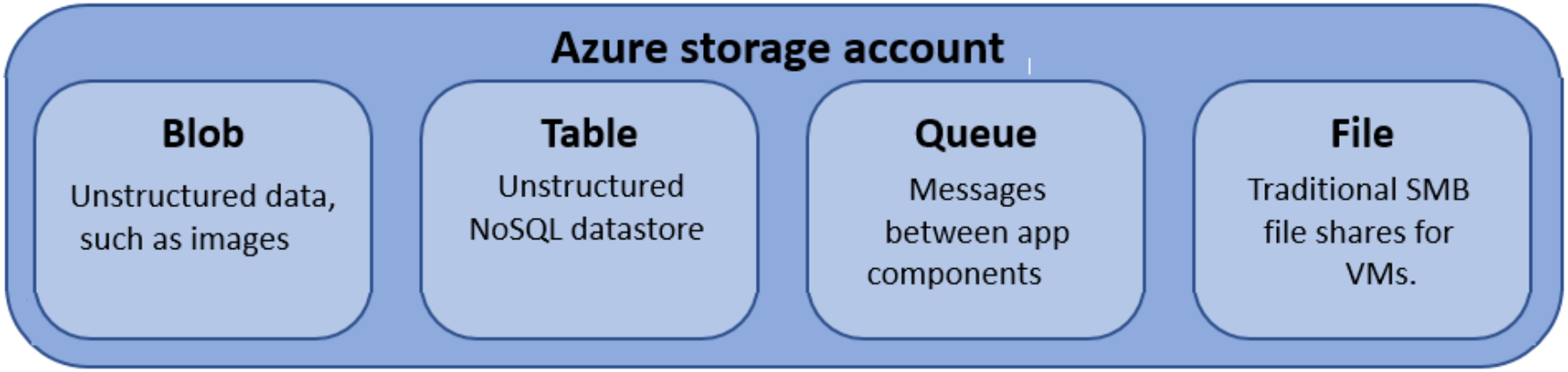
- With Storage Explorer, you can work against local storage by using the Azure Storage Emulator. This approach lets you write code against and test storage without necessarily having a storage account deployed on Azure, because the storage account is being emulated by the Azure Storage Emulator.

Azure Storage Client Tools

Azure Storage Client Tool	Block Blob	Page Blob	Append Blob	Tables	Queues	Files	Free
Microsoft Azure Portal	X	X	X	X	X	X	Y
Microsoft Azure Storage Explorer	X	X	X	X	X	X	Y
Microsoft Visual Studio Server Explorer	X	X	X	X	X		Y
Cerabrata: Azure Management Studio	X	X	X	X	X	X	Trial
Redgate: Azure Explorer	X	X	X				Y
Azure Web Storage Explorer	X	X		X	X		Y
CloudBerry Explorer	X	X				X	Y/N
Cloud Combine	X	X		X	X		Trial
ClumsyLeaf: AzureXplorer, CloudXplorer, TableXplorer	X	X	X	X	X	X	Y

Azure Storage

- We look at the different types of storage in Azure and when to use them.
- We also look at redundancy and replication options for Azure storage, and how to get the best performance for your applications.
- Azure offers at least 4 (four) types of storage media: Blobs, Files, Table and Queues



- To access those storage media, you need a storage account.
- You can have as many storage accounts as you need
- Sometime Azure Disks are added to this classification as the 5th type of storage.

Scenarios for Different Storage Types

Feature	Description	When to use
Azure Files	Provides an SMB interface, client libraries, and a REST interfacethat allows access from anywhere to stored files.	You want to "lift and shift" an application to the cloud which already uses the native file system APIs to share data between it and other applications running in Azure. You want to store development and debugging tools that need to be accessed from many virtual machines.
Azure Blobs	Provides client libraries and a REST interface that allows unstructured data to be stored and accessed at a massive scale in block blobs.	You want your application to support streaming and random access scenarios. You want to be able to access application data from anywhere.
Azure Disks	Provides client libraries and a REST interface that allows data to be persistently stored and accessed from an attached virtual hard disk.	You want to lift and shift applications that use native file system APIs to read and write data to persistent disks. You want to store data that is not required to be accessed from outside the virtual machine to which the disk is attached.

Storage & Access Pricing

- https://azure.microsoft.com/en-us/pricing/details/storage/blobs/
- These are the costs of storing your data in block blobs. The prices shown below are the monthly charges per GB of data stored. These prices vary based on the access tier of Block Blob storage (Hot, Cool or Archive) and the redundancy option that you choose, as well as the amount of data you store.

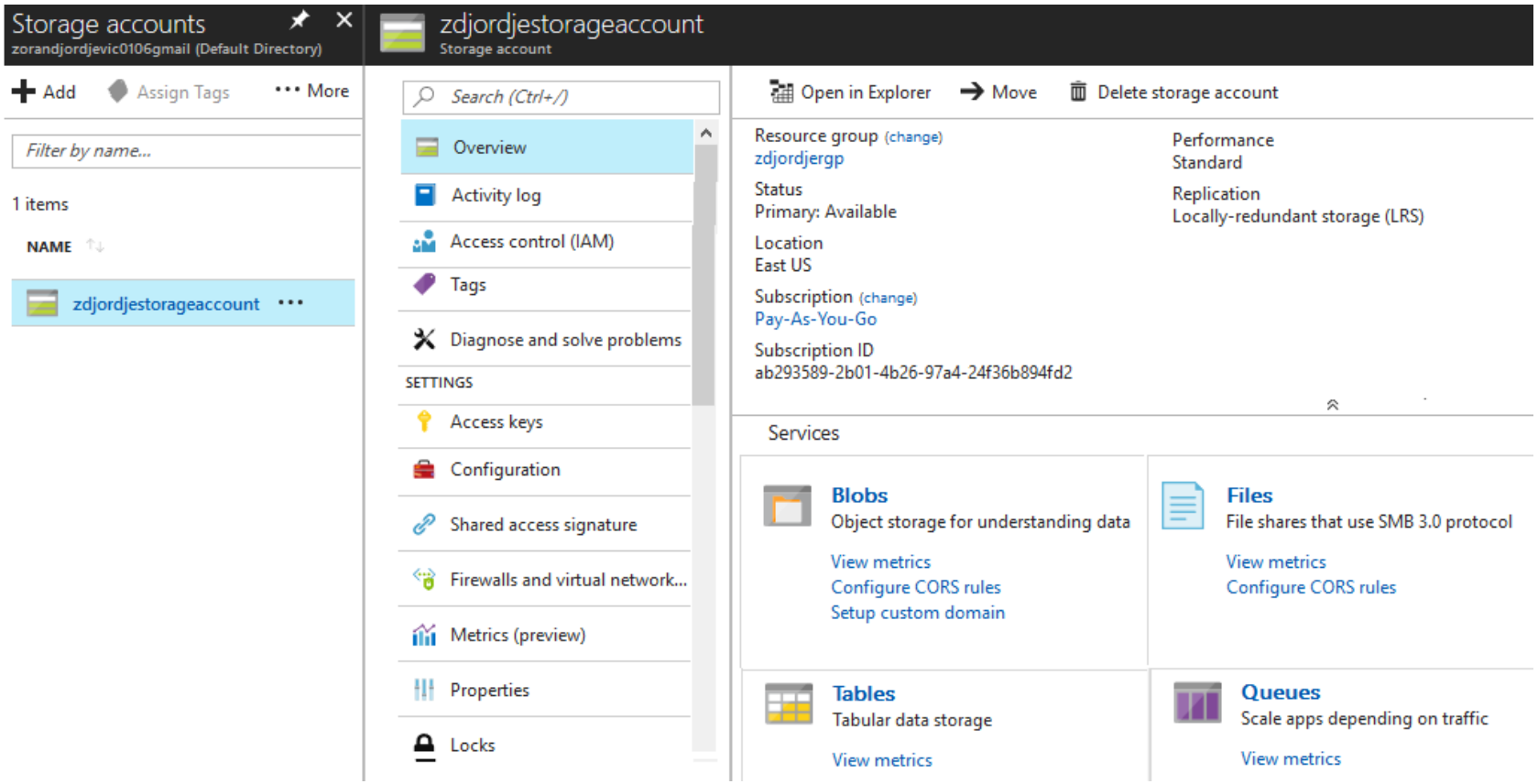
	HOT	COOL
First 50 TB / Month	$0.0184	$0.01
Next 450 TB / Month	$0.0177	$0.01

- Bellow are costs of HTTP operations against your block blob data, as well as the cost of retrieving data from or writing data into your block blobs.

	HOT	COOL
Write Operations* (per 10,000)	$0.05	$0.10
List and Create Container Operations (per 10,000)	$0.05	$0.10
Read Operations** (per 10,000)	$0.004	$0.01
All other Operations, except Delete which is free	$0.004	$0.01
Data Retrieval (per GB)	Free	$0.01

Inside a Storage Account

- Create a storage account, in an existing resource group, and you will see:



Types of Azure Storage

- **Blob storage** - for unstructured data such as media files and documents. Applications can store files in blob storage, such as images, audio, video, etc., and then render them on the Web or other clients.
- **Table storage** - for semi-structured data . Azure Table, like Google Big Table, could meet great performance requirements when processing large amounts of data. You typically store meta data in Table storage.
- **Queue storage** - for cloud applications to communicate between the various tiers and components in a reliable and consistent manner. You can create, read, and delete messages that pass between application components. We could use queue storage to pass messages between the web front-end when a customer makes an order and the store/application back-ends.
- **File storage –** is a Server Message Block (SMB) file share, accessible by both Windows and Linux/macOS platforms. Often used to centralize log collection from VMs.

Azure Python SDK and azurerm

- On your Ubuntu VM

```
sudo apt install python-pip
sudo pip install --upgrade pip
```

You can install each Azure service's library individually:

```
$ pip install azure-batch          # Install the latest Batch runtime library
$ pip install azure-mgmt-storage   # Install the latest Storage management library
```

Preview packages can be installed using the --pre flag:

```
$ pip install --pre azure-mgmt-compute # will install only the latest Compute Management library
```

- You can also install a set of Azure libraries in a single line using the azure meta-package.

```
$ pip install azure
```

Finally to check whether you have `azurerm` package, type:

```
$ pip install azurerm
```

Azure Storage Account

- Microsoft Azure Storage is a Microsoft-managed cloud service that provides storage that is highly available, secure, durable, scalable, and redundant. Microsoft takes care of maintenance and handles critical problems for you.
- Azure Storage consists of three data services: Blob storage, File storage, and Queue storage.
- Blob storage supports both standard and premium storage, with premium storage using only SSDs for the fastest performance possible. Another feature is cool storage, allowing you to store large amounts of rarely accessed data for a lower cost.
- https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction

- Storage objects must have unique names all written in lower case.
- There are 2 type of Blobs:
  - Block blob: Use case Documents, backups
    - Allows random read-writes, you can change document and store it back.
  - Page blobs: Used for VM data disks up to 1 TB

Storage Account Endpoints

- Every object that you store in Azure Storage has a unique URL address. The storage account name forms the subdomain of that address. The combination of subdomain and domain name, which is specific to each service, forms an *endpoint* for your storage account.
- For example, if your storage account is named *mystorageaccount*, then the default endpoints for your storage account are:
  - Blob service: http://*mystorageaccount*.blob.core.windows.net
  - Table service: http://*mystorageaccount*.table.core.windows.net
  - Queue service: http://*mystorageaccount*.queue.core.windows.net
  - File service: http://*mystorageaccount*.file.core.windows.net
- You can also configure a custom domain name to use with your storage account.

Create Storage Account

- On Win, Linux or Azure Cloud Shell prompt, create a resource group

```
az group create --name zdjResourceGroup  --location eastus
```

- If you're unsure which region to specify for the --location parameter, you can retrieve a list of supported regions for your subscription with the az account list-locations command .

```
az account list-locations --query "[].{Region:name}" --out table
```

- There are several types of storage accounts appropriate for different usage scenarios, each of which supports one or more of the storage services (blobs, files, tables, or queues). The following table details the available storage account types.

Type of storage account	General-purpose Standard	General-purpose Premium	Blob storage, hot and cool access tiers
Services supported	Blob, File, Table, Queue services	Blob service	Blob service
Types of blobs supported	Block blobs, page blobs, append blobs	Page blobs	Block blobs and append blobs

Create General Purpose Storage Account

- Create a general-purpose standard storage account with the `az storage account create` command.

```
az storage account create --name zdjstorageaccount02 --resource-group zdjResourceGroup
--location eastus --sku Standard_LRS --encryption blob
{/ Finished ..
  "accessTier": null,
  "creationTime": "2017-11-14T18:04:04.389448+00:00",
  "customDomain": null,
  "enableHttpsTrafficOnly": false,
  "encryption": {
    "keySource": "Microsoft.Storage",
    "keyVaultProperties": null,
    "services": {
      "blob": {
        "enabled": true,"lastEnabledTime": "2017-11-14T18:04:04.393445+00:00"
      },
      "file": {
        "enabled": true,
        "lastEnabledTime": "2017-11-14T18:04:04.393445+00:00"
      },
      "queue": null,   "table": null
    }
  },
  "id": "/subscriptions/ab293589-2b01-4b26-97a4-24f36b894fd2/resourceGroups/zdjresourcegroup/providers/
Microsoft.Storage/storageAccounts/zdjstorageaccount",
  "identity": null,
  "kind": "Storage",
  "lastGeoFailoverTime": null,
  "location": "eastus",
  "name": "zdjstorageaccount",
  "networkRuleSet": {
    "bypass": "AzureServices",
    "defaultAction": "Allow",
    "ipRules": [],
    "virtualNetworkRules": []
  },
```

Create General Purpose Storage Account


```
"primaryEndpoints": {
    "blob": "https://zdjstorageaccount.blob.core.windows.net/",
    "file": "https://zdjstorageaccount.file.core.windows.net/",
    "queue": "https://zdjstorageaccount.queue.core.windows.net/",
    "table": "https://zdjstorageaccount.table.core.windows.net/"
  },
  "primaryLocation": "eastus",
  "provisioningState": "Succeeded",
  "resourceGroup": "zdjresourcegroup",
  "secondaryEndpoints": null,
  "secondaryLocation": null,
  "sku": {
    "capabilities": null,
    "kind": null,
    "locations": null,
    "name": "Standard_LRS",
    "resourceType": null,
    "restrictions": null,
    "tier": "Standard"
  },
  "statusOfPrimary": "available",
  "statusOfSecondary": null,
  "tags": {},
  "type": "Microsoft.Storage/storageAccounts"
}
```

Specify storage account credentials

- The Azure CLI needs your storage account credentials for most of the commands in this tutorial. While there are several options for doing so, one of the easiest ways to provide them is to set the AZURE_STORAGE_ACCOUNT and AZURE_STORAGE_ACCESS_KEY environment variables.

```
$ az storage account keys list   --account-name zdjstorageaccount02    --
resource-group zdjResourceGroup   --output table
KeyName      Value
---------   ----------------------------------------------------------------------------------
key1        ezNEovJ1opRZtzEa1J7CNoB1jKFAXiXqPp20QzaI2LyLxTkiJAmsxX/9rFVFwpWAUmQaMEaSctly/QhTl9gTTA==
key2        9CmS40v3cySpVrua+eSyMObwPwu91FhQZc9rfsdSPx8ymPolzcE4eYIBASC2ZxlFQ2i/NhNNl8/SkjoCz7T2uA==
```

- Create or set environmental variables. Two keys are primary and secondary access keys.

```
AZURE_STORAGE_ACCESS_KEY=ezNEovJ1opRZtzEa1J7CNoB1jKFAXiXqPp20QzaI2LyLxTkiJAmsxX
/9rFVFwpWAUmQaMEaSctly/QhTl9gTTA==
AZURE_STORAGE_ACCOUNT=zdjstorageaccount
```

- You can view and copy storage access keys in Azure Portal
- In the Azure portal, navigate to your storage account, click **All settings** and then click **Access keys** to view, copy, and regenerate your account access keys. The **Access Keys** blade also includes pre-configured connection strings using your primary and secondary keys that you can copy to use in your applications.

Create a Container

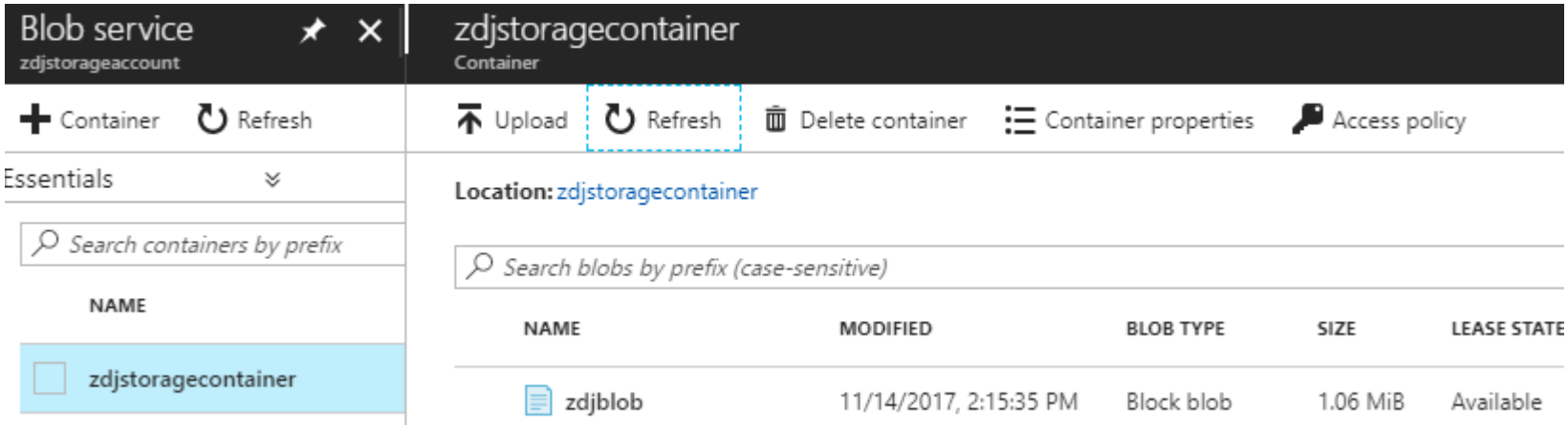
- Blobs are always uploaded into a container. Containers allow you to organize groups of blobs like you organize files in directories on your computer.
- Create a container for storing blobs with the az storage container create command

```
$ az storage container create --name zdjstoragecontainer02
{
  "created": true
}
```

Upload a Blob

- Blob storage supports block blobs, append blobs, and page blobs.
- Most files stored in Blob storage are stored as block blobs.
- Append blobs are used when data must be added to an existing blob without modifying its existing contents, such as for logging.
- Page blobs back the VHD files of IaaS virtual machines.
- In this example, we upload a blob to the container we created in the last step with the az storage blob upload command.

```
az storage blob upload --container-name zdjstoragecontainer02 --name blob03 --file
E:\CLASSES\codeda06\Welcome_DeepAzure.pdf
Finished[#############################################################]  100.0000%
{
  "etag": "\"0x8D52B9415508800\"",
  "lastModified": "2017-11-14T19:15:35+00:00"
}
```



List Blobs

- Previous operation created the blob if it doesn't already exist, and overwrites it if it does. You can upload as many files as you like.

- We list the blobs in the container with the `az storage blob list` command.

```
$ az storage blob list  --container-name zdjstoragecontainer02 --
output table

az storage blob list  --container-name zdjstoragecontainer --
output table
Name      Blob Type    Blob Tier      Length  Content Type
--------  -----------  -----------  --------  ----------------------------
----------------------------------------
zdjblob   BlockBlob                    20675
application/vnd.openxmlformats-officedocument.wordprocessingml.document
2017-11-14T19:25:01+00:00

zdjblob2  BlockBlob                  1109743  application/pdf
2017-11-14T19:29:00+00:00
```

Download a Blob

- We could use the `az storage blob download` command to download a blob you uploaded earlier. The syntax of the command

```
az storage blob download \
    --container-name mystoragecontainer02 \
    --name blobName \
    --file ~/destination/path/for/file
```

```
$ mkdir Downloads
$ chmod 777 –R Downloads # this to make sure you can write to this directory
$ az storage blob download   --container-name
zdjstoragecontainer02  --name blob03  --file
E:\CLASSES\codeda06\NewFile.pdf
```

- `SomeFile.pdf` is the local name of a file into which you copy the blob

Table Storage

- Azure Table is an imitation of Google's Big Table. It is much more modest in capabilities, though. Azure Table was Azure's first entry in NoSQL world.
- Azure now offers a more versatile NoSQL tool called Azure Cosmos DB. Azure Cosmos DB in public preview that offers throughput-optimized tables, global distribution, and automatic secondary indexes.
- Very soon CosmosDB will be fully functional product.
- Azure Table storage is a service that stores structured NoSQL data in the cloud, providing a key/attribute store with a schemaless design. Because Table storage is schemaless, it's easy to adapt your data as the needs of your application evolve. Access to Table storage data is fast and cost-effective for many types of applications, and is typically lower in cost than traditional SQL for similar volumes of data.
- You can use Table storage to store flexible datasets like user data for web applications, address books, device information, or other types of metadata your service requires. You can store any number of entities in a table, and a storage account may contain any number of tables, up to the capacity limit of the storage account.

Table Storage

- Azure Table storage stores large amounts of structured data. The service is a NoSQL datastore which accepts authenticated calls from inside and outside the Azure cloud.
- Azure tables are ideal for storing structured, non-relational data. Common uses of Table storage include:
  - Storing TBs of structured data capable of serving web scale applications
  - Storing datasets that don't require complex joins, foreign keys, or stored procedures and can be denormalized for fast access
  - Quickly querying data using a clustered index
  - Accessing data using the OData protocol and LINQ queries with WCF Data Service .NET Libraries
- You can use Table storage to store and query huge sets of structured, non-relational data, and your tables will scale as demand increases.

Example

- We will create a single table called "`itemstable`" in Azure Table storage.
- That table will store for use a Pizza Menu and a few items from a Clothing Store. Perhaps this table will support a web site for people who once they buy some nice clothing, they go to eat pizza.

Table properties

- URL format: Code addresses tables in an account using this address format:
- `http://<storage account>.table.core.windows.net/<table>`
- You can address Azure tables directly using this address with the OData protocol.
- Storage Account: All access to Azure Storage is done through a storage account.
- Table: A table is a collection of entities. Tables don't enforce a schema on entities, which means a single table can contain entities that have different sets of properties. The number of tables that a storage account can contain is limited only by the storage account capacity limit.
- Entity: An entity is a set of properties, similar to a database row. An entity can be up to 1MB in size.
- Properties: A property is a name-value pair. Each entity can include up to 252 properties to store data. Each entity also has three system properties that specify a `partition key`, a `row key`, and a `timestamp`.
- Entities with the same partition key can be queried more quickly, and inserted/updated in atomic operations.
- An entity's row key is its unique identifier within a partition.

Code to Create and Insert data in an Azure Table

```
import string,random,time,azurerm,json
from azure.storage.table    import TableService, Entity

# Define variables to handle Azure authentication
auth_token = azurerm.get_access_token_from_cli()
subscription_id = azurerm.get_subscription_from_cli()

# Define variables with random resource group and storage account names
resourcegroup_name = 'zdj'+''.join(random.choice(string.ascii_lowercase + string.digits) for _ in range(6))
storageaccount_name = 'zdj'+''.join(random.choice(string.ascii_lowercase + string.digits) for _ in range(6))
location = 'eastus'
###
# Create the a resource group for our demo
# We need a resource group and a storage account.
# A random name is generated, as each storage account name must be globally unique.
###
response = azurerm.create_resource_group(auth_token, subscription_id, resourcegroup_name, location)
if response.status_code == 200 or response.status_code == 201:
   print('Resource group: ' + resourcegroup_name + ' created successfully.')
else:
   print('Error creating resource group')

# Create a storage account for our demo
response = azurerm.create_storage_account(auth_token, subscription_id,
      resourcegroup_name, storageaccount_name,  location, storage_type='Standard_LRS')
if response.status_code == 202:
   print('Storage account: ' + storageaccount_name + ' created successfully.')
   time.sleep(2)
else:
   print('Error creating storage account')
```

Code to Create and Insert data in Azure Table

```
## Use the Azure Storage Storage SDK for Python to create a Table
print('\nLet\'s create an Azure Storage Table to store some data.')
raw_input('Press Enter to continue...')
# Each storage account has a primary and secondary access key.
# These keys are used by aplications to access data in your storage account, such as Tables.
# Obtain the primary storage access key for use with the rest of the demo
response = azurerm.get_storage_account_keys(auth_token, subscription_id, resourcegroup_name, storageaccount_name)
storageaccount_keys = json.loads(response.text)
storageaccount_primarykey = storageaccount_keys['keys'][0]['value‘]
## Create the Table with the Azure Storage SDK and the access key obtained in the previous step
table_service = TableService(account_name=storageaccount_name, account_key=storageaccount_primarykey)
response = table_service.create_table('itemstable')
if response == True:
   print('Storage Table: itemstable created successfully.\n')
else:
   print('Error creating Storage Table.\n')

time.sleep(1)
## Use the Azure Storage Storage SDK for Python to create some entries in the Table
print('Now let\'s add some entries to our Table.\nRemember, Azure Storage Tables is a NoSQL datastore.’)
print(‘ This is similar to adding records to a database.')
raw_input('Press Enter to continue...')
# Each entry in a Table is called an 'Entity'.
# Here, we add an entry for first pizza with two pieces of data - the name, and the cost
#
```

Code to Create and Insert data in Azure Table

```
# A partition key tracks how like-minded entries in the Table are created and queried.
# A row key is a unique ID for each entity in the partition
# These two properties are used as a primary key to index the Table. This makes queries much quicker.

pizza = Entity()
pizza.PartitionKey = 'pizzamenu'
pizza.RowKey = '001'
pizza.description = 'Pepperoni'
pizza.cost = 18
table_service.insert_entity('itemstable', pizza)
print('Created entry for pepperoni...')

pizza = Entity()
pizza.PartitionKey = 'pizzamenu'
pizza.RowKey = '002'
pizza.description = 'Veggie'
pizza.cost = 15
table_service.insert_entity('itemstable', pizza)
print('Created entry for veggie...')

pizza = Entity()
pizza.PartitionKey = 'pizzamenu'
pizza.RowKey = '003'
pizza.description = 'Hawaiian'
pizza.cost = 12
table_service.insert_entity('itemstable', pizza)
print('Created entry for Hawaiian...\n')
```

Code to Create and Insert data in Azure Table

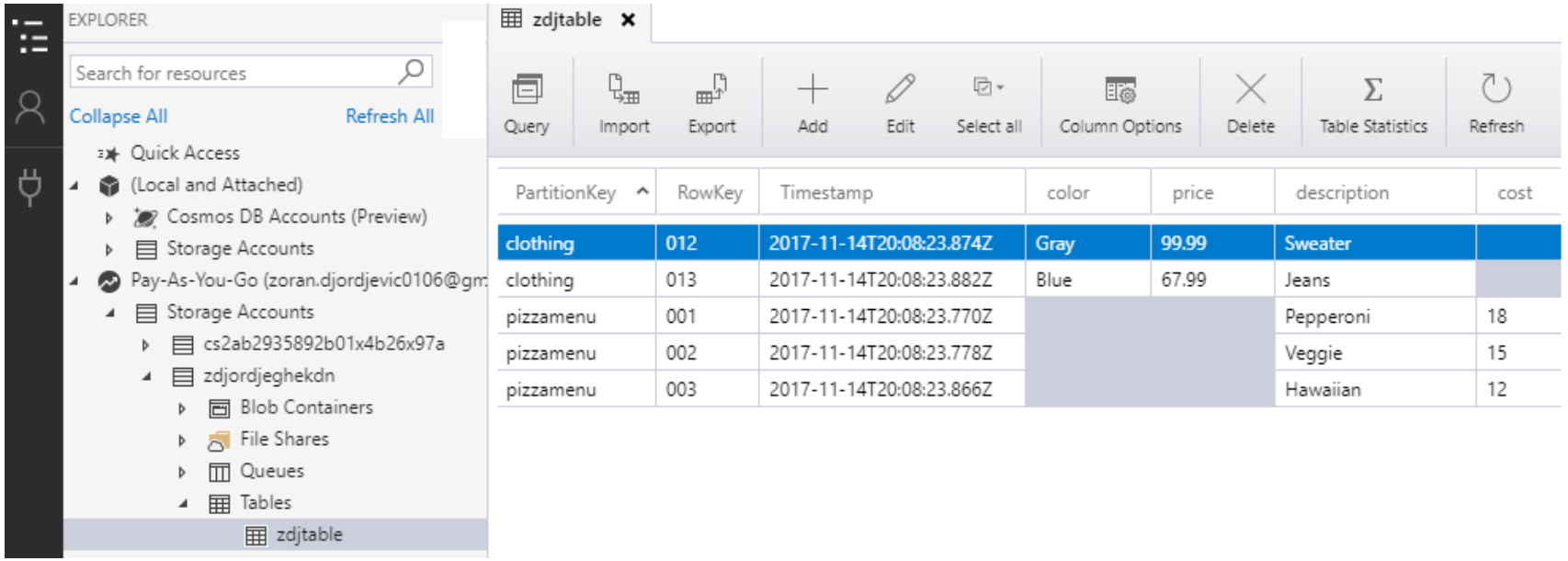
```
# A partition key tracks how like-minded entries in the Table are created and queried.
# A row key is a unique ID for each entity in the partition
# These two properties are used as a primary key to index the Table. This makes queries much quicker.

clothing = Entity()
clothing.PartitionKey = 'clothingstore'
clothing.RowKey = '005'
clothing.sku = 'BLK203123'
clothing.item = 'sweater'
clothing.cost = 22.99
table_service.insert_entity('itemstable', clothing)
print('Created entry for a Sweater...\n')
time.sleep(1)

clothing = Entity()
clothing.PartitionKey = 'clothingstore'
clothing.RowKey = '006'
clothing.sku = 'BLK203143'
clothing.item = 'jeans'
clothing.cost = 55.99
table_service.insert_entity('itemstable', clothing)
print('Created entry for Jeans...\n')
time.sleep(1)

###
# Use the Azure Storage Storage SDK for Python to query for entities in our Table
###
print('With some data in our Azure Storage Table, we can query the data.\n’
print(‘Let\'s see what the pizza menu looks like.')
raw_input('Press Enter to continue...')
```

Code to Create and Insert data in Azure Table

```
# In this query, you define the partition key to search within, and then which properties to retrieve
# Structuring queries like this improves performance as your application scales up and keeps the queries efficient
items = table_service.query_entities('itemstable', filter="PartitionKey eq 'pizzamenu'", select='description,cost')
for item in items:
    print('Name: ' + item.description)
    print('Cost: ' + str(item.cost) + '\n')

items = table_service.query_entities('itemstable', filter="PartitionKey eq 'clothingstore'", select='description,price')
for item in items:
    print('Name: ' + item.description)
    print('Price: ' + str(item.price) + '\n')

time.sleep(1)
## This was a quick demo to see Tables in action.
# Although the actual cost is minimal (fractions of a cent per month) for the three entities we created,
# it's good to clean up resources when you're done
###
print('\nWe should  clean up the Azure Storage resources we created.')
raw_input('Press Enter to continue...')
response = table_service.delete_table('itemstable')
if response == True:
    print('Storage table: itemstable deleted successfully.')
else:
    print('Error deleting Storage Table')

response = azurerm.delete_resource_group(auth_token, subscription_id, resourcegroup_name)
if response.status_code == 202:
    print('Resource group: ' + resourcegroup_name + ' deleted successfully.')
else:
    print('Error deleting resource group.')
```

View of Table Data with Azure Storage Explorer

- If you navigate to your Storage Account and find newly generated table you will see something like this:



- Notice that information about Pizzas and Clothing does not share the same schema.
- Some might recognize a denormalized table

Getting the Script to Azure Cloud Shell

- To get that script to the shell (file share) upload it to your GitHub repository and use git to transfer it to your “home” directory in the Cloud:

```
$ git clone https://github.com/yourrepository/table_demo.py
```

- You need a few Python packages for Azure. On the command prompt type:

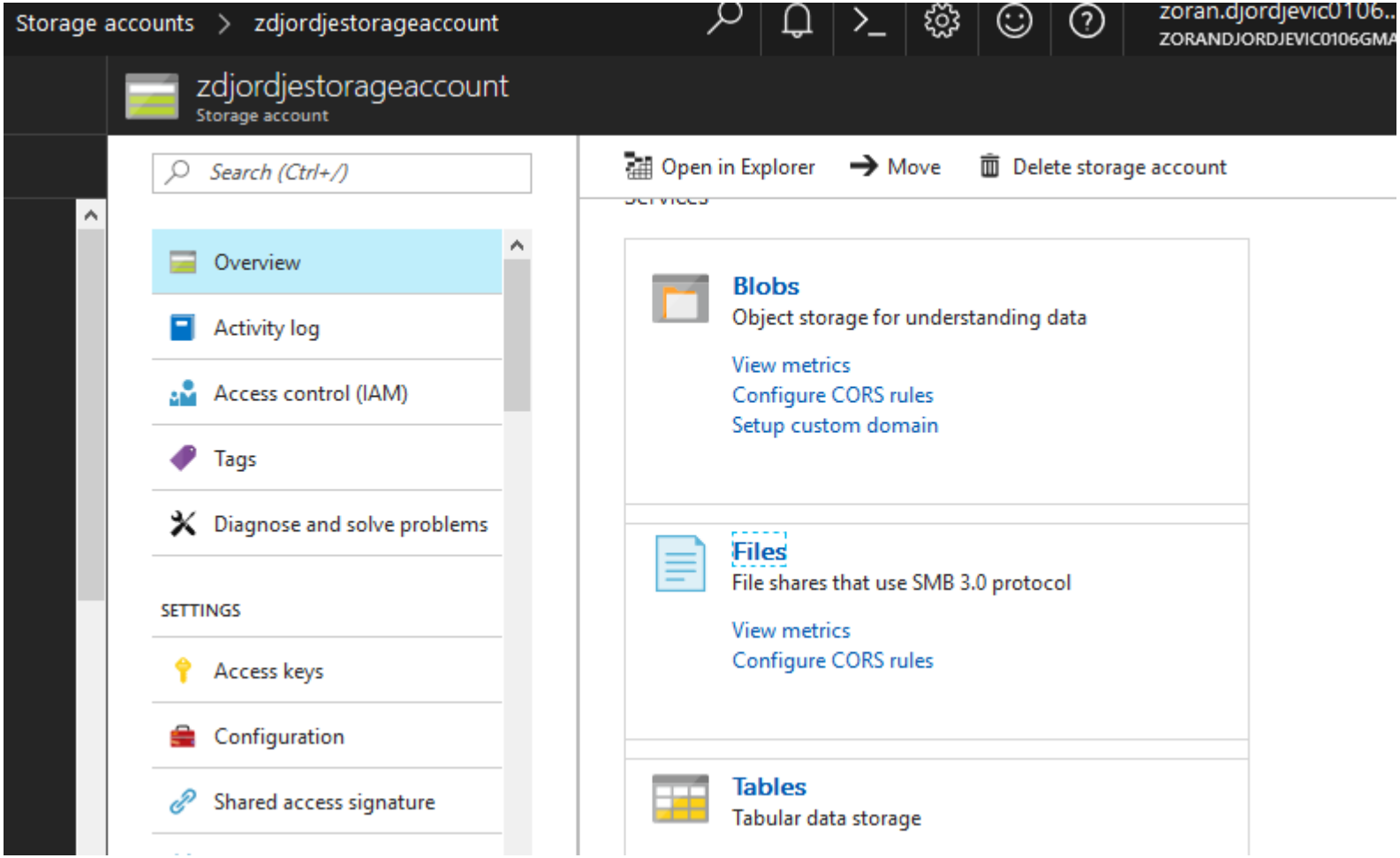
```
$ pip2 install --user azurerm azure-storage
```

- You need `--user` when you install the packages in Azure shell?
- In Azure Cloud Shell, you cannot install packages in the core system. You don’t have permission. Instead, the packages are installed in your user’s environment. These package installs should persist across sessions.
- Now, you can run your script:

```
$ python2.7  table_demo.py
```

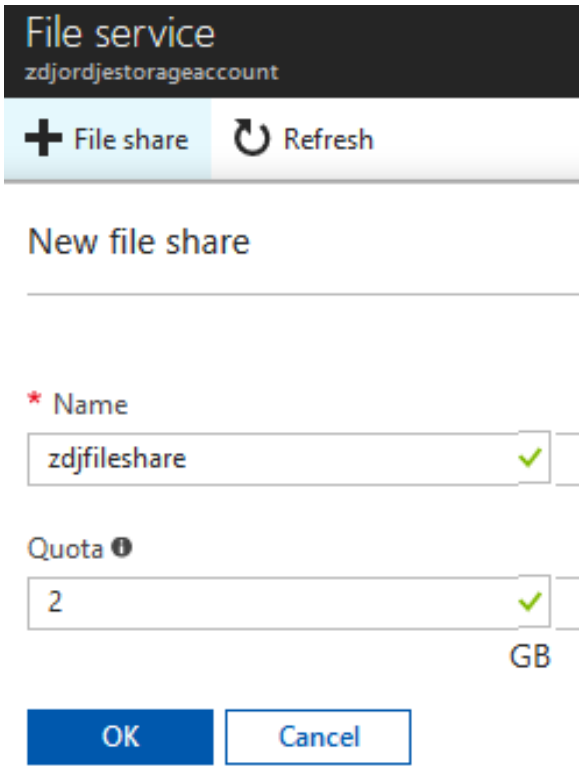
File Share

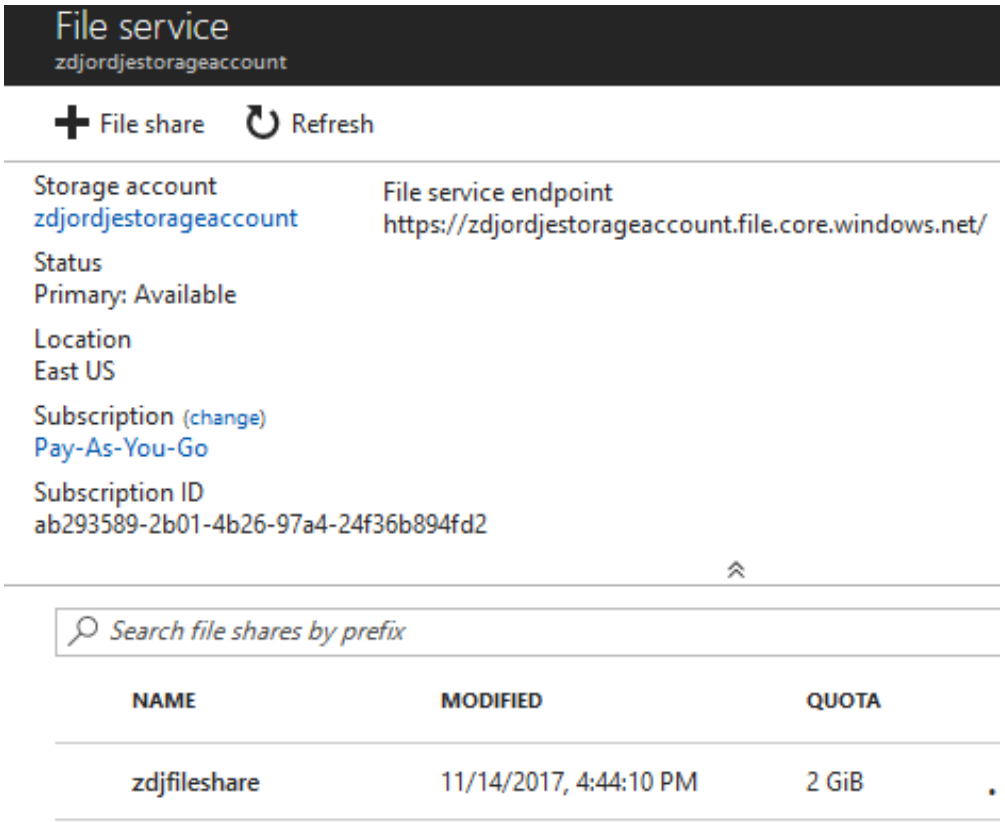
- You can create File Share in Portal. Open one of your storage accounts and click on Files. On the following screen click on + File share



File service wizard

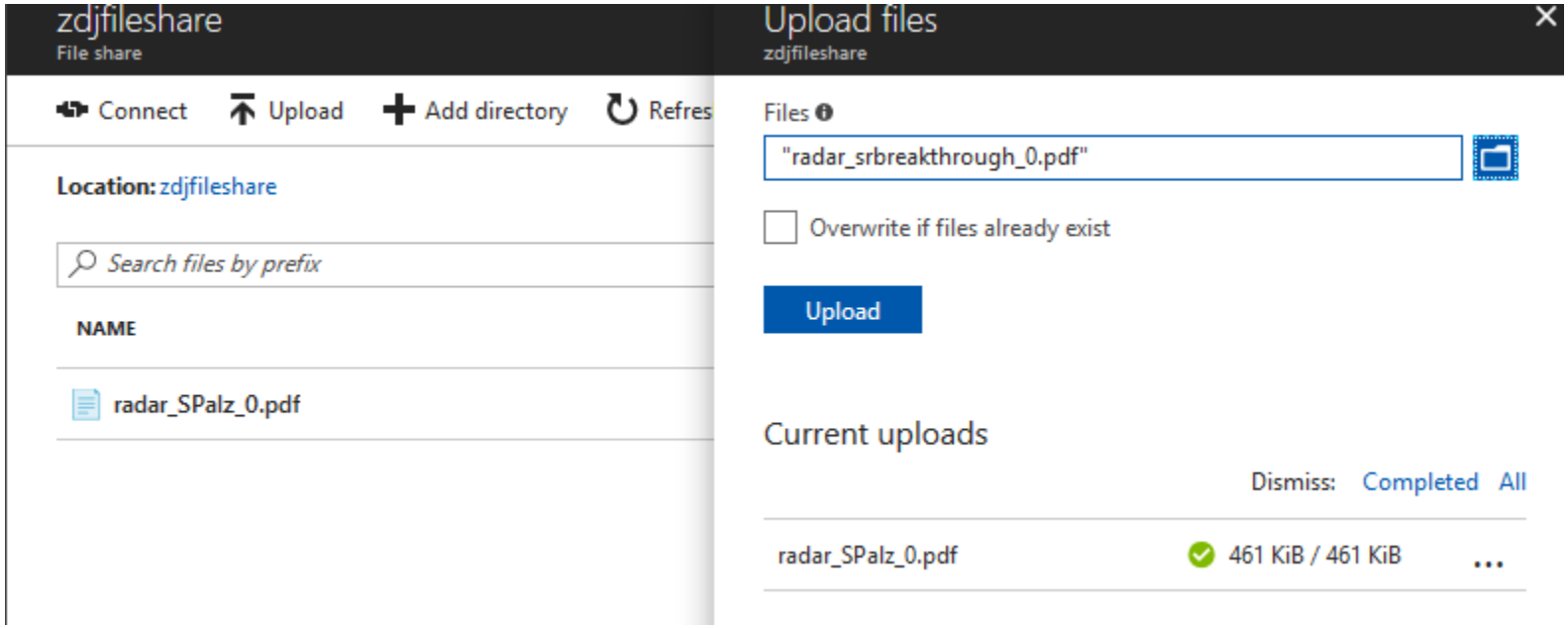
- Provide name and Quota of new file share.





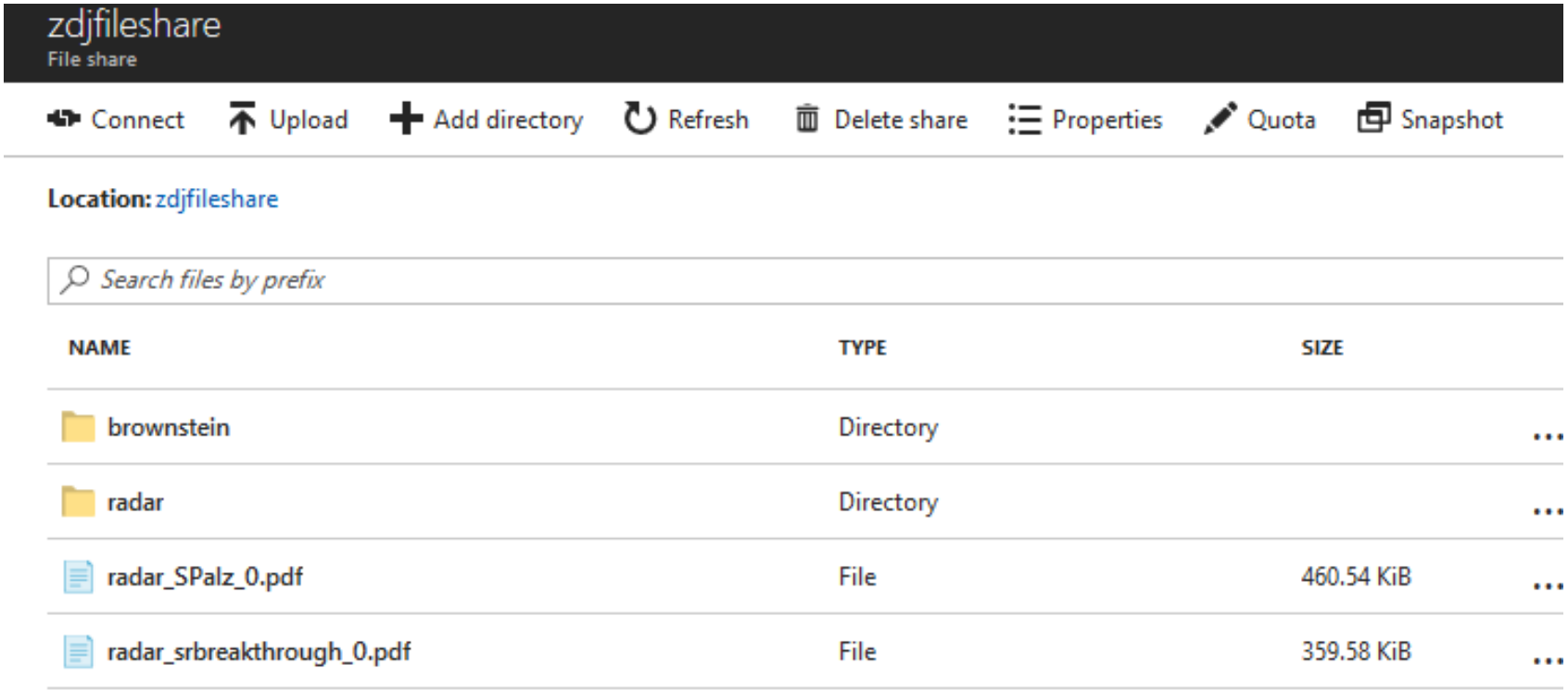
Upload a file

- Click on You File Share Name. Hit Upload and then use Upload files widget to navigate to a file on your OS. Select a file and hit Upload



Create Directories

- Once you open the File service wizard you can also create directories and navigate to those directories



Create file share through CLI

- Create a connection string to the storage account where you want to create the share.
- Replace `<storage-account>` and `<resource_group>` with your storage account name and resource group in the following example:

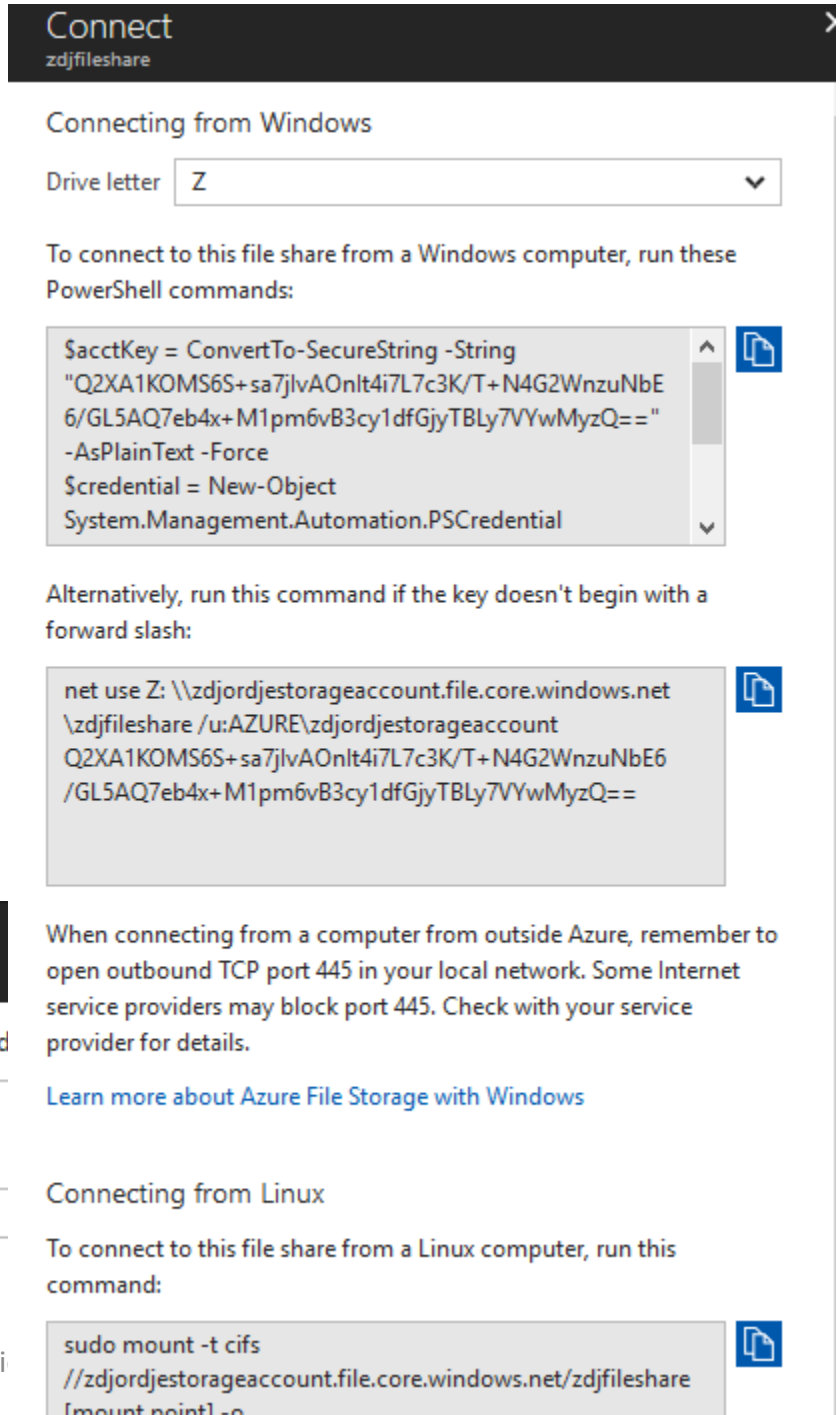
```
current_env_conn_string = $(az storage account show-connection-string -n
<storage-account> -g <resource-group> --query 'connectionString' -o tsv)

 if [[ $current_env_conn_string == "" ]]; then
     echo "Couldn't retrieve the connection string."
 fi
```

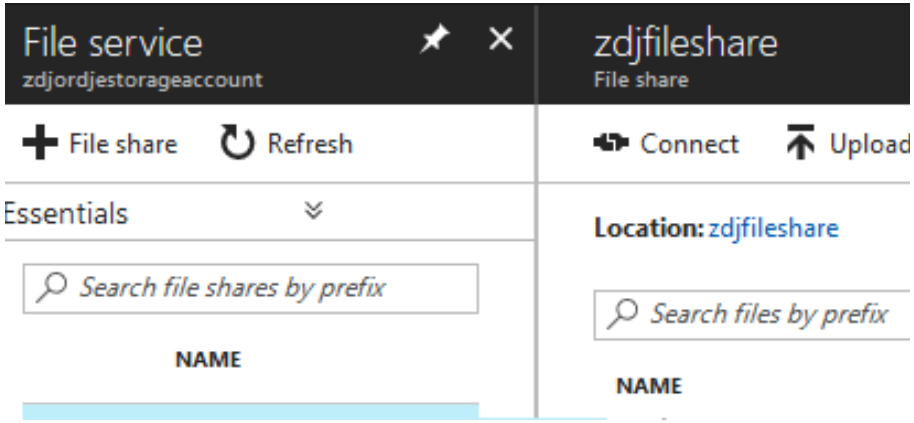
- Create file share

```
az storage share create --name files --quota 2048 --connection-
string $current_env_conn
```

File Share as Windows drive

- Back in Azure Portal on File service Wizard, select Connect icon.
- Choose drive letter, e.g. Z
- Open outbound port 445 on your machine's firewall.
- Copy and run the first script on PowerShell prompt.
- Or, copy and run the second command on DOS prompt.





Mounting File Share on Ubuntu

- First open port 445

```
$ sudo ufw enable
$ sudo ufw allow 445
```

- Install cifs-utils

```
$ sudo apt-get install cifs-utils
$ sudo mkdir /mnt/winshare
$ sudo chmod 777 /mnt/winshare
```

- I have an Azure File share in one of my storage accounts. My command for mounting Azure file share on Linux reads, all on one line.

```
$ sudo mount -t cifs
//zdjordjestorageaccount.file.core.windows.net/zdjordjefileshare
/mnt/winshare -o
vers=3.0,username=zdjordjestorageaccount,password=Kiou8gaKPds3jbmgQ/oE5wRHF
tsi1P43TgqIk3FkCY4iOPnFgyHrQd/kUjw1EmaEMpZfhwV8RvCOPxMqVLCkog==,dir_mode=07
77,file_mode=0777,sec=ntlmssp
```

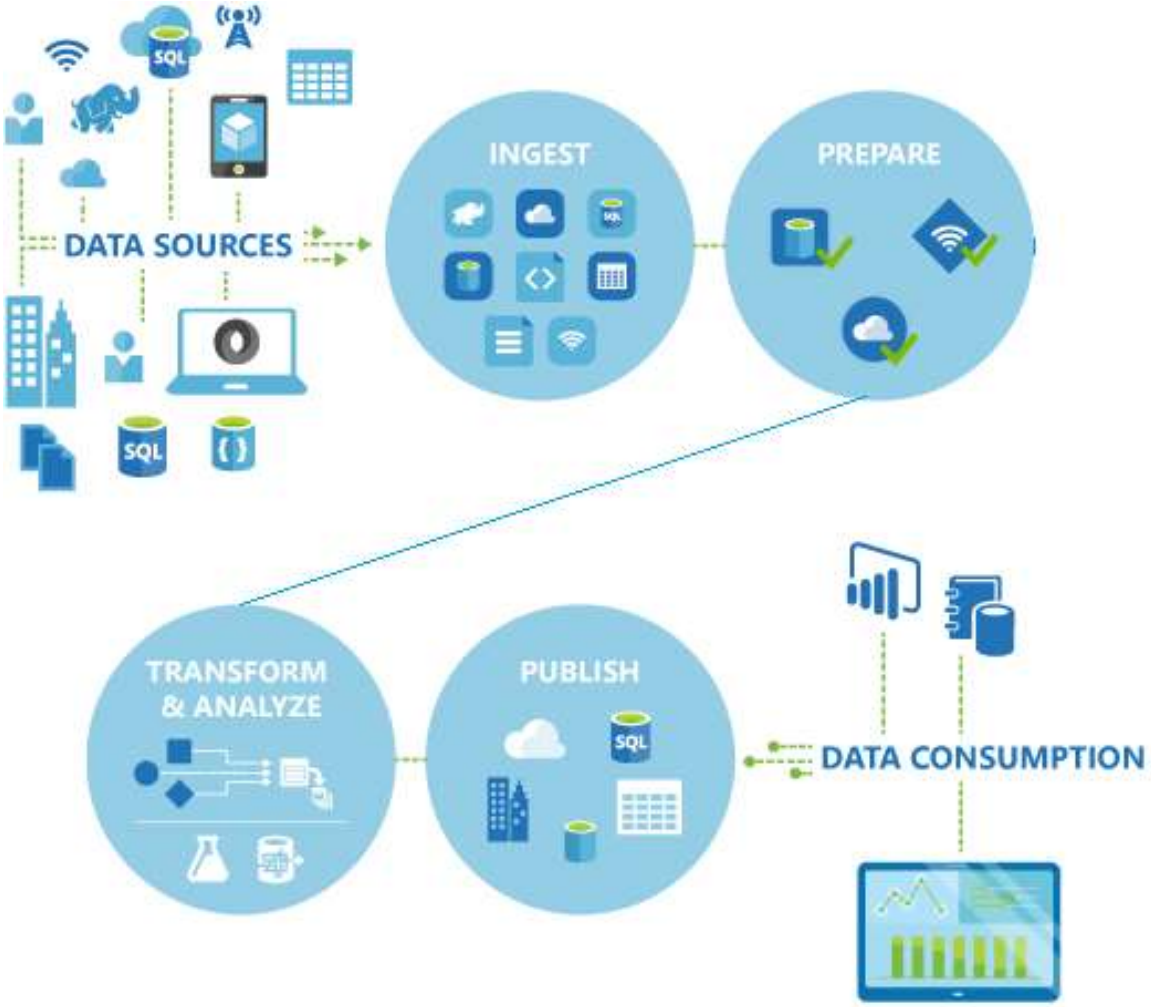
- I uploaded an image file, `Earnings.png`, to my File share. On Ubuntu, I did:

```
$  ls /mnt/winshare
Earnings.png
```

- File uploaded to the File share in Azure is visible on my Linux machine.
- Before removing your Azure File share, u(n)mount the directory `/mnt/winshare`
- `$ umount /mnt/winshare`

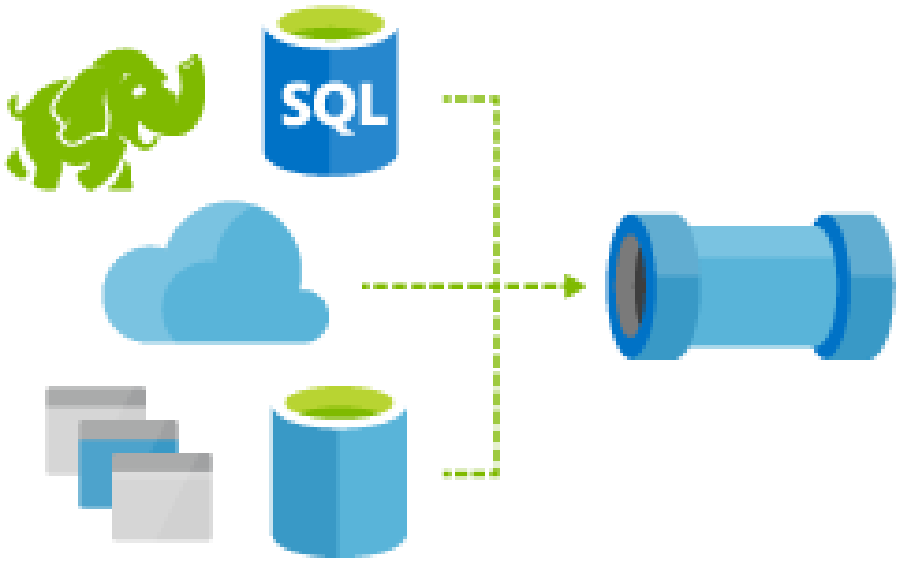
Data Factory

- Azure Data Factory is a hybrid data integration service that allows you to create, schedule and orchestrate your ETL/ELT workflows at scale.
- ADF will fetch the data wherever your data lives, in cloud or self-hosted networks.
- Azure Data Factory's extensive capabilities will help your meet security and compliance needs.



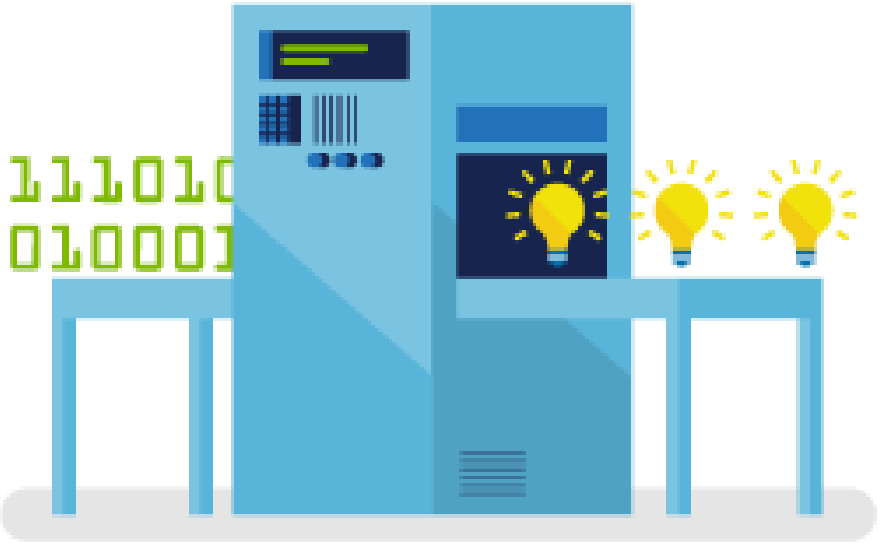
Ingest and prepare

- Use Azure Data Factory, a globally-deployed data movement service in the cloud, with ability to ingest data from multiple on-premises and cloud sources.
- ADF will help you
  - orchestrate your data integration workflows wherever your data lives,
  - accelerate your data integration with multiple data connectors, and
  - prepare and partition your data as you ingest it or apply pre-processing steps.



Transform and Analyze

- ADF will schedule and manage your data transformation and analysis process.
- With ADF you can choose from a wide range of processing services, and put them into managed data pipelines to use the best tool for the job.
- For example, you can have
  - Hadoop processing step for big or semi-structured data,
  - stored procedure invocation step for structured data,
  - machine-learning step for analytics, or
  - insert your own custom code as a processing step in any pipeline.



Use **data pipelines** to transform raw data into finished or shaped data that's ready for consumption by BI tools or applications. Get your valuable data where it needs to go for consumption by your on-premises or cloud applications and services.

Pricing

- Pricing is somewhat complex. There are several elements that determine the price of your operations:
  - **Number of activities run.** First 50,000 activity runs—$0.55 per 1,000 runs
    Beyond 50,000 activity runs—$0.50 per 1,000 runs
  - **Volume of data moved.** $0.02 /GB
  - **SQL Server Integration Services (SSIS) compute hours**. 4 core, 8GB RAM $0.42/hour
  - **Whether a pipeline is active or not.** $0.125 per hour, per GB

Operationalized Analytic Solutions

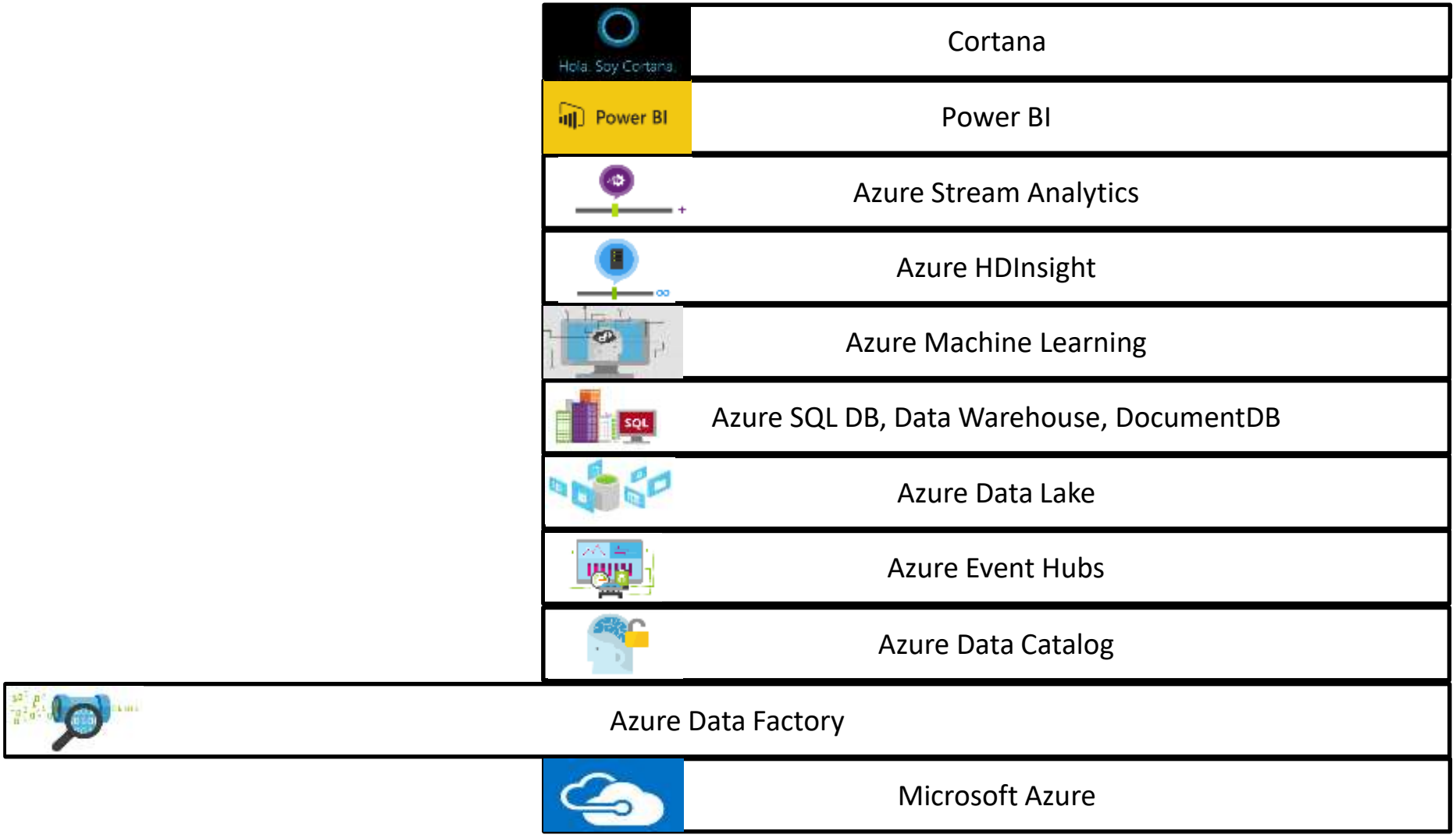
Example of Microsoft Usage, Cortana

- Cortana Analytics Suite delivers an end-to-end platform with integrated and comprehensive set of tools and services to help you build intelligent applications that let you easily take advantage of Advanced Analytics.
- First Cortana Analytics Suite provides services to bring data in, so that you can analyze it. It provides information management capabilities like Azure Data Factory so that you can pull data from any source (relational DB like SQL or non-relational ones like your Hadoop cluster) in an automated and scheduled way, while performing the necessary data transforms (like setting certain data colums as dates vs. currency etc). Think ETL (Extract, Transform, Load) in the cloud. Event hub does the same for IoT type ingestion of data that streams in from lots of end points.

- The data brought in then can be persisted in flexible big data storage services like Data Lake and Azure SQL DW.
- You can then use a wide range of analytics services from Azure ML to Azure HDInsight to Azure Stream Analytics to analyze the data that are stored in the big data storage. This means you can create analytics services and models specific to your business need (say real time demand forecasting).

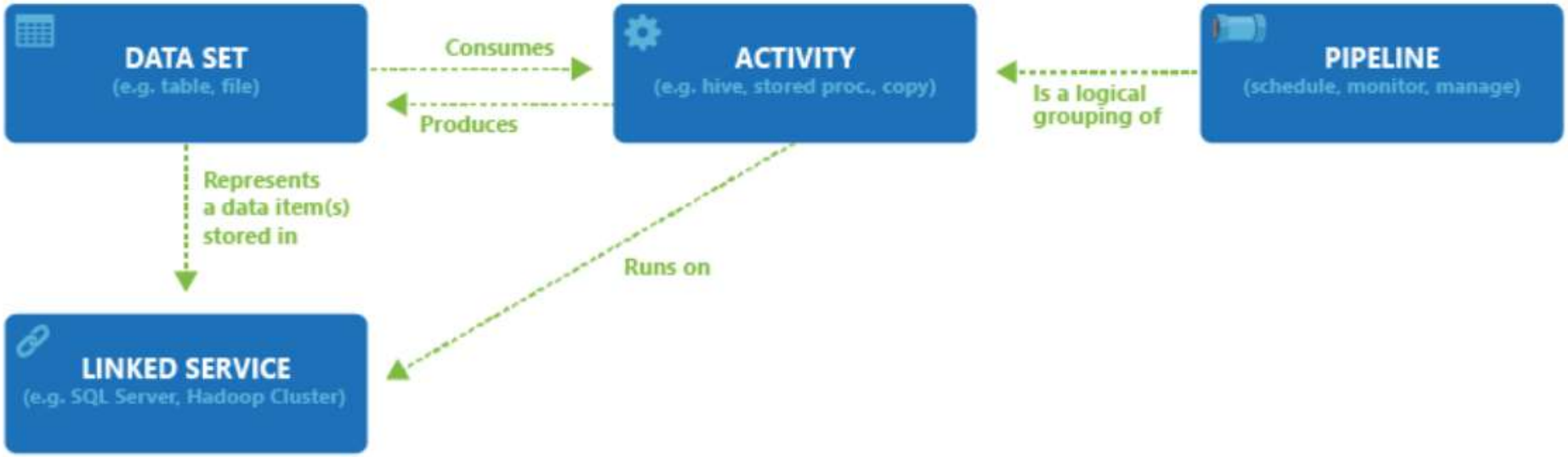
- The resultant analytics services and models created by taking these steps can then be surfaced as interactive dashboards and visualizations via Power BI

Cortana Analytics Stack



@Zoran B. Djordjevic, Nishava, Inc.

ADF Components

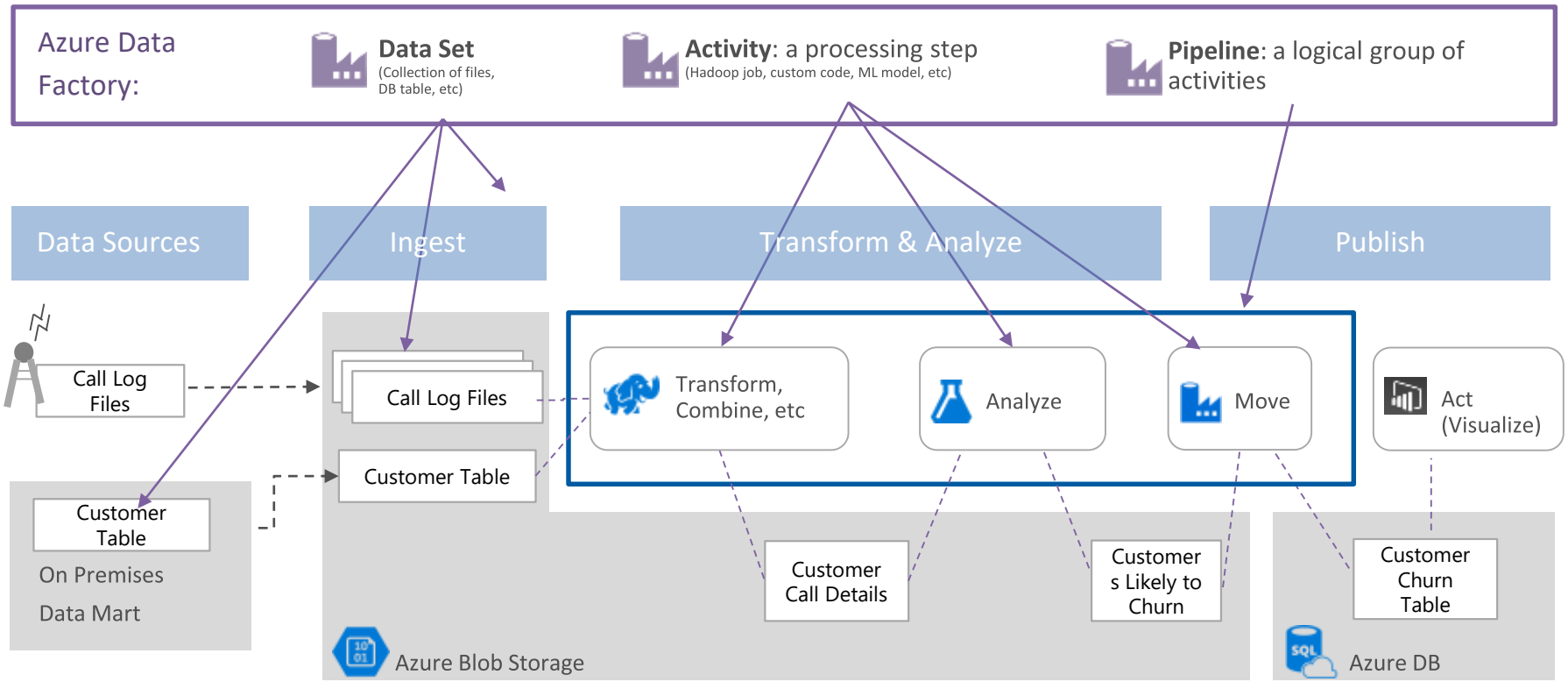


ADF Process

1. **Define Architecture:** Set up objectives and flow
2. **Create the Data Factory:** Portal, PowerShell, VS
3. **Create Linked Services:** Connections to Data and Services
4. **Create Datasets**: Input and Output
5. **Create Pipeline:** Define Activities
6. **Monitor and Manage:** Portal or PowerShell, Alerts and Metrics

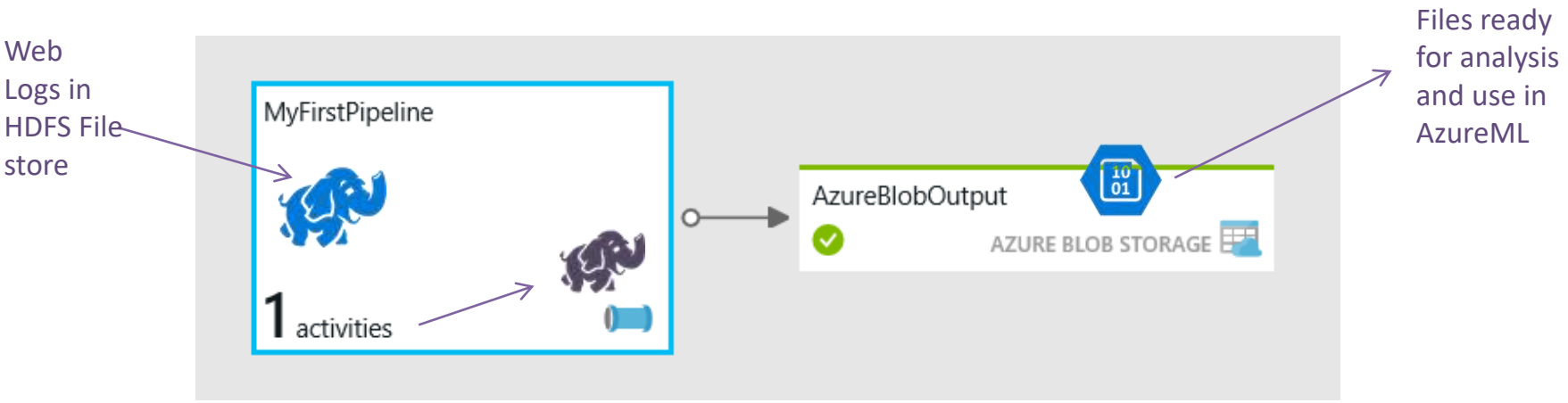
Design Process

- Define data sources, processing requirements, and output – also management and monitoring

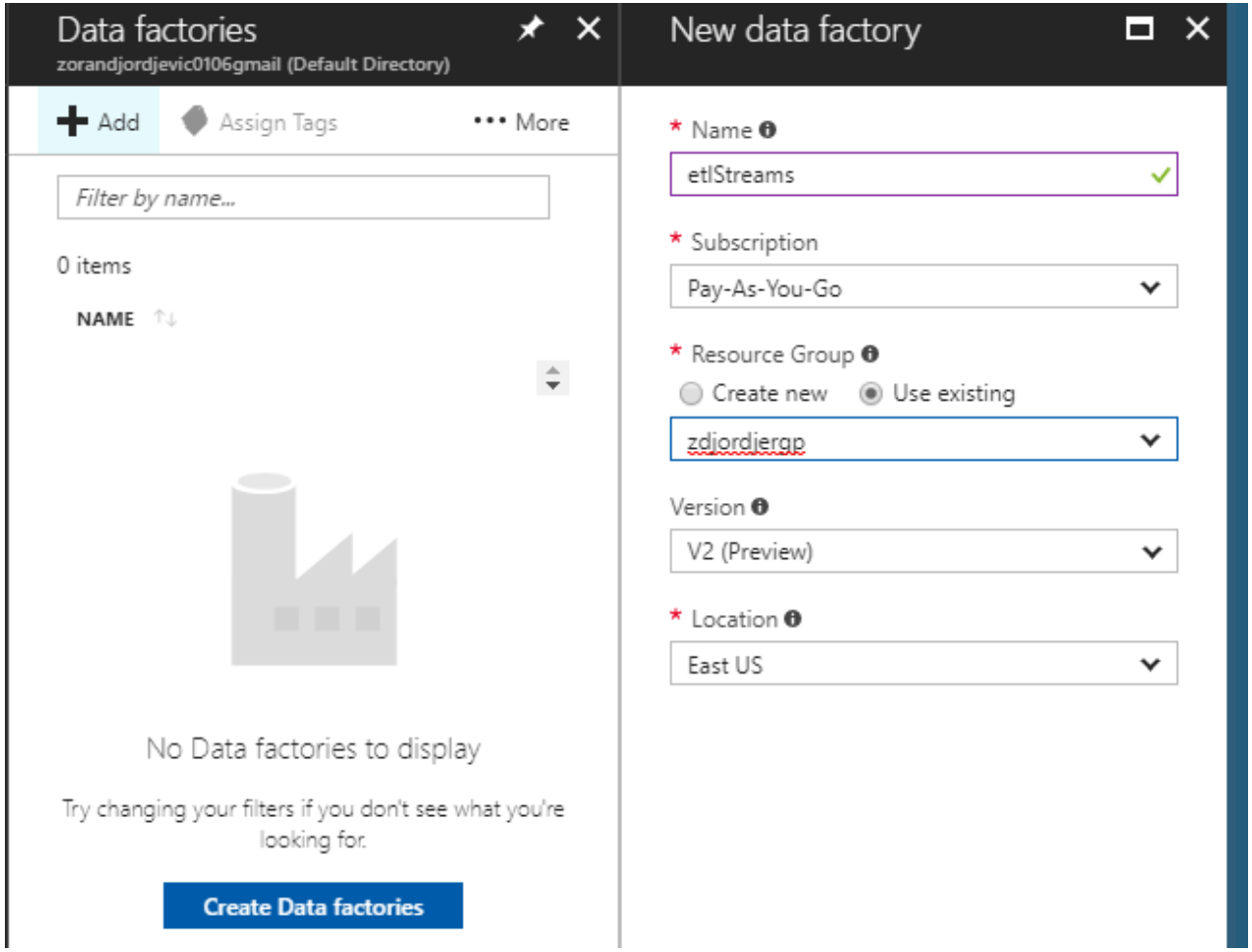


Typical ADF:

- Transform and Analyze Web Logs each month
- Transform Raw Weblogs stored in a temporary location, using a Hive Query, storing the results in Blob Storage



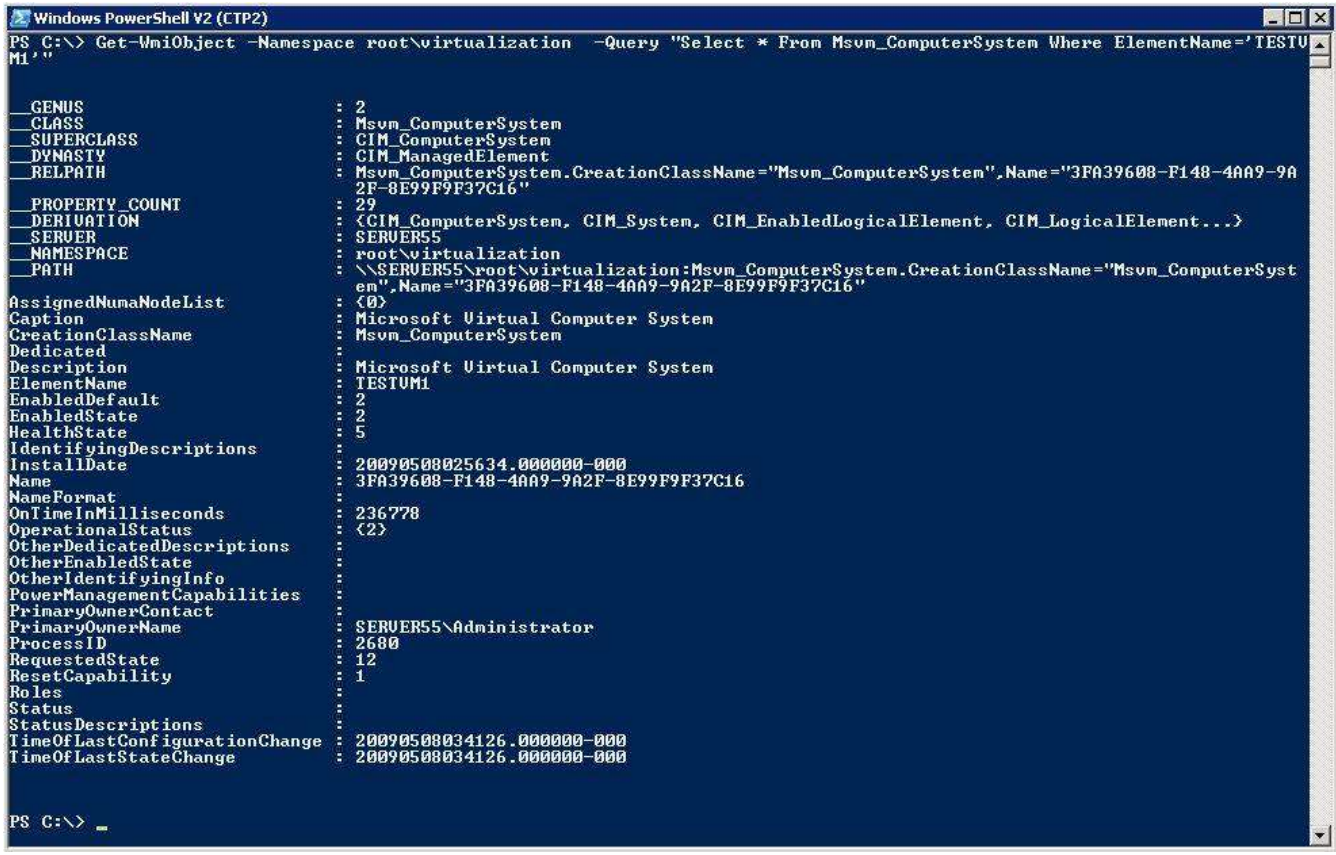
Creating Data Factory Using the Portal



@Zoran B. Djordjevic, Nishava, Inc.

Create ADF Using PowerShell

- Use in MS Clients
- Use for Automation
- Use for quick set up and tear down

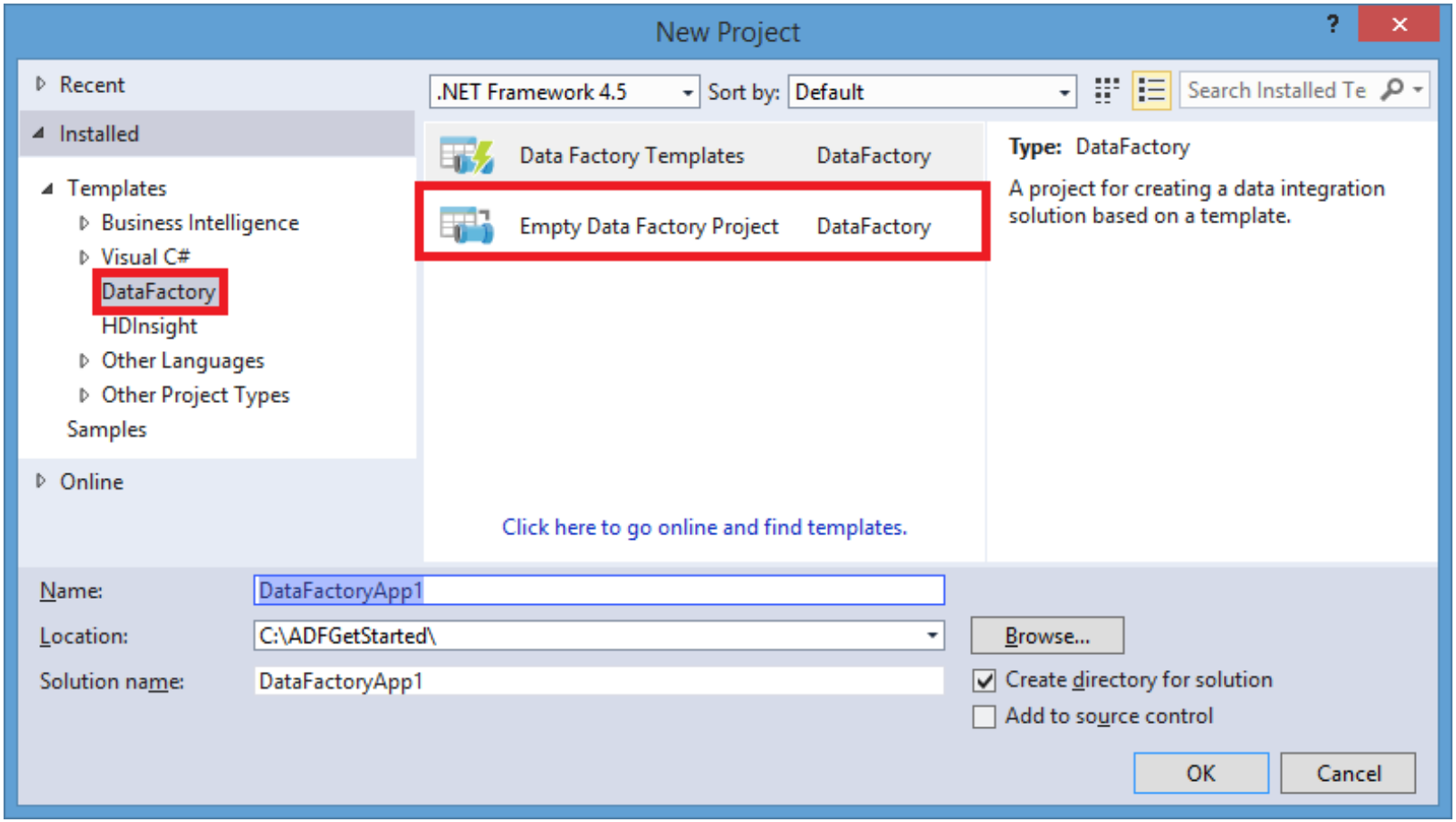


PowerShell ADF Example

1. Run Add-AzureAccount and enter the user name and password
2. Run Get-AzureSubscription to view all the subscriptions for this account.
3. Run Select-AzureSubscription to select the subscription that you want to work with.
4. Run Switch-AzureMode AzureResourceManager
5. Run New-AzureResourceGroup -Name ADFTutorialResourceGroup -Location "West US"
6. Run New-AzureDataFactory -ResourceGroupName ADFTutorialResourceGroup –Name DataFactory(your alias)Pipeline –Location "West US"

Create ADF, Using Visual Studio

- Use in mature dev environments
- Use when integrated into larger development process



Create Linked Services, Data Options

Source	Sink
Blob	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, DocumentDB, OnPrem File System, Data Lake Store
Table	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, DocumentDB, Data Lake Store
SQL Database	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, DocumentDB, Data Lake Store
SQL Data Warehouse	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, DocumentDB, Data Lake Store
DocumentDB	Blob, Table, SQL Database, SQL Data Warehouse, Data Lake Store
Data Lake Store	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, DocumentDB, OnPrem File System, Data Lake Store

Create Linked Services, Data Options

Source	Sink
SQL Server on IaaS	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, Data Lake Store
OnPrem File System	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, OnPrem File System, Data Lake Store
OnPrem SQL Server	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, Data Lake Store
OnPrem Oracle Database	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, Data Lake Store
OnPrem MySQL Database	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, Data Lake Store
OnPrem DB2 Database	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, Data Lake Store
OnPrem Teradata Database	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, Data Lake Store
OnPrem Sybase Database	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, Data Lake Store
OnPrem PostgreSQL Database	Blob, Table, SQL Database, SQL Data Warehouse, OnPrem SQL Server, SQL Server on IaaS, Data Lake Store

Data Transformation Options

Transformation activity/tools	Compute environment
Hive	HDInsight [Hadoop]
Pig	HDInsight [Hadoop]
MapReduce	HDInsight [Hadoop]
Hadoop Streaming	HDInsight [Hadoop]
Machine Learning activities: Batch Execution and Update Resource	Azure VM
Stored Procedure	Azure SQL
Data Lake Analytics U-SQL	Azure Data Lake Analytics
DotNet	HDInsight [Hadoop] or Azure Batch

Create Dataset Using ARM

```
{
    "name": "<name of dataset>",
    "properties":
    {
       "structure": [ ],
       "type": "<type of dataset>",
        "external": <boolean flag to indicate external data>,
        "typeProperties":
        {
        },
        "availability":
        {

        },
       "policy":
        {

        }
    }.
```

Create Pipelines Using ARM

```
{
    "name": "PipelineName",
    "properties":
    {
        "description" : "pipeline
description",
        "activities":
        [

        ],
        "start": "<start date-time>",
        "end": "<end date-time>"
    }
}
```

Scheduling, Monitoring, Disposition

- Monitoring could helps you identify/locate failures within a pipeline