



Microsoft

Microsoft Homework
Assignment

Name: Roy Atali

Date: 21/02/2023

Table Of Contents

1. Prerequisites Requirements	3
1.1 Azure CLI Installation.....	3
1.2 Python Installation	6
2. Code Snippets.....	12
2.1 createResourceGroup.....	12
2.2 createAccountStorage	13
2.3 createLinuxVirtualMachine	14
2.4 createWindowsVirtualMachine	18
2.5 setNetworkSecurityGroup.....	22
2.6 AssociateNetworkSecurityGroupWithNIC	23
2.7 SetPort3389ForRemoteConnection.....	24
2.8. getStorageAccountNames.....	26
2.9 renewConnectionStrings.....	26
2.10 CopyBlobsFromStorageAccountAToB	27
3. Usage	28
4. Parameters Obtaining.....	29
4.1 Subscription ID	29
5. Automating the task copy_blobs.....	30
6. Monitoring	50
6.1 Server (Virtual Machine Metrics)	53
6.1.1 Server CPU percentage/utilization	53
6.1.2 Server Available Memory.....	53
6.1.3 Server Network Traffic	53
6.2 Source Storage Account Metrics	55
6.2.1 Used capacity	55
6.2.2 Transactions.....	55
6.2.3 Egress.....	55
6.2.4 Ingress.....	55
6.2.5 Availability	56
6.2.6 Latency.....	56
6.3 Destination Storage Account Metrics	57
6.3.1 Used capacity	57
6.3.2 Transactions.....	57
6.3.3 Egress.....	57

6.3.4 Ingress.....	57
6.3.5 Availability	58
6.3.6 Latency.....	58
7. Github Link.....	58

1. Prerequisites Requirements

1.1 Azure CLI Installation

Go to this [link](#). In the Install section click on **Install on Windows**

Run Azure CLI in a Docker container
Run Azure Cloud Shell using Bash
Sign in
> What's new
> Tutorials
> Samples
> How-to guides
> Reference Summaries

Install

The current version of the Azure CLI is **2.45.0**. For information about the latest release, see the [release notes](#). To find your installed version and see if you need to update, run `az version`.

- **Install on Windows**
- [Install on macOS](#)

Click on **Latest release of the azure CLI** and download the .msi file and install it on your machine.

Latest version

Download and install the latest release of the Azure CLI. When the installer asks if it can make changes to your computer, click the "Yes" box.

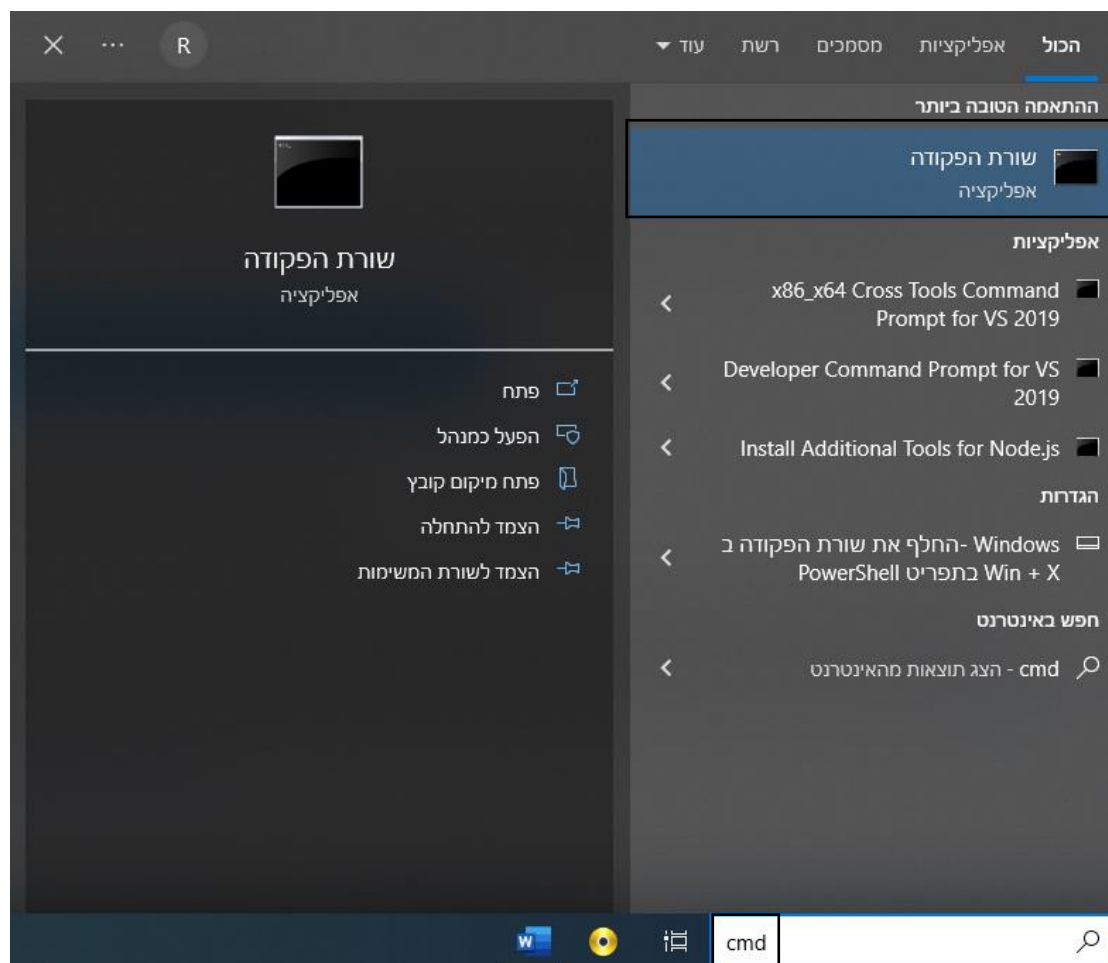
Latest release of the Azure CLI

Specific version

To download the MSI installer for specific version, change the version segment in URL [https://azcliprod.blob.core.windows.net/msi/azure-cli-
<version>.msi](https://azcliprod.blob.core.windows.net/msi/azure-cli-2.45.0(1).msi) and download it. Available versions can be found at [Azure CLI release notes](#).

^ azure-cli-2.45.0 (1).msi

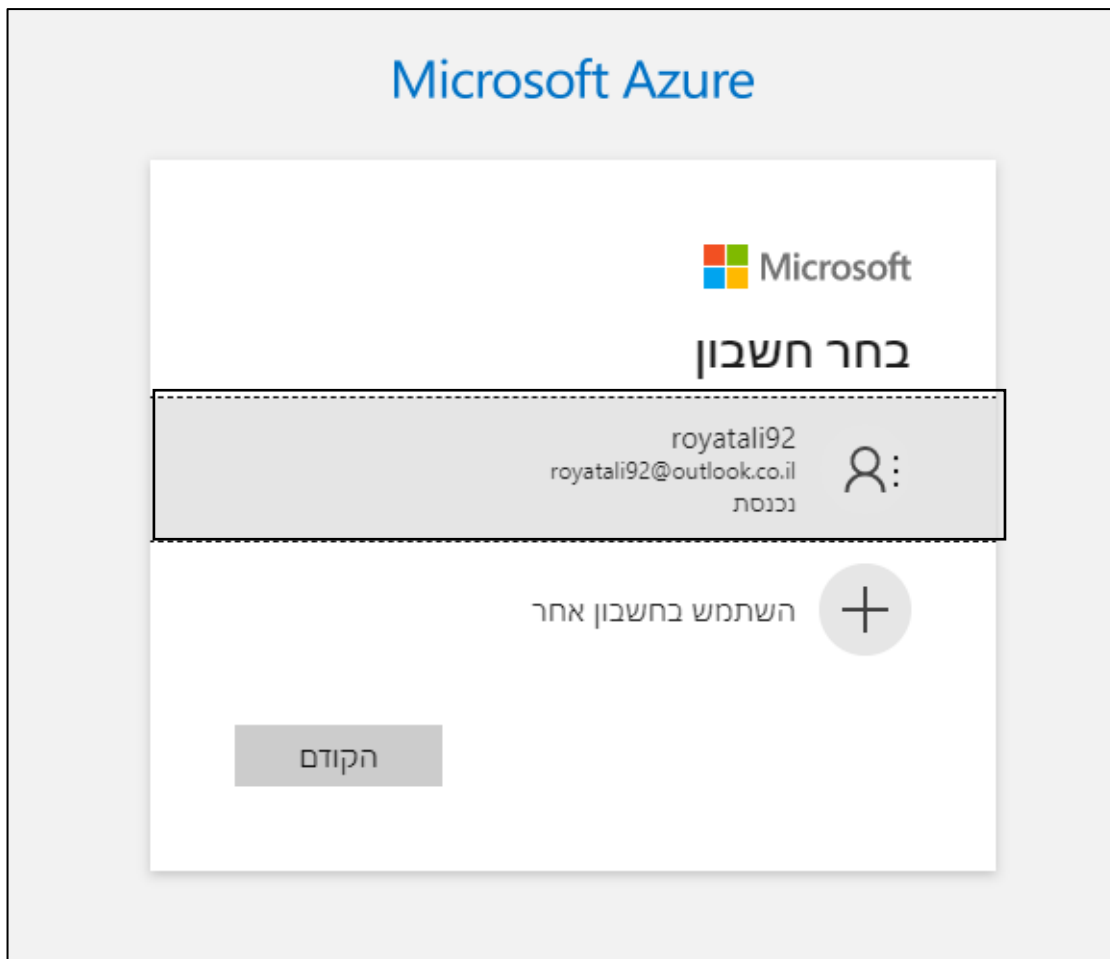
after the installation finished type **cmd** in the search bar and click on it



In the window that opened type the command **az login** and press Enter.

```
שורת הפקודה
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. כל הזכויות שמורות.
C:\Users\royat>az login
```

In the browser select your Microsoft outlook account



This step will authenticate your azure CLI with your account

You have logged into Microsoft Azure!

You can close this window, or we will redirect you to the [Azure CLI documentation](#) in 10 seconds.

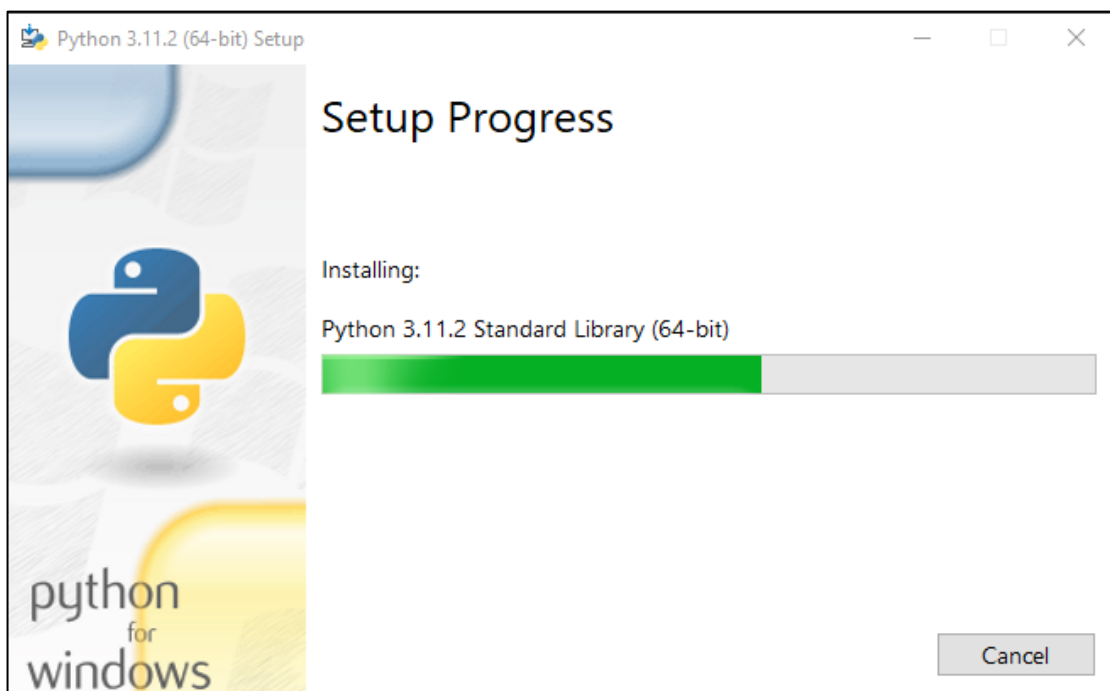
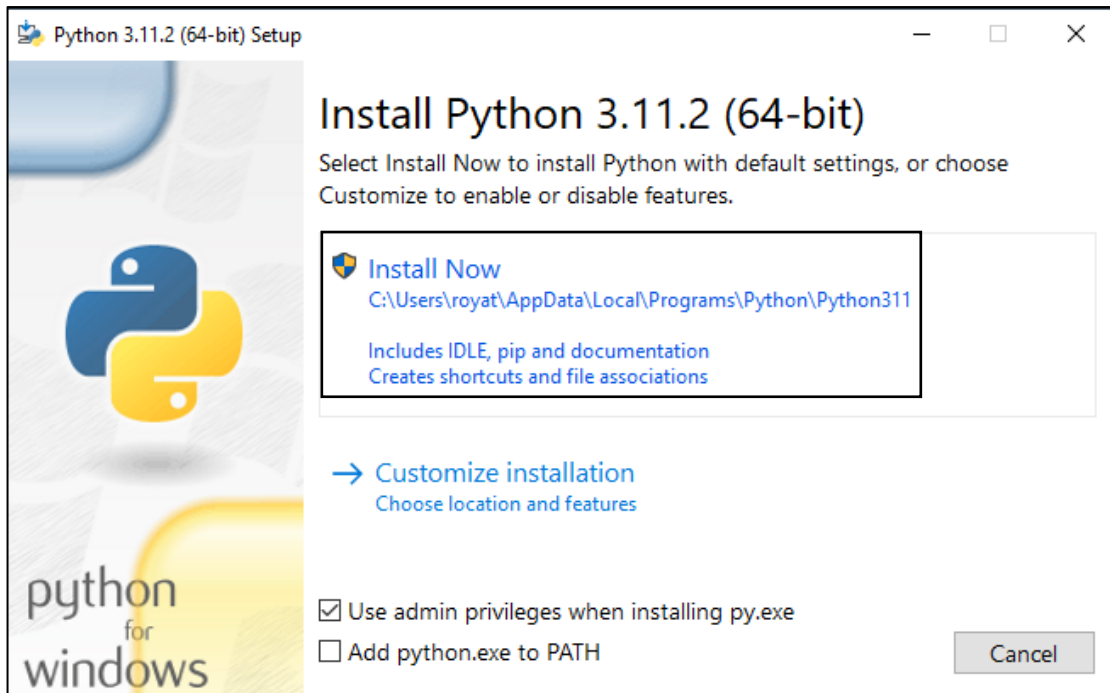
1.2 Python Installation

In order to run the code snippets below you will need to install **Python** first.

1. Go to this [link](#) and download the latest version of python for your machine



2. After it finished downloading. Click open and install Now

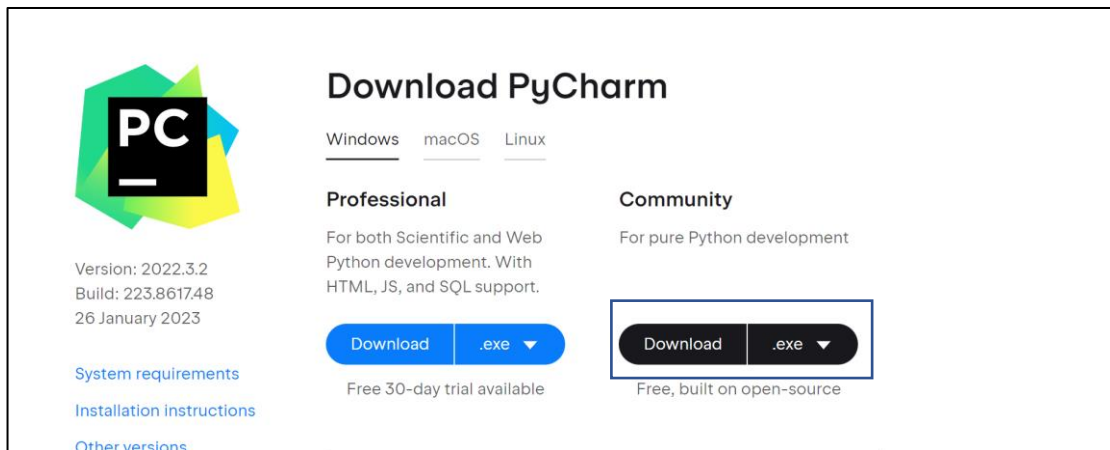


After that you will need **PyCharm**

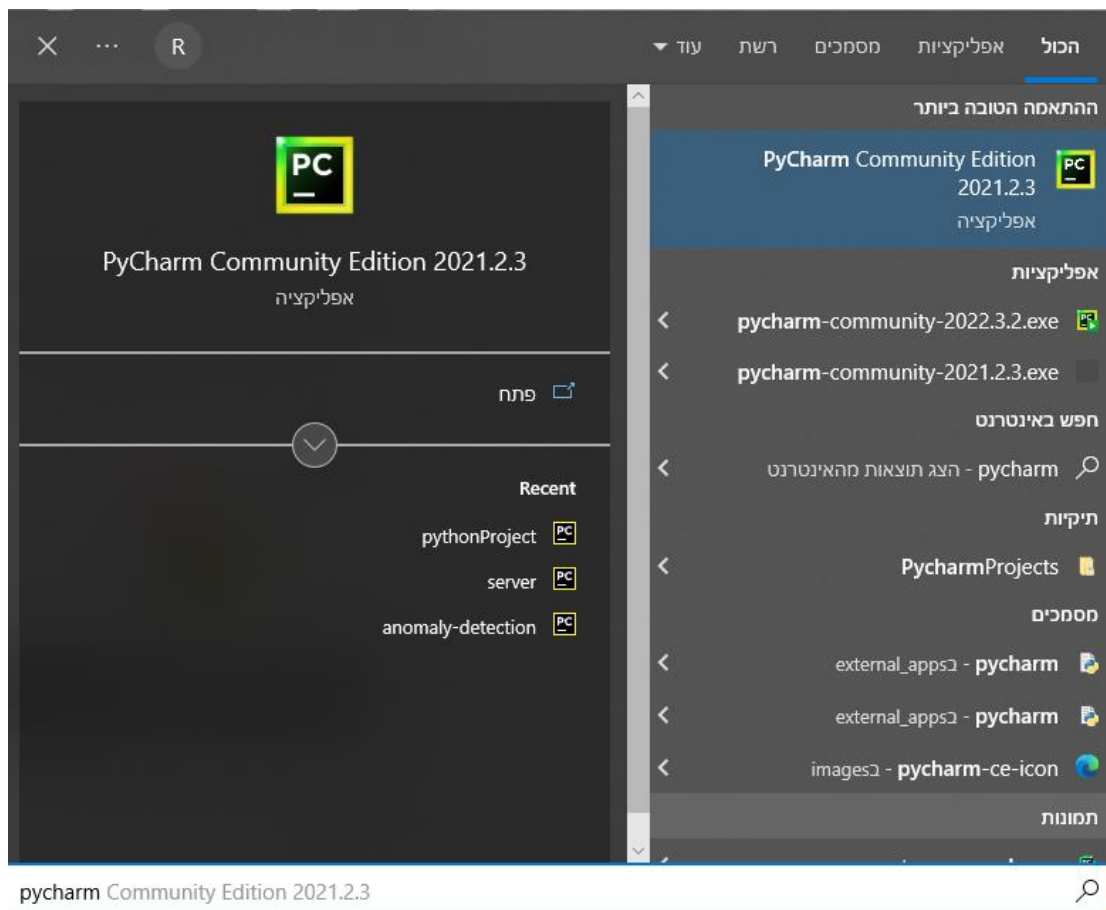
Go to this [link](#) and download **PyCharm** for your machine



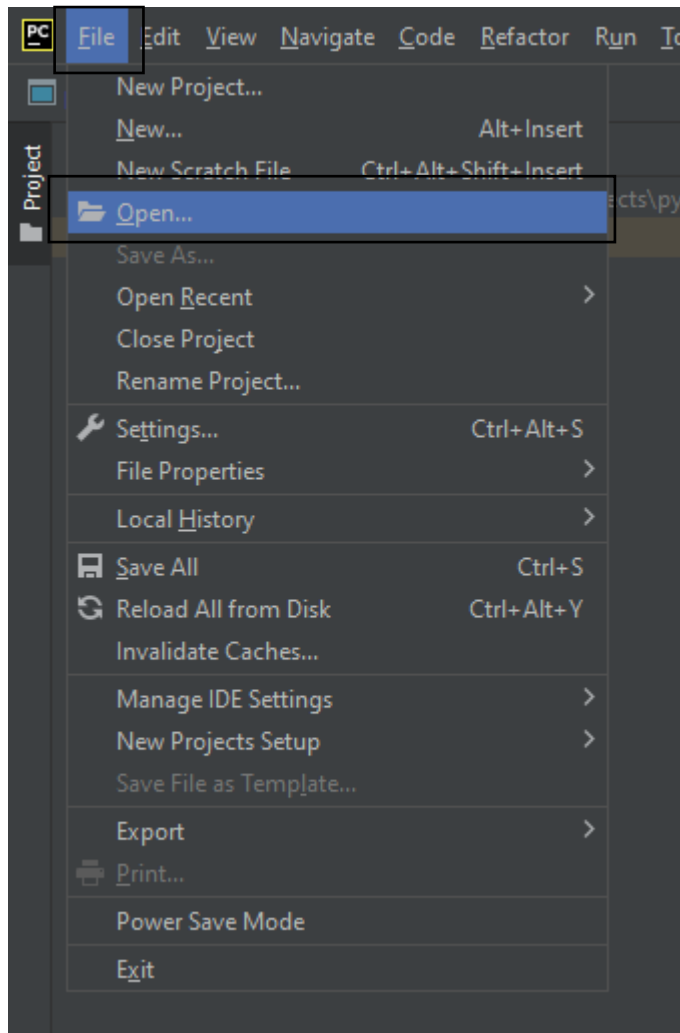
Click on download and download the community version and install it afterwards



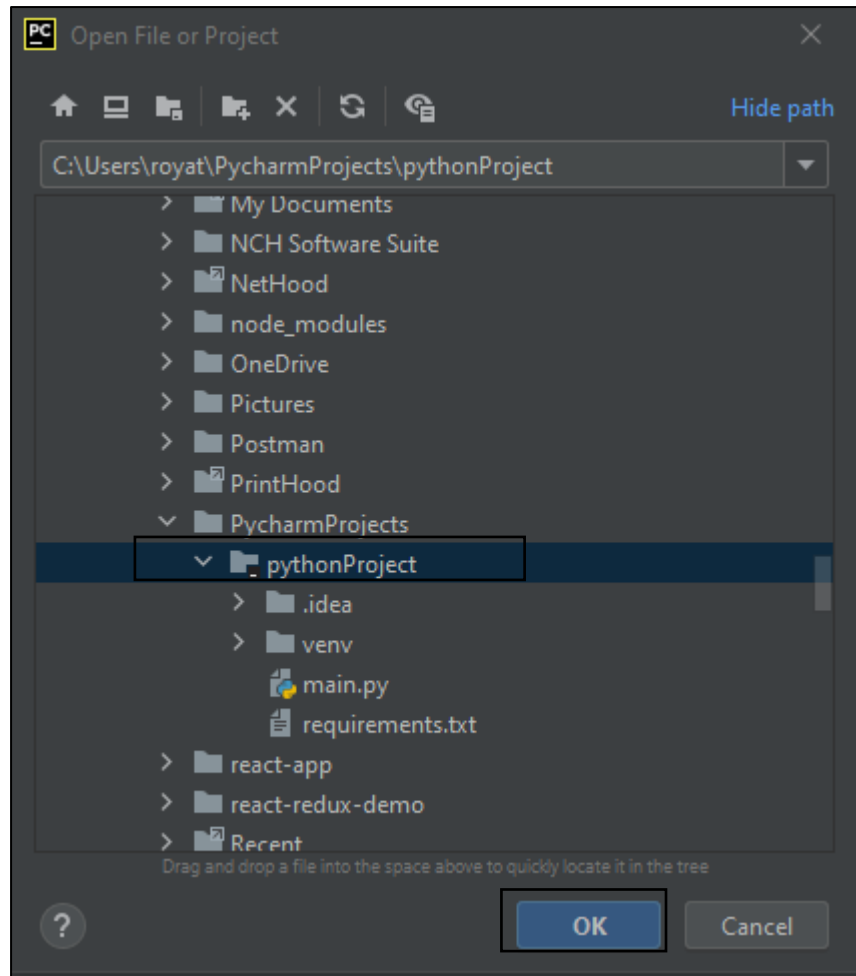
Once installed open **PyCharm** by typing **PyCharm** in the windows search bar and click to open it



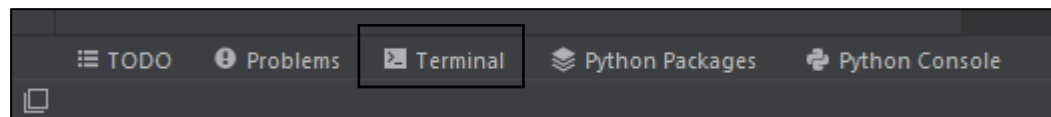
Once it opens click on **file->open**



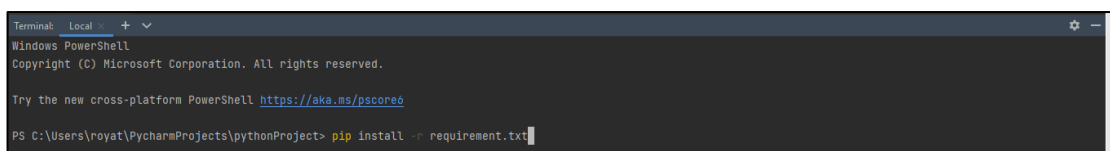
In the window that just opened select the folder pythonProject where it located and press **OK**



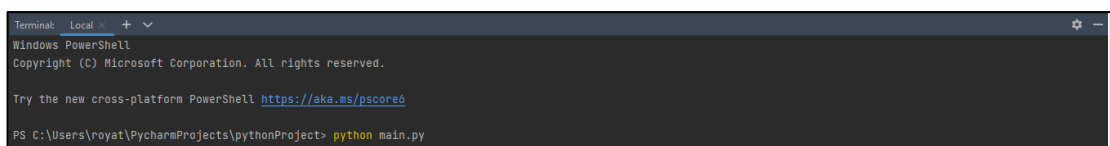
Open the terminal console. It located on the bottom of the window



In the terminal type **pip install -r requirements.txt**. that will install all the necessary packages related to the project.



After all the requirements installed. Type in the terminal **python main.py** to run the script.



2. Code Snippets

2.1 createResourceGroup

```
def createResourceGroup(subscription_id, resource_group_name, location):  
    print('Creating ResourceGroup' + resource_group_name + '...')  
  
    # Set the subscription ID and resource group name  
    subscription_id = subscription_id  
    resource_group_name = resource_group_name  
    location = location  
  
    # Create a ResourceManagementClient instance using AzureCliCredential  
    credential = AzureCliCredential()  
    resource_client = ResourceManagementClient(credential, subscription_id)  
  
    # Create the resource group  
    resource_group_params = ResourceGroup(location=location)  
    resource_client.resource_groups.create_or_update(resource_group_name, resource_group_params)  
  
    print('Finished Creating ResourceGroup' + resource_group_name + '...')
```

This code defines a function named 'createResourceGroup' that takes three parameters:

- subscription_id: The ID of your Azure subscription
- resource_group_name: The name of the resource group to be created
- location: The Azure region where you want to deploy the resource group

The function then uses the Azure SDK to create a new resource group in azure with the given parameters. Here's a summary of what the code does:

1. Print a message indicating that the function is starting to create a new resource group with the specified name.
2. Set the values of the subscription_id, resource_group_name, and location parameters to local variables of the same names.

3. Create an instance of the `AzureCliCredential` class, which represents the credentials needed to authenticate with Azure using the Azure CLI.
4. Create an instance of the `ResourceManagementClient` class, which is used to interact with Azure resources.
5. Create a `ResourceGroup` object with the specified location.
6. Call the `create_or_update` method of the `resource_client.resource_groups` object, passing in the resource group name and the `ResourceGroup` object created in step 5. This method creates a new resource group or updates an existing one with the same name.
7. Print a message indicating that the function has finished creating the new resource group.

2.2 createAccountStorage

```
def createAccountStorage(subscription_id, resource_group_name, account_name, loc):  
  
    print('Creating account storage'+account_name+'...')  
    # Replace with your Azure subscription ID  
    subscription_id = subscription_id  
  
    # Replace with your Azure resource group and storage account names  
    resource_group_name = resource_group_name  
    account_name = account_name  
  
    # Replace with the Azure region where you want to deploy the storage account  
    location = loc  
  
    # Create the Azure Storage account parameters  
    account_params = StorageAccountCreateParameters(  
        sku={"name": "Standard_LRS"}, # Replace with the desired SKU (e.g., Standard_LRS, Standard_GRS, etc.)  
        kind="StorageV2",  
        location=location  
    )  
  
    # Create the Azure Resource Manager and Storage clients  
    credential = AzureCliCredential()  
    resource_client = ResourceManagementClient(credential, subscription_id)  
    storage_client = StorageManagementClient(credential, subscription_id)  
  
    # Create the resource group if it doesn't exist  
    resource_client.resource_groups.create_or_update(resource_group_name, {"location": location})  
  
    # Create the Storage account  
    storage_account = storage_client.storage_accounts.begin_create(resource_group_name, account_name, account_params).result()  
  
    print(f"Storage account created with name '{storage_account.name}' and ID '{storage_account.id}'")
```

This Python function that creates an Azure Storage account using the Azure Resource Manager and Storage clients. It takes in four parameters:

- `subscription_id`: The ID of your Azure subscription
- `resource_group_name`: The name of the resource group in which the storage account will be created
- `account_name`: The name of the storage account to be created
- `loc`: The Azure region where you want to deploy the storage account

The function first creates a `StorageAccountCreateParameters` object, which specifies the parameters for the storage account to be created, including the SKU (e.g., `Standard_LRS`) and kind (e.g., `StorageV2`). It then creates an `AzureCliCredential` object to authenticate with your Azure subscription, and creates instances of the `ResourceManagementClient` and `StorageManagementClient` classes. The function uses these clients to create the resource group (if it doesn't already exist) and the storage account.

Finally, the function prints a message indicating that the storage account has been created, including its name and ID.

2.3 createLinuxVirtualMachine

```
def
createLinuxVm(subscription_id,resource_group_name,vname,subname,ipnam
e,ipconfigname,nicname,vmname,username,password,loc):

    print(
        "Provisioning a virtual machine...some operations might take
a \
minute or two."
    )

    # Acquire a credential object using CLI-based authentication.
    credential = AzureCliCredential()

    # Step 1: Provision a resource group

    # Obtain the management object for resources, using the
credentials
    # from the CLI login.
    resource_client = ResourceManagementClient(credential,
subscription_id)

    # Constants we need in multiple places: the resource group name
and
    # the region in which we provision resources. You can change
these
    # values however you want.
    RESOURCE_GROUP_NAME = resource_group_name
    LOCATION = loc

    # Provision the resource group.
    rg_result = resource_client.resource_groups.create_or_update(
        RESOURCE_GROUP_NAME, {"location": LOCATION}
    )

    print(
```

```

        f"Provisioned resource group {rg_result.name} in the \
        {rg_result.location} region"
    )

    # For details on the previous code, see Example: Provision a
resource
    # group at https://learn.microsoft.com/azure/developer/python/
    # azure-sdk-example-resource-group

    # Step 2: provision a virtual network

    # A virtual machine requires a network interface client (NIC). A
NIC
    # requires a virtual network and subnet along with an IP address.
    # Therefore we must provision these downstream components first,
then
    # provision the NIC, after which we can provision the VM.

    # Network and IP address names
    VNET_NAME = vname
    SUBNET_NAME = subname
    IP_NAME = ipname
    IP_CONFIG_NAME = ipconfigname
    NIC_NAME = nicname

    # Obtain the management object for networks
    network_client = NetworkManagementClient(credential,
subscription_id)

    # Provision the virtual network and wait for completion
    poller = network_client.virtual_networks.begin_create_or_update(
        RESOURCE_GROUP_NAME,
        VNET_NAME,
        {
            "location": LOCATION,
            "address_space": {"address_prefixes": ["10.0.0.0/16"]},
        },
    )

    vnet_result = poller.result()

    print(
        f"Provisioned virtual network {vnet_result.name} with address
\
        prefixes {vnet_result.address_space.address_prefixes}"
    )

    # Step 3: Provision the subnet and wait for completion
    poller = network_client.subnets.begin_create_or_update(
        RESOURCE_GROUP_NAME,
        VNET_NAME,
        SUBNET_NAME,
        {"address_prefix": "10.0.0.0/24"},
    )

    subnet_result = poller.result()

    print(
        f"Provisioned virtual subnet {subnet_result.name} with
address \
        prefix {subnet_result.address_prefix}"
    )

```

```

    # Step 4: Provision an IP address and wait for completion
    poller =
network_client.public_ip_addresses.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    IP_NAME,
    {
        "location": LOCATION,
        "sku": {"name": "Standard"},
        "public_ip_allocation_method": "Static",
        "public_ip_address_version": "IPV4",
    },
)

ip_address_result = poller.result()

print(
    f"Provisioned public IP address {ip_address_result.name} \
with address {ip_address_result.ip_address}"
)

# Step 5: Provision the network interface client
poller =
network_client.network_interfaces.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    NIC_NAME,
    {
        "location": LOCATION,
        "ip_configurations": [
            {
                "name": IP_CONFIG_NAME,
                "subnet": {"id": subnet_result.id},
                "public_ip_address": {"id":
ip_address_result.id},
            }
        ],
    },
)

nic_result = poller.result()

print(f"Provisioned network interface client {nic_result.name}")

# Step 6: Provision the virtual machine

# Obtain the management object for virtual machines
compute_client = ComputeManagementClient(credential,
subscription_id)

VM_NAME = vmname
USERNAME = username
PASSWORD = password

print(
    f"Provisioning virtual machine {VM_NAME}; this operation
might \
take a few minutes."
)

# Provision the VM specifying only minimal arguments, which
defaults

```



```

    # to an Ubuntu 18.04 VM on a Standard_DS1_v2 plan with a public
IP address
    # and a default virtual network/subnet.

    poller = compute_client.virtual_machines.begin_create_or_update(
        RESOURCE_GROUP_NAME,
        VM_NAME,
        {
            "location": LOCATION,
            "storage_profile": {
                "image_reference": {
                    "publisher": "Canonical",
                    "offer": "UbuntuServer",
                    "sku": "16.04.0-LTS",
                    "version": "latest",
                }
            },
            "hardware_profile": {"vm_size": "Standard_DS1_v2"},
            "os_profile": {
                "computer_name": VM_NAME,
                "admin_username": USERNAME,
                "admin_password": PASSWORD,
            },
            "network_profile": {
                "network_interfaces": [
                    {
                        "id": nic_result.id,
                    }
                ]
            },
        },
    )

    vm_result = poller.result()

    print(f"Provisioned virtual machine {vm_result.name}")

```

This Python script provisions a Linux virtual machine in Azure, including the necessary network components such as a virtual network, subnet, and IP address. The script uses the Azure SDK for Python and requires the user to provide some parameters, such as the subscription ID, resource group name, and VM name. The script then uses these parameters to create the necessary resources, using Azure CLI-based authentication.

Here's a brief summary of the steps involved:

1. Acquire a credential object using CLI-based authentication.
2. Provision a resource group.
3. Provision a virtual network.
4. Provision a subnet.
5. Provision an IP address.
6. Provision a network interface client.
7. Provision the virtual machine.

The script uses polling to wait for the completion of each resource provisioning step before moving on to the next one.

2.4 createWindowsVirtualMachine

```
def createWindowsVm(subscription_id, resource_group_name, vname,
subname, ipname, ipconfigname, nicname, vmname, username, password,
loc):

    print(
        "Provisioning a virtual machine...some operations might take
a \
minute or two."
    )

    # Acquire a credential object using CLI-based authentication.
    credential = AzureCliCredential()

    # Step 1: Provision a resource group

    # Obtain the management object for resources, using the
credentials
    # from the CLI login.
    resource_client = ResourceManagementClient(credential,
subscription_id)

    # Constants we need in multiple places: the resource group name
and
    # the region in which we provision resources. You can change
these
    # values however you want.
    RESOURCE_GROUP_NAME = resource_group_name
    LOCATION = loc

    # Provision the resource group.
    rg_result = resource_client.resource_groups.create_or_update(
        RESOURCE_GROUP_NAME, {"location": LOCATION}
    )

    print(
        f"Provisioned resource group {rg_result.name} in the \
{rg_result.location} region"
```

```

    )

    # For details on the previous code, see Example: Provision a
resource
    # group at https://learn.microsoft.com/azure/developer/python/
    # azure-sdk-example-resource-group

    # Step 2: provision a virtual network

    # A virtual machine requires a network interface client (NIC). A
NIC
    # requires a virtual network and subnet along with an IP address.
    # Therefore we must provision these downstream components first,
then
    # provision the NIC, after which we can provision the VM.

    # Network and IP address names
    VNET_NAME = vname
    SUBNET_NAME = subname
    IP_NAME = ipname
    IP_CONFIG_NAME = ipconfigname
    NIC_NAME = nicname

    # Obtain the management object for networks
    network_client = NetworkManagementClient(credential,
subscription_id)

    # Provision the virtual network and wait for completion
    poller = network_client.virtual_networks.begin_create_or_update(
        RESOURCE_GROUP_NAME,
        VNET_NAME,
        {
            "location": LOCATION,
            "address_space": {"address_prefixes": ["10.0.0.0/16"]},
        },
    )

    vnet_result = poller.result()

    print(
        f"Provisioned virtual network {vnet_result.name} with address
\
prefixes {vnet_result.address_space.address_prefixes}"
    )

    # Step 3: Provision the subnet and wait for completion
    poller = network_client.subnets.begin_create_or_update(
        RESOURCE_GROUP_NAME,
        VNET_NAME,
        SUBNET_NAME,
        {"address_prefix": "10.0.0.0/24"},
    )
    subnet_result = poller.result()

    print(
        f"Provisioned virtual subnet {subnet_result.name} with
address \
prefix {subnet_result.address_prefix}"
    )

    # Step 4: Provision an IP address and wait for completion

```

```

    poller =
network_client.public_ip_addresses.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    IP_NAME,
    {
        "location": LOCATION,
        "sku": {"name": "Standard"},
        "public_ip_allocation_method": "Static",
        "public_ip_address_version": "IPv4",
    },
)

ip_address_result = poller.result()

print(
    f"Provisioned public IP address {ip_address_result.name} \
    with address {ip_address_result.ip_address}"
)

# Step 5: Provision the network interface client
poller =
network_client.network_interfaces.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    NIC_NAME,
    {
        "location": LOCATION,
        "ip_configurations": [
            {
                "name": IP_CONFIG_NAME,
                "subnet": {"id": subnet_result.id},
                "public_ip_address": {"id":
ip_address_result.id},
            }
        ],
    },
)

nic_result = poller.result()

print(f"Provisioned network interface client {nic_result.name}")

compute_client = ComputeManagementClient(credential,
subscription_id)

VM_NAME = vmname
USERNAME = username
PASSWORD = password

print(
    f"Provisioning virtual machine {VM_NAME}; this operation
might \
    take a few minutes."
)

# Provision the VM specifying only minimal arguments, which
defaults
# to an Ubuntu 18.04 VM on a Standard DS1 v2 plan with a public
IP address
# and a default virtual network/subnet.

# Provision a Windows Server 2019 VM

```

```

poller = compute_client.virtual_machines.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    VM_NAME,
    {
        "location": LOCATION,
        "storage_profile": {
            "image_reference": {
                "publisher": "MicrosoftWindowsServer",
                "offer": "WindowsServer",
                "sku": "2019-Datacenter",
                "version": "latest",
            }
        },
        "hardware_profile": {"vm_size": "Standard_DS1_v2"},
        "os_profile": {
            "computer_name": VM_NAME,
            "admin_username": USERNAME,
            "admin_password": PASSWORD,
            "windows_configuration": {
                "provision_vmagent": True,
                "enable_automatic_updates": True
            }
        },
        "network_profile": {
            "network_interfaces": [
                {
                    "id": nic_result.id,
                }
            ]
        },
    },
)

vm_result = poller.result()

print(f"Provisioned virtual machine {vm_result.name}")

```

This code creates a Windows virtual machine in Azure. It first provisions a resource group, virtual network, subnet, IP address, and network interface client, before finally provisioning the virtual machine. The virtual machine is created with Windows Server 2019, with the specified hardware profile and OS profile. The network profile of the virtual machine is also specified, along with the network interface client created earlier.

The provisioning of each resource is done using Azure SDK for Python, and the credentials are acquired using Azure CLI-based authentication. The script prints messages to provide information on each provisioning step.

2.5 setNetworkSecurityGroup

```
def setNSG(subscription_id, resource_group_name, nsg_name, loc):  
  
    print("Setting Network Security Group...")  
  
    # Replace the values with your own  
    subscription_id = subscription_id  
    resource_group_name = resource_group_name  
    nsg_name = nsg_name  
  
    # Create the Azure CLI credential  
    credential = AzureCliCredential()  
    network_client = NetworkManagementClient(credential, subscription_id)  
  
    # Create the NSG  
    nsg = NetworkSecurityGroup(location=loc)  
    nsg_result = network_client.network_security_groups.begin_create_or_update(  
        resource_group_name,  
        nsg_name,  
        nsg  
    )  
  
    print("Finished Setting Network Security Group!")
```

This Python code defines a function setNSG that sets up a new Network Security Group (NSG) in Azure using the Azure SDK for Python. Here's what the code does:

1. Imports the necessary libraries for working with Azure resources: AzureCliCredential and NetworkManagementClient from the azure.mgmt.network package.
2. Defines the function setNSG with four input parameters: subscription_id, resource_group_name, nsg_name, and loc.
3. Sets the values of subscription_id, resource_group_name, and nsg_name to the input parameters of the same names.
4. Creates an instance of the AzureCliCredential class to authenticate the Azure CLI.
5. Creates an instance of the NetworkManagementClient class, passing in the credential and subscription_id.
6. Creates a new NetworkSecurityGroup object, passing in the value of loc as the location parameter.
7. Calls the begin_create_or_update method of the network_security_groups client to create or update the NSG in Azure, passing in the values of resource_group_name, nsg_name, and nsg as parameters.
8. Prints a message indicating that the NSG setup is complete.

This function can be used to set up a new Network Security Group in Azure using Python. To use this function, you would need to provide values for the input parameters subscription_id, resource_group_name, nsg_name, and loc.

2.6 AssociateNetworkSecurityGroupWithNIC

```
def associateNsgWithNic(subscription_id, resource_group_name, nsg_name, nic_name):
    print("Associating Network Security Group with server nic...")

    # Replace the values with your own
    subscription_id = subscription_id
    resource_group_name = resource_group_name
    nsg_name = nsg_name
    nic_name = nic_name

    # Create the Azure CLI credential
    credential = AzureCliCredential()
    network_client = NetworkManagementClient(credential, subscription_id)

    # Get the NSG
    nsg = network_client.network_security_groups.get(resource_group_name, nsg_name)

    # Get the NIC
    nic = network_client.network_interfaces.get(resource_group_name, nic_name)

    # Associate the NSG with the NIC
    nic.network_security_group = nsg
    nic_result = network_client.network_interfaces.begin_create_or_update(resource_group_name, nic_name, nic)

    print("Finished Associating Network Security Group with server nic!")
```

This function associates a Network Security Group (NSG) with a network interface card (NIC) in an Azure environment. The NSG provides security rules to control network traffic to and from the NIC.

Here is a breakdown of what the function does:

The function takes in four parameters:

- `subscription_id`: The ID of the Azure subscription that contains the resources.
- `resource_group_name`: The name of the resource group that contains the resources.
- `nsg_name`: The name of the NSG to associate with the NIC.
- `nic_name`: The name of the NIC to associate with the NSG.

The function creates an Azure CLI credential and a `NetworkManagementClient` object.

The function retrieves the NSG from the resource group.

The function retrieves the NIC from the resource group.

The function sets the `network_security_group` property of the NIC to the retrieved NSG.

The function updates the NIC with the new NSG association.

The function prints a message indicating that the association is complete.

Overall, this function is useful for managing network security in an Azure environment by associating NSGs with NICs.

2.7 SetPort3389ForRemoteConnection

```
def setPort3389(subscription_id, resource_group_name, nsg_name):  
  
    print("Setting Port 3389 for remote connection...")  
  
    # Set subscription ID, resource group name, and NSG name  
    subscription_id = subscription_id  
    resource_group_name = resource_group_name  
    nsg_name = nsg_name  
  
    # Create a NetworkManagementClient instance using  
    # AzureCliCredential  
    credential = AzureCliCredential()  
    network_client = NetworkManagementClient(credential,  
subscription_id)  
  
    # Get the NSG  
    nsg =
```



```

network_client.network_security_groups.get(resource_group_name,
nsg_name)

    # Get the existing NSG object
    nsg =
network_client.network_security_groups.get(resource_group_name,
nsg_name)

    # Create the new security rule object
    new_rule = SecurityRule(
        name="RDP",
        description="Allow RDP traffic on port 3389",
        protocol=SecurityRuleProtocol.tcp,
        source_address_prefix="*",
        source_port_range="*",
        destination_address_prefix="*",
        destination_port_range="3389",
        access=SecurityRuleAccess.allow,
        direction=SecurityRuleDirection.inbound,
        priority=1010 # Make sure this priority value is higher than
any existing rules
    )

    # Add the new security rule to the NSG
    nsg.security_rules.append(new_rule)

    # Update the NSG on Azure

network_client.network_security_groups.begin_create_or_update(resource_group_name, nsg_name, nsg)

    print("Finished Setting Port 3389 for remote connection!")

```

This function sets up an inbound security rule to allow remote desktop connection on port 3389 in a network security group (NSG). The function performs the following actions:

1. Sets the Azure subscription ID, resource group name, and NSG name
2. Uses the AzureCliCredential to create a NetworkManagementClient instance
3. Retrieves the existing NSG object using the get() method
4. Creates a new security rule object to allow RDP traffic on port 3389
5. Appends the new security rule to the existing NSG object
6. Calls the begin_create_or_update() method to update the NSG on Azure.

This function assumes that the NSG already exists and that the Azure CLI is already authenticated.

2.8. getStorageAccountNames

```
def getStorageAccountNames(subscription_id, resource_group_name):
    storage_accounts_names = []
    # Create an AzureCliCredential object to authenticate with Azure
    credential = AzureCliCredential()

    # Create a StorageManagementClient object using the AzureCliCredential object
    storage_mgmt_client = StorageManagementClient(credential, subscription_id)

    # Get the list of storage accounts associated with the current Azure credentials
    storage_accounts = storage_mgmt_client.storage_accounts.list_by_resource_group(resource_group_name)

    # Iterate over the storage accounts and print their names
    for account in storage_accounts:
        storage_accounts_names.append(account.name)

    return storage_accounts_names
```

This function takes two parameters as input: `subscription_id` and `resource_group_name`. It uses the `AzureCliCredential` object to authenticate with Azure and create a `StorageManagementClient` object to interact with Azure Storage services.

The function then retrieves a list of storage accounts associated with the specified `resource_group_name` using the `list_by_resource_group` method of the `storage_accounts` object. It iterates over the storage accounts and adds their names to a list called `storage_accounts_names`.

Finally, the function returns the `storage_accounts_names` list containing the names of all the storage accounts associated with the specified `resource_group_name`.

2.9 renewConnectionStrings

```
def renewConnectionStrings(subscription_id, resource_group_name, storage_account_name):
    # Create the Azure Resource Manager and Storage clients
    credential = AzureCliCredential()

    storage_client = StorageManagementClient(credential, subscription_id)
    keys = storage_client.storage_accounts.list_keys(resource_group_name, storage_account_name)
    sas_token = generate_account_sas(
        account_name=storage_account_name,
        account_key=keys.keys[0].value,
        resource_types=ResourceTypes(service=True, container=True, object=True),
        permission=AccountSasPermissions(read=True, write=True, delete=True, list=True, add=True, create=True,
                                         update=True, process=True, immutablestorage=True, permanentdelete=True),
        expiry=datetime.utcnow() + timedelta(hours=1)
    )

    connection_string = f'BlobEndpoint=https://{storage_account_name}.blob.core.windows.net;QueueEndpoint=https://{storage_account'
    return connection_string
```

This function takes three parameters as input: `subscription_id`, `resource_group_name`, and `storage_account_name`. It uses the `AzureCliCredential`

object to authenticate with Azure and create a `StorageManagementClient` object to interact with Azure Storage services.

The function retrieves the account keys of the specified storage account using the `list_keys` method of the `storage_accounts` object. It then generates a shared access signature (SAS) token for the storage account using the `generate_account_sas` method.

The SAS token is generated with read, write, delete, list, add, create, update, process, immutablestorage, and permanentdelete permissions, and an expiration time set to one hour from the current time.

Finally, the function constructs a connection string for the storage account that includes the SAS token, and returns it.

2.10 CopyBlobsFromStorageAccountAToB

```
def copy_blobs(source_connection_string, dest_connection_string,
src_container_name, dst_container_name, number_of_blobs):
    print('Start creating, uploading, and copying ' +
str(number_of_blobs) + ' blobs...')

    # Enter your source and destination storage account connection
strings
    source_connection_string = source_connection_string
    destination_connection_string = dest_connection_string

    # Create a BlobServiceClient object for each storage account
    source_blob_service_client =
BlobServiceClient.from_connection_string(source_connection_string)
    destination_blob_service_client =
BlobServiceClient.from_connection_string(destination_connection_string)

    # Create a container in the source storage account
    source_container_name = src_container_name

source_blob_service_client.create_container(source_container_name)

    # Upload 100 blobs to the source container
    for i in range(1, number_of_blobs+1):
        source_blob_name = "blob" + str(i)
        source_blob_data = b"Hello, World!"
        source_blob_client =
source_blob_service_client.get_blob_client(container=source_container
_name,
blob=source_blob_name)
        source_blob_client.upload_blob(source_blob_data)

    # Create a container in the destination storage account
    destination_container_name = dst_container_name
```

```

destination_blob_service_client.create_container(destination_container_name)

    # Copy the 100 blobs from the source container to the destination container
    for i in range(1, 101):
        source_blob_name = "blob" + str(i)
        destination_blob_name = "copy_blob" + str(i)
        source_blob_url =
source_blob_service_client.get_blob_client(container=source_container_name,
blob=source_blob_name).url
        destination_blob_client =
destination_blob_service_client.get_blob_client(container=destination_container_name,
blob=destination_blob_name)
        destination_blob_client.start_copy_from_url(source_blob_url)

    print('Finished copying ' + str(number_of_blobs) + ' blobs!')

```

This function copies blobs from a source storage account to a destination storage account. It uses the Azure Blob Storage Python SDK to create BlobServiceClient objects for each storage account and then creates containers in both accounts. It then uploads a specified number of blobs to the source container and copies them to the destination container. The function takes in the source and destination storage account connection strings, the names of the source and destination containers, and the number of blobs to create, upload, and copy

3. Usage

To use this functions in a python script you need to call them with the desirable parameters.

```

def createResourceGroup(subscription_id,resource_group_name,location)
def createAccountStorage(subscription_id, resource_group_name, account_name, loc)
def createLinuxVm(subscription_id, resource_group_name, vname, subname, ipname, ipconfigname, nicname, vmname,username, password, loc)
def createWindowsVm(subscription_id, resource_group_name, vname, subname, ipname, ipconfigname, nicname, vmname,username, password, loc)
def setNSG(subscription_id, resource_group_name, nsg_name, loc)
def associateNsgWithNic(subscription_id, resource_group_name, nsg_name, nic_name)
def setPort3389(subscription_id, resource_group_name, nsg_name)
def copy_blobs(source_connection_string, dest_connection_string, src_container_name, dst_container_name,number_of_blobs)

```

4. Parameters Obtaining

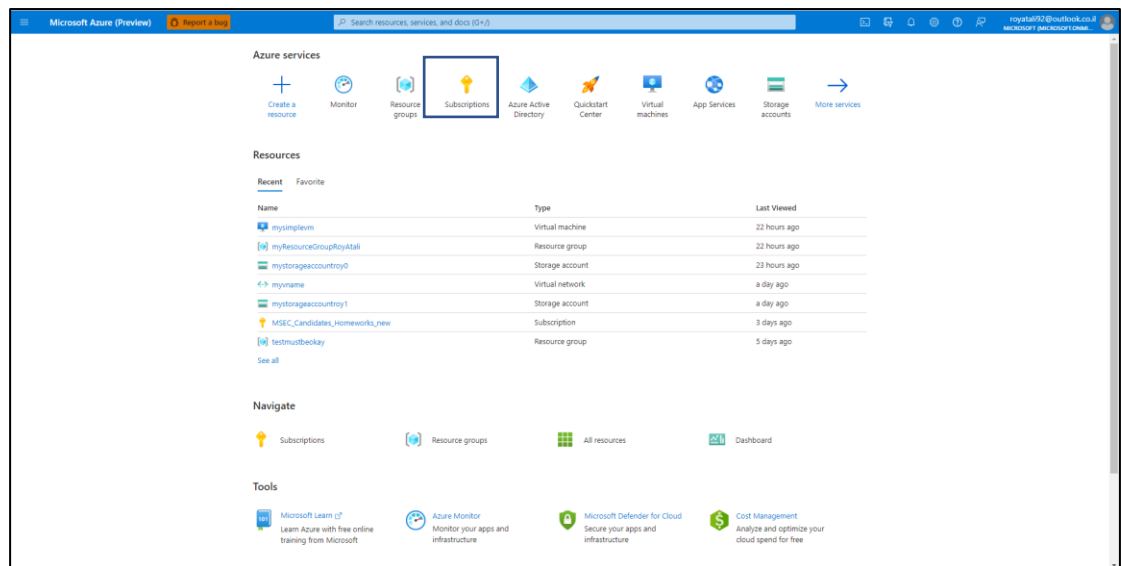
There is a couple of parameters you need to provide to the functions above.

Here is how to obtain them:

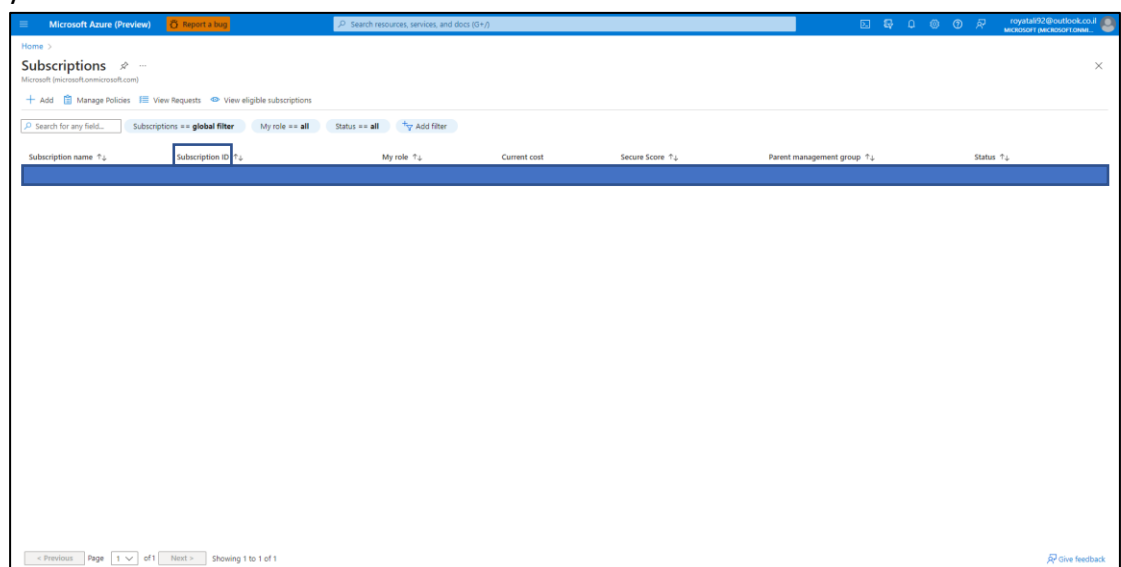
4.1 Subscription ID

1. Open this [link](#) to Azure portal

In the window that opened click on **Subscriptions**



2. After you clicked in the next window you should see the subscription ID of your account.



5. Automating the task copy_blobs

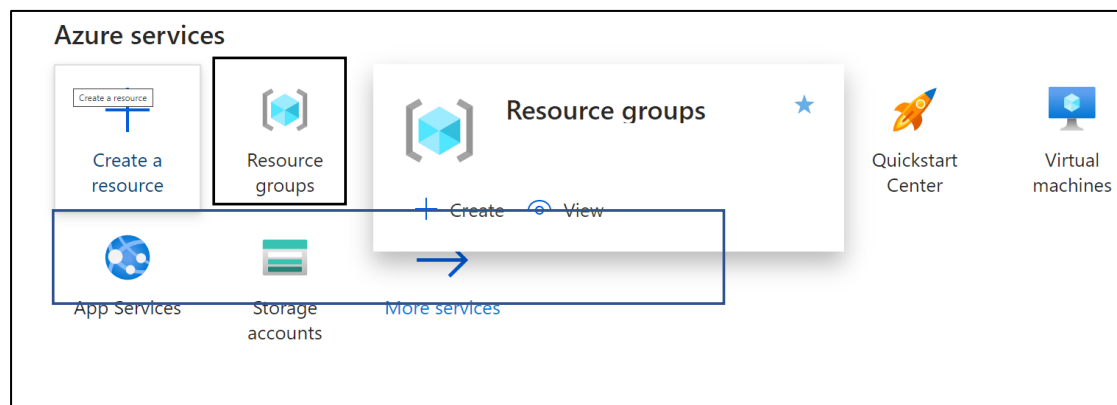
As part of the assignment the function copy_blobs that creates, uploads and copy 100 blobs from one storage account to another (source to destination) needs to run on the virtual machine (server) created previously and it needs to be automated.

I chose to automate it using **Windows scheduler**.

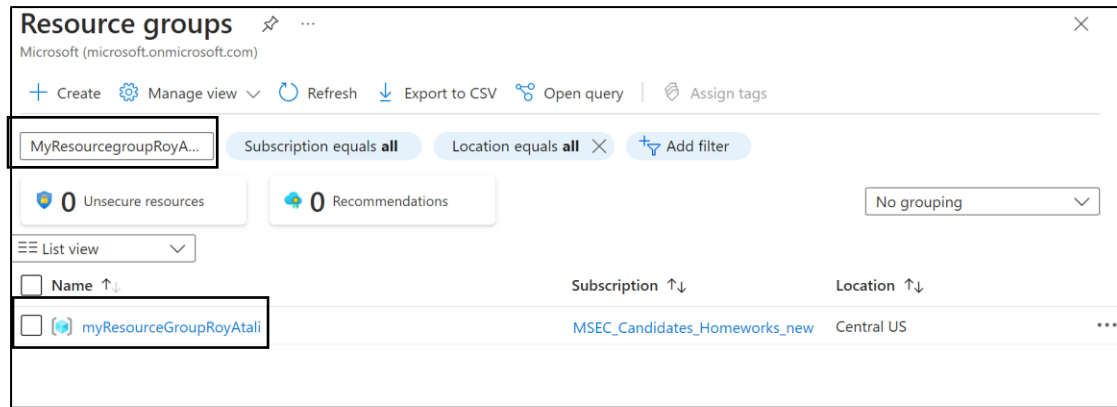
Here is how to do it:

1.Go to this [link](#). It will open the **Azure portal**

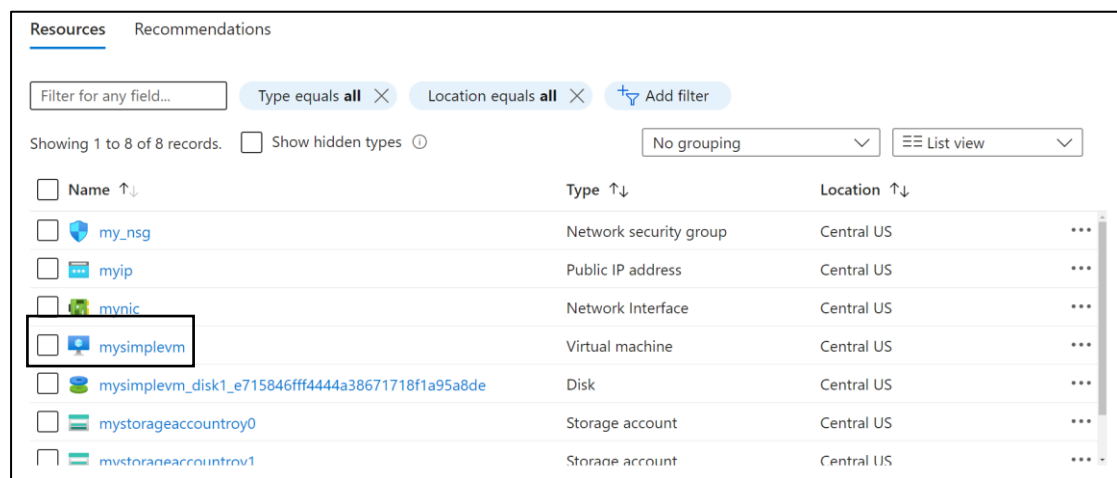
2.click on **Resource Groups**



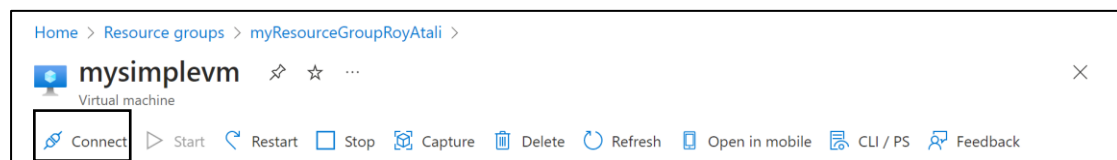
3.search your resource group and click on it



In the resources click on your virtual machine

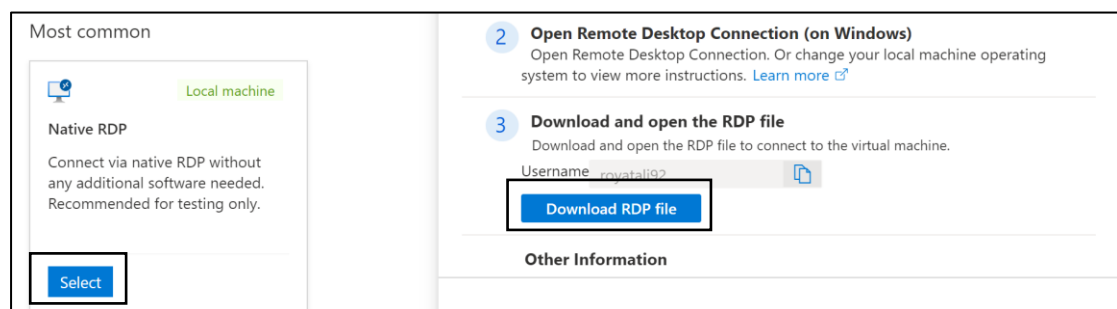


Click on **Connect**



Click on **Select**

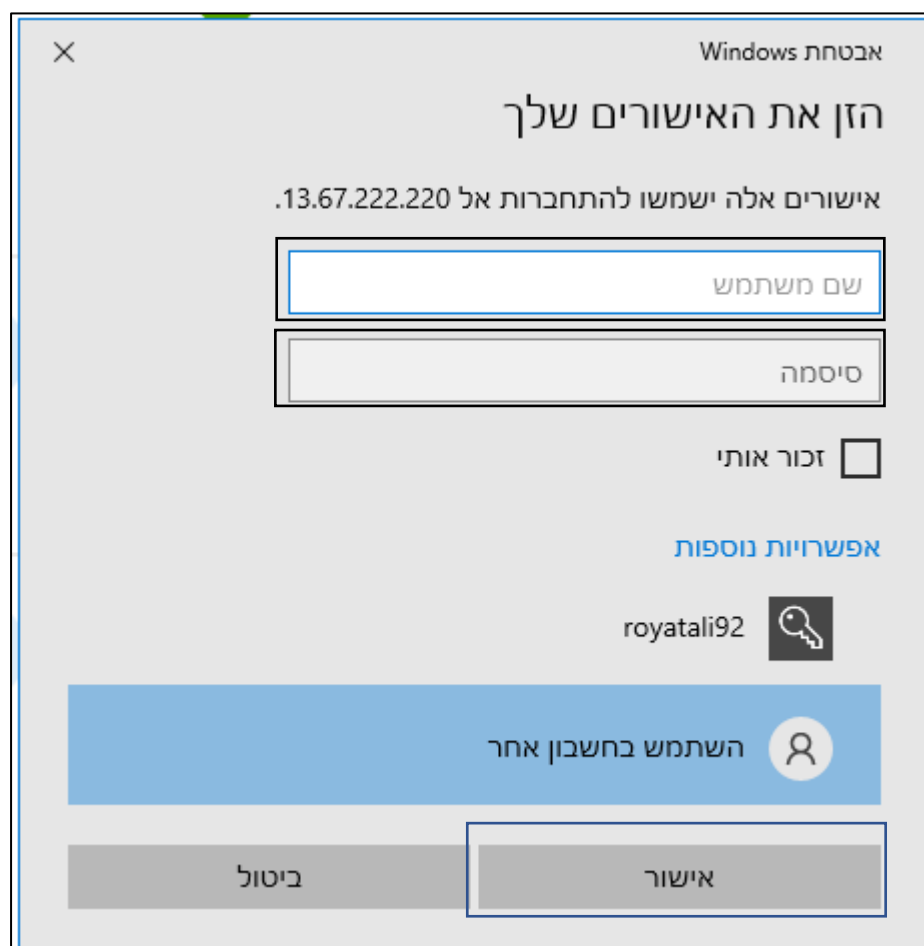
And **Download RDP file**. This will download a remote desktop connection for connecting remotely to your virtual machine



Click on **התחבר/connect**

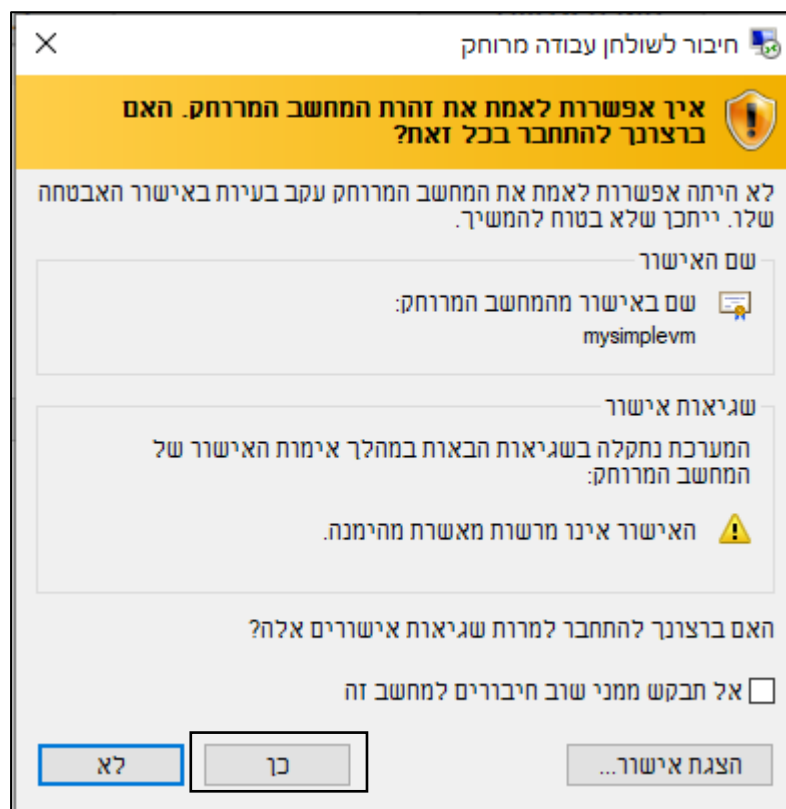


Enter your credentials (username and password you provide previously when creating the virtual machine)



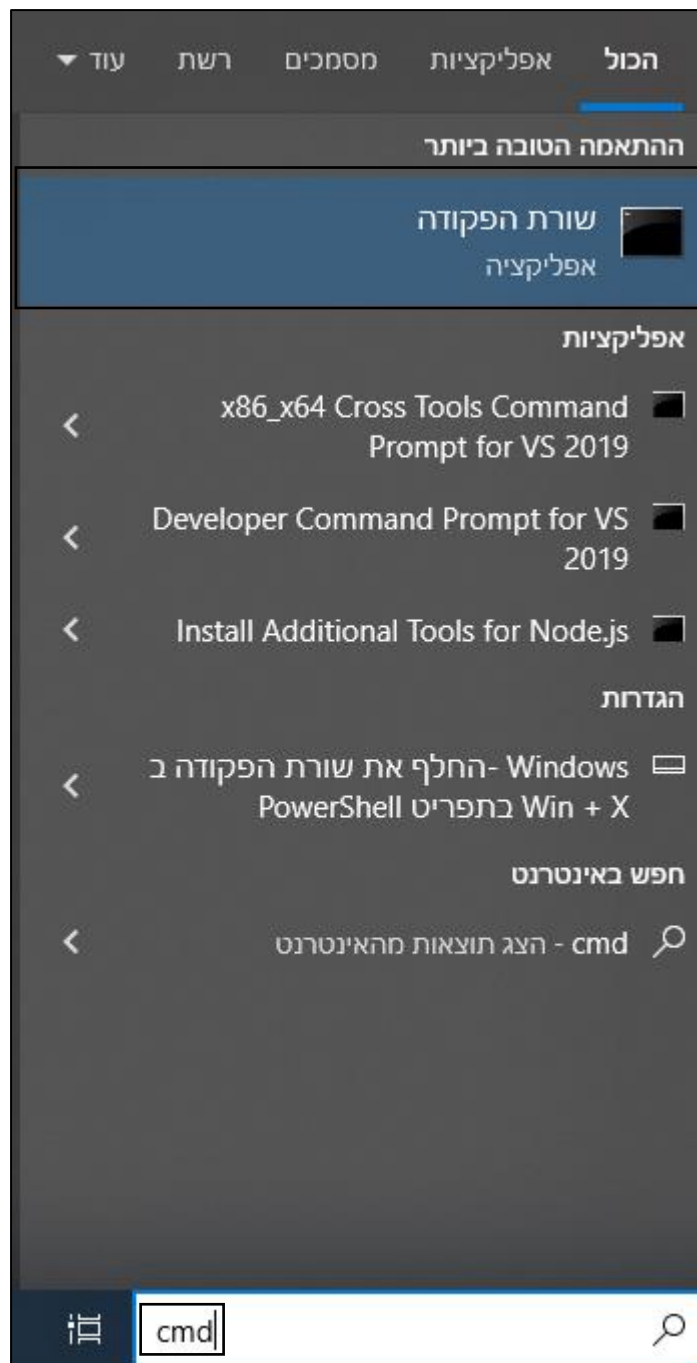
And click **אישור / OK**

click **כן / Yes**



After that you will need to install Python and PyCharm on the virtual machine.
You can do that by looking in the **Prerequisites/Requirements** section above.

After installed go to the search bar on the Virtual Machine and type **cmd**.



In the command prompt type the command:

cd AppData\Local\Programs\Python\Python311\Scripts

and press enter.

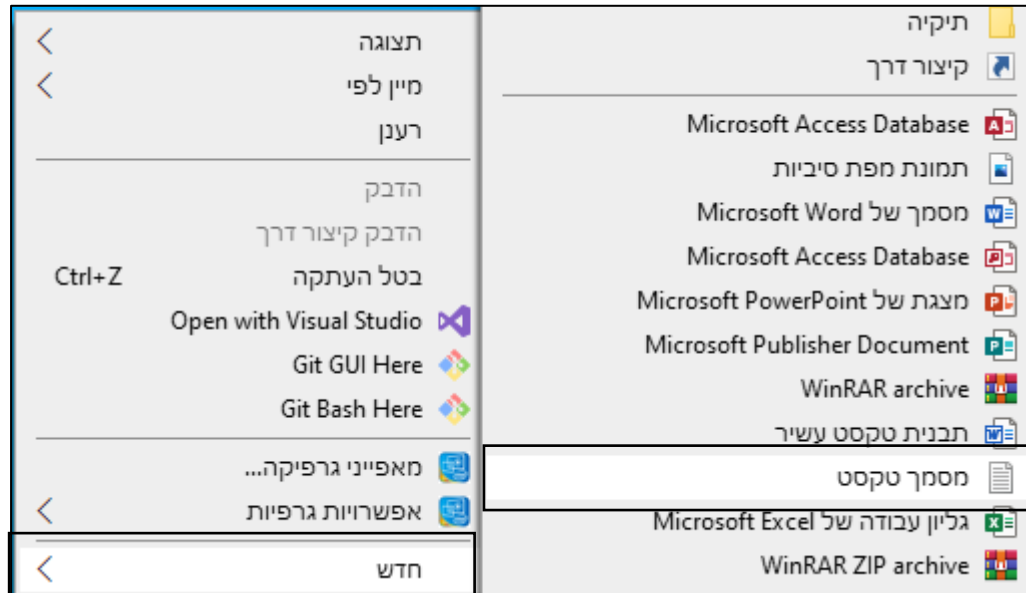
```
שורת הפקודה
Microsoft Windows [Version 10.0.19044.2486]
(c) Microsoft Corporation. כל הזכויות שמורות.

C:\Users\royat>cd AppData\Local\Programs\Python\Python311\Scripts
C:\Users\royat\AppData\Local\Programs\Python\Python311\Scripts>
```

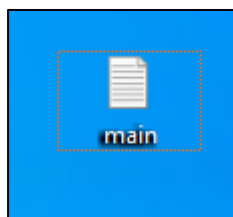
Then type these commands one by one and then press Enter after each one:

1. pip install azure-identity
2. pip install azure-storage-blob
3. pip install azure-mgmt-storage

After the installation is finished open notepad by clicking right click on the mouse then select חדש/new->טקסט מסמך/Text document



Rename the file to whatever name you like



Copy and paste this code to it:

```
from datetime import datetime, timedelta

from azure.identity import AzureCliCredential

from azure.mgmt.storage import StorageManagementClient

from azure.storage.blob import BlobServiceClient, generate_account_sas,
ResourceTypes, AccountSasPermissions


def getStorageAccountNames(subscription_id, resource_group_name):

    storage_accounts_names = []

    # Create an AzureCliCredential object to authenticate with Azure
    credential = AzureCliCredential()

    # Create a StorageManagementClient object using the AzureCliCredential object
    storage_mgmt_client = StorageManagementClient(credential, subscription_id)

    # Get the list of storage accounts associated with the current Azure credentials
    storage_accounts =
storage_mgmt_client.storage_accounts.list_by_resource_group(resource_group_name)

    # Iterate over the storage accounts and print their names
    for account in storage_accounts:
        storage_accounts_names.append(account.name)

    return storage_accounts_names
```

```

def
renewConnectionStrings(subscription_id,resource_group_name,storage_account_name):

    # Create the Azure Resource Manager and Storage clients

    credential = AzureCliCredential()

    storage_client = StorageManagementClient(credential, subscription_id)

    keys = storage_client.storage_accounts.list_keys(resource_group_name,
storage_account_name)

    sas_token = generate_account_sas(
        account_name=storage_account_name,
        account_key=keys.keys[0].value,
        resource_types=ResourceTypes(service=True, container=True, object=True),
        permission=AccountSasPermissions(read=True, write=True, delete=True,
list=True, add=True, create=True,
                                update=True, process=True, immutablestorage=True,
permanentdelete=True),
        expiry=datetime.utcnow() + timedelta(hours=1)
    )

    connection_string =
f'BlobEndpoint=https://{storage_account_name}.blob.core.windows.net/;QueueEnd
point=https://{storage_account_name}.queue.core.windows.net/;FileEndpoint=http
s://{storage_account_name}.file.core.windows.net/;TableEndpoint=https://{storage
_account_name}.table.core.windows.net/;SharedAccessSignature={sas_token}'

    return connection_string


def copy_blobs(source_connection_string, dest_connection_string,
src_container_name, dst_container_name, number_of_blobs):

    print('Start creating, uploading, and copying ' + str(number_of_blobs) + ' blobs...')

```

```

# Enter your source and destination storage account connection strings
source_connection_string = source_connection_string
destination_connection_string = dest_connection_string

# Create a BlobServiceClient object for each storage account
source_blob_service_client =
BlobServiceClient.from_connection_string(source_connection_string)
destination_blob_service_client =
BlobServiceClient.from_connection_string(destination_connection_string)

# Create a container in the source storage account
source_container_name = src_container_name
source_blob_service_client.create_container(source_container_name)

# Upload 100 blobs to the source container
for i in range(1, number_of_blobs+1):
    source_blob_name = "blob" + str(i)
    source_blob_data = b"Hello, World!"
    source_blob_client =
source_blob_service_client.get_blob_client(container=source_container_name,
                                           blob=source_blob_name)
    source_blob_client.upload_blob(source_blob_data)

# Create a container in the destination storage account
destination_container_name = dst_container_name
destination_blob_service_client.create_container(destination_container_name)

# Copy the 100 blobs from the source container to the destination container
for i in range(1, number_of_blobs+1):

```

```

source_blob_name = "blob" + str(i)
destination_blob_name = "copy_blob" + str(i)

source_blob_url =
source_blob_service_client.get_blob_client(container=source_container_name,
                                           blob=source_blob_name).url

destination_blob_client =
destination_blob_service_client.get_blob_client(container=destination_container_n
ame,
                                                blob=destination_blob_name)

destination_blob_client.start_copy_from_url(source_blob_url)


print('Finished copying ' + str(number_of_blobs) + ' blobs!')

```

#function calls

```

#source_storage_account,dest_storage_account =
getStorageAccountNames(subscription_id,resource_group_name)

#source_connection_string = renewConnectionStrings(subscription_id,
resource_group_name, source_storage_account)

#dest_connetion_string = renewConnectionStrings(subscription_id,
resource_group_name, dest_storage_account)

#copy_blobs(source_connection_string, dest_connection_string,
src_container_name, dst_container_name,number_of_blobs)

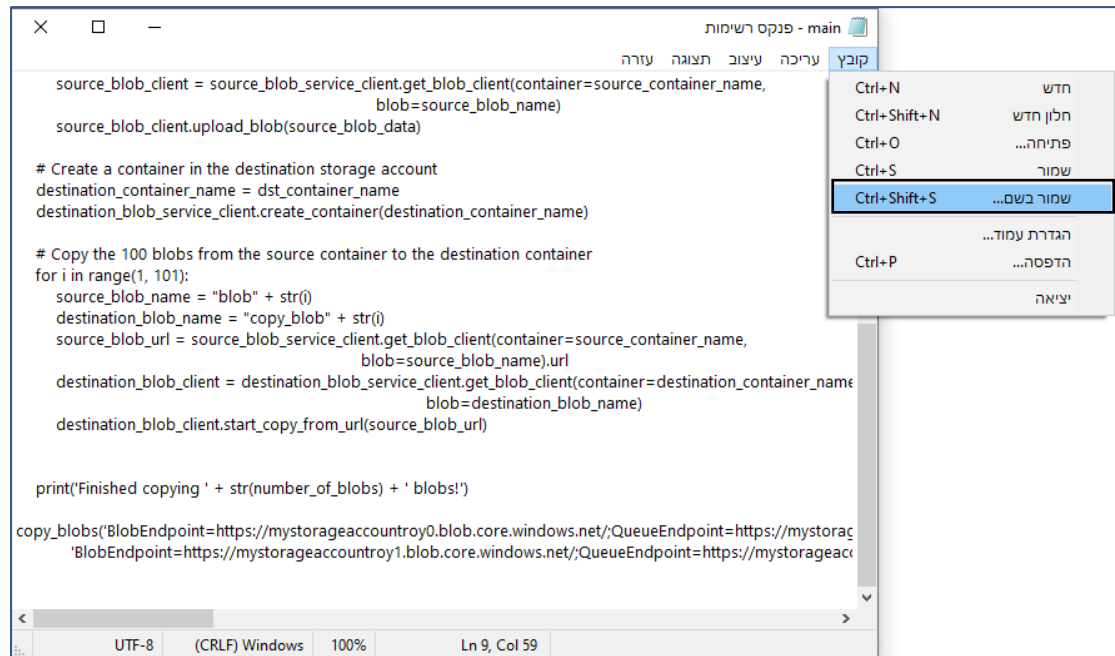
```

***remember to change these parameters in the functions calls according to yours:**

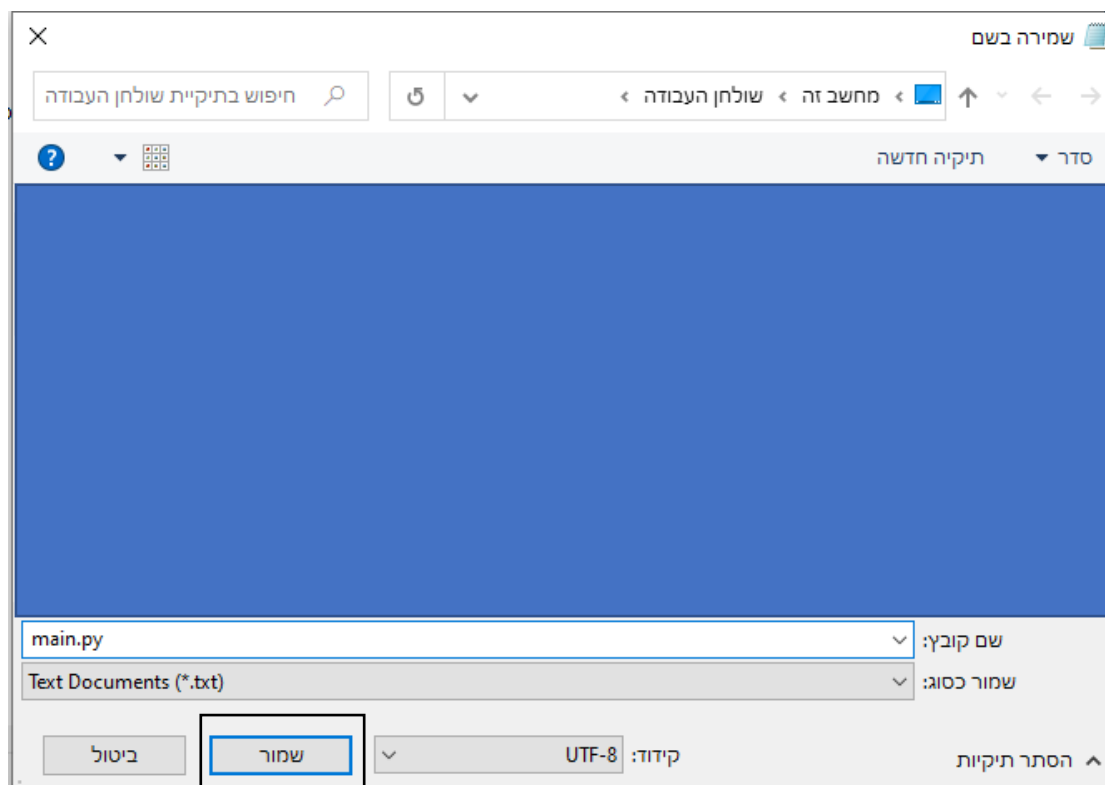
1. subscription_id
2. resource_group_name
3. src_container_name
4. dst_container_name

5. number_of_blobs

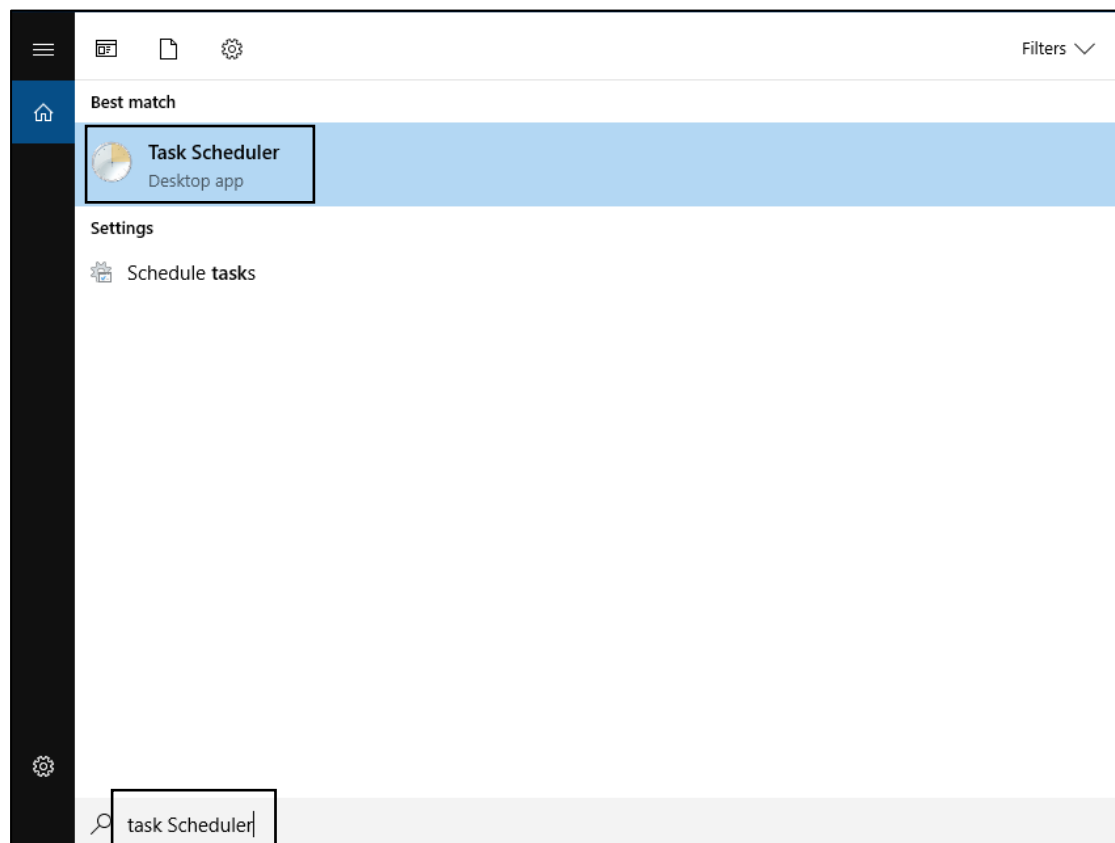
When everything is done click on **שומר בשם->file/קובץ**



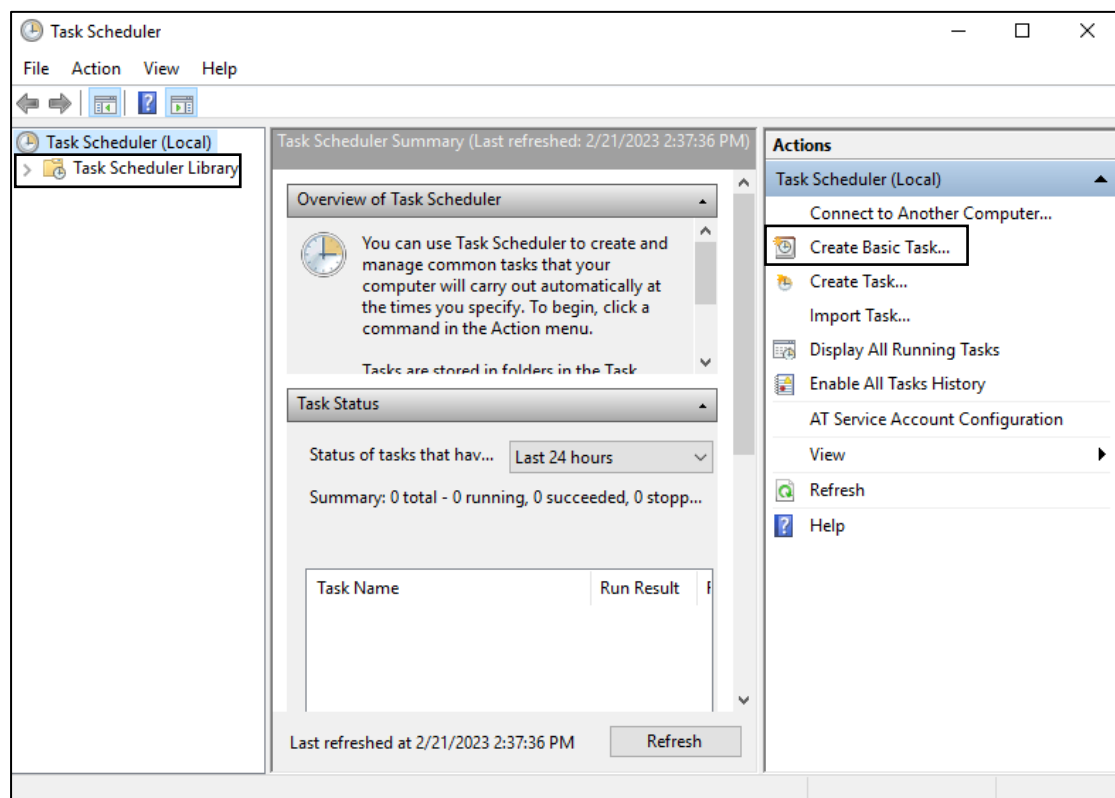
Save the file with **.py** extension for example **main.py**



Open the windows search bar and type task Scheduler and click on it



In the window that opened choose Task Scheduler Library and then Create Task



In General Tab Give the task a name and description like in the picture below

The 'Create Task' dialog box is shown with the 'General' tab selected. The 'Name' field contains 'copy blobs'. The 'Location' field is empty. The 'Author' field contains 'mysimplevm\royatali92'. The 'Description' field contains 'copy 100 blobs between one storage account to another'. The 'Security options' section shows 'When running the task, use the following user account:' with 'mysimplevm\royatali92' selected and a 'Change User or Group...' button. Below this, the 'Run only when user is logged on' radio button is selected. There are also checkboxes for 'Do not store password' and 'Run with highest privileges', both of which are unchecked. At the bottom, there is a 'Hidden' checkbox (unchecked) and a 'Configure for:' dropdown menu set to 'Windows Vista™, Windows Server™ 2008'. 'OK' and 'Cancel' buttons are at the bottom right.

Move to Triggers Tab and select New...

The 'Create Task' dialog box is shown with the 'Triggers' tab selected. The 'General' tab is also visible. The text 'When you create a task, you can specify the conditions that will trigger the task.' is displayed. Below this is a table with three columns: 'Trigger', 'Details', and 'Status'. The table is currently empty. At the bottom left of the table area are three buttons: 'New...', 'Edit...', and 'Delete'. The 'New...' button is highlighted with a blue border. 'OK' and 'Cancel' buttons are at the bottom right.

In the New Trigger Window choose the appropriate start date/time to run the script and click OK

New Trigger

Begin the task: On a schedule

Settings

☒ One time
☐ Daily
☐ Weekly
☐ Monthly

Start: 2/21/2023 2:43:06 PM ☐ Synchronize across time zones

Advanced settings

☐ Delay task for up to (random delay): 1 hour

☐ Repeat task every: 1 hour for a duration of: 1 day

☐ Stop all running tasks at end of repetition duration

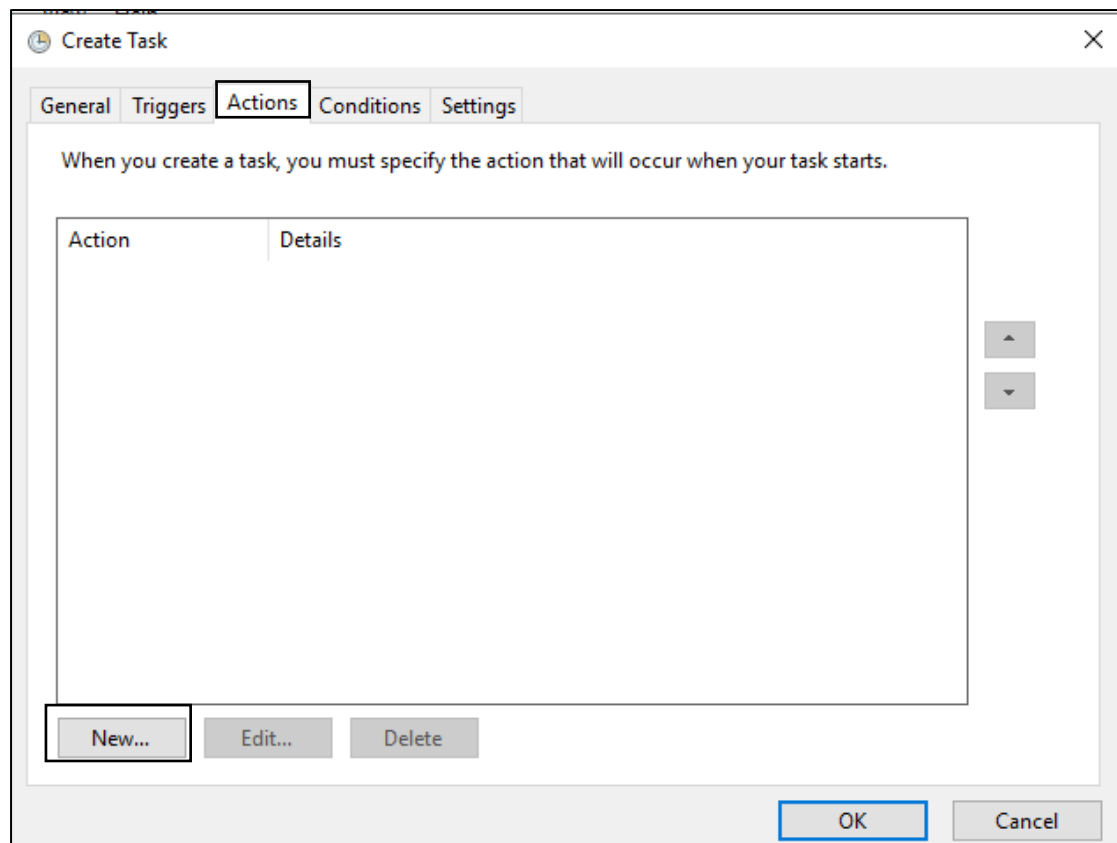
☐ Stop task if it runs longer than: 3 days

☐ Expire: 2/21/2024 2:43:07 PM ☐ Synchronize across time zones

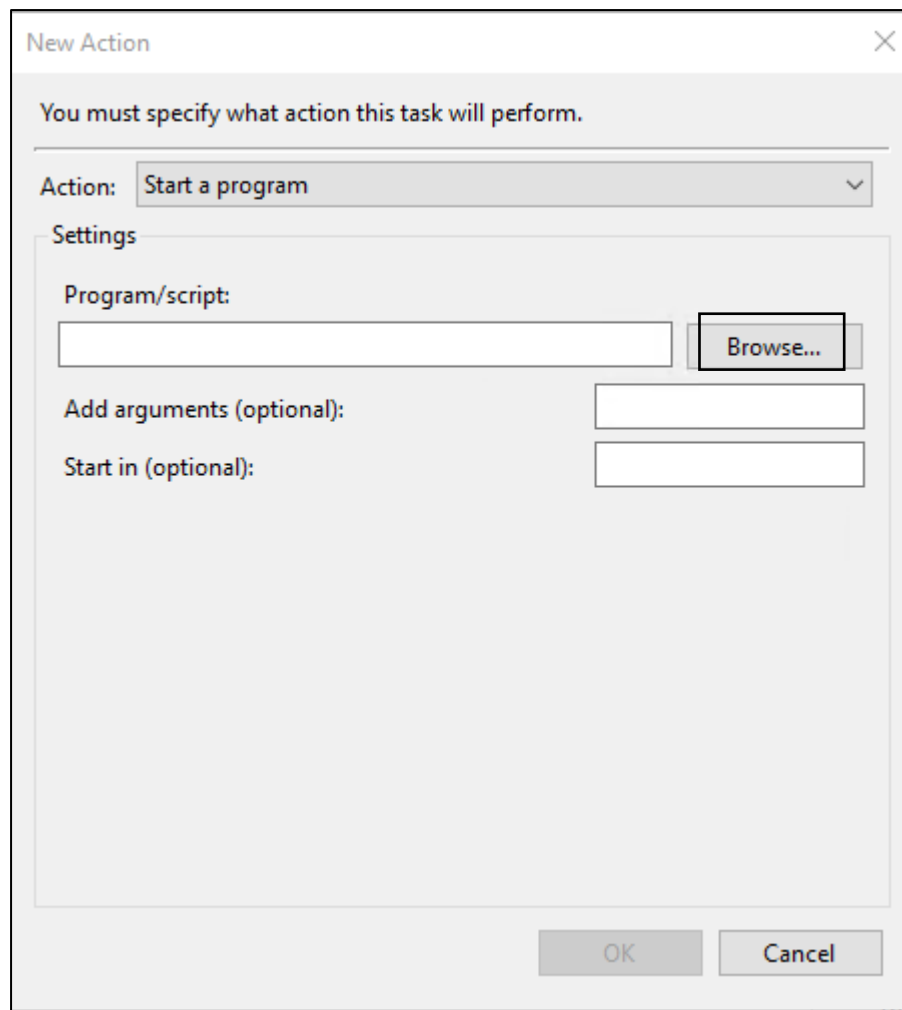
☒ Enabled

OK Cancel

Move to Actions Tab and click New...



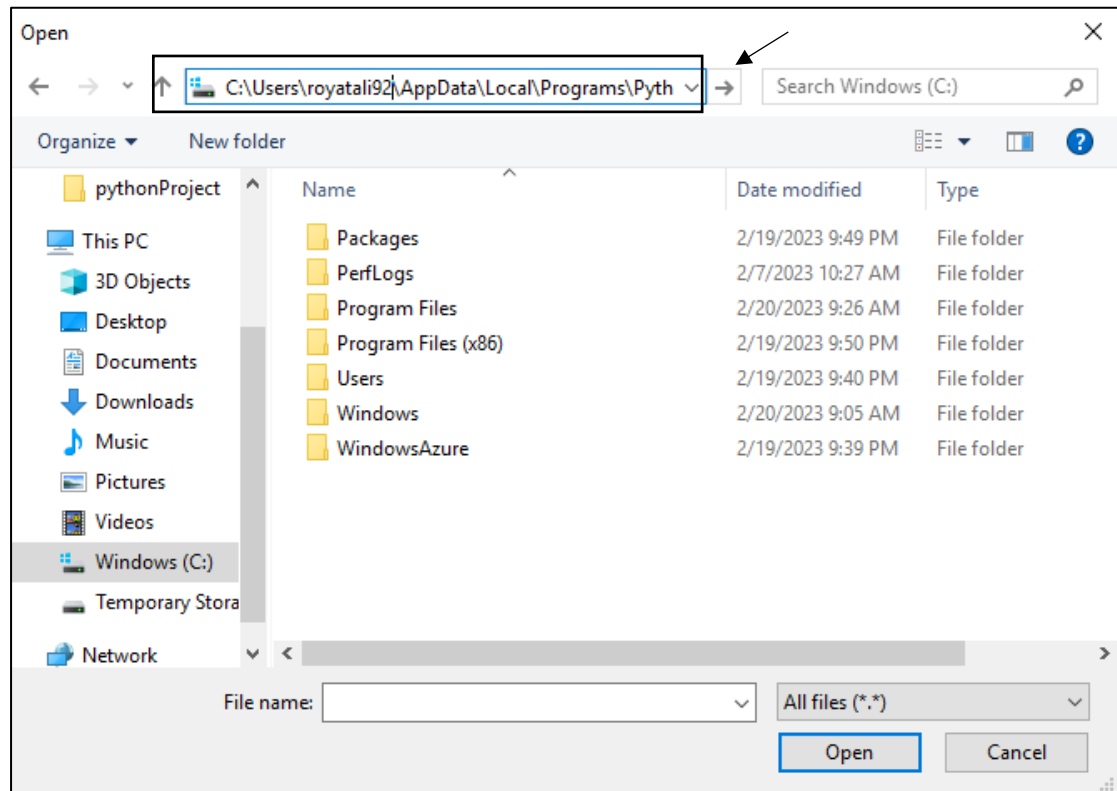
In the New Action Window select Browse



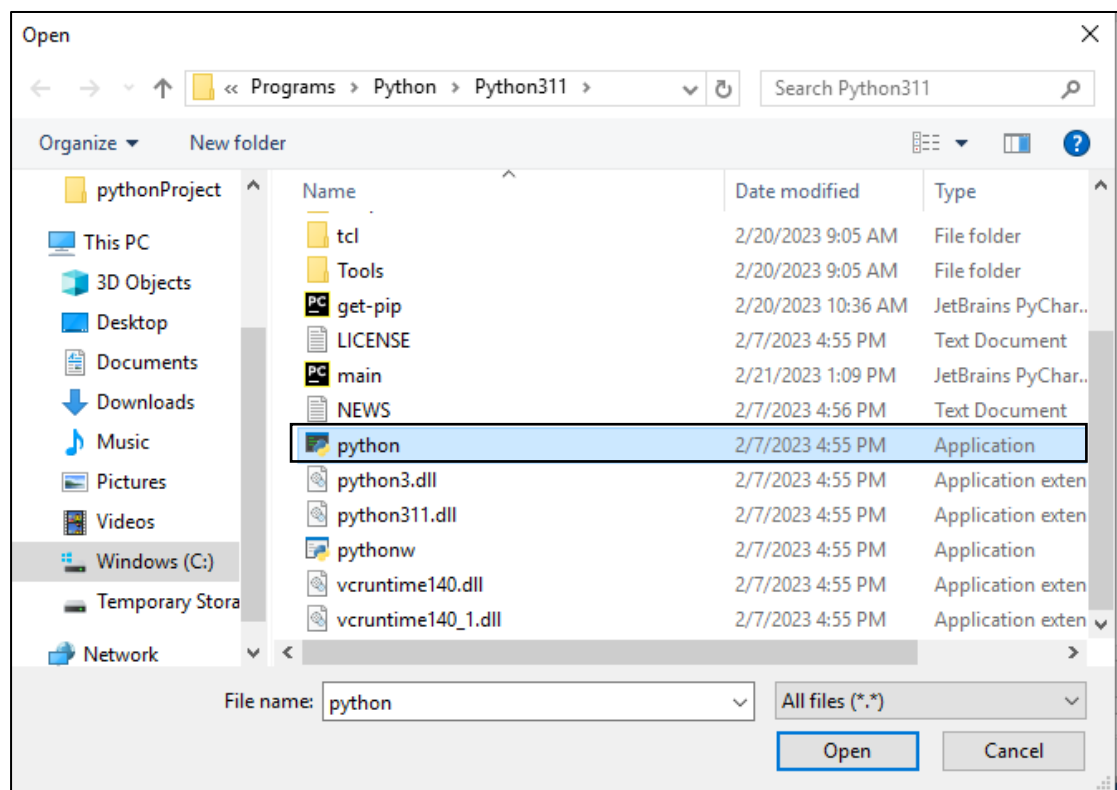
In the browser enter this path:

C:\Users\yourUserName\AppData\Local\Programs\Python\Python311\

And press the arrow button in the picture below

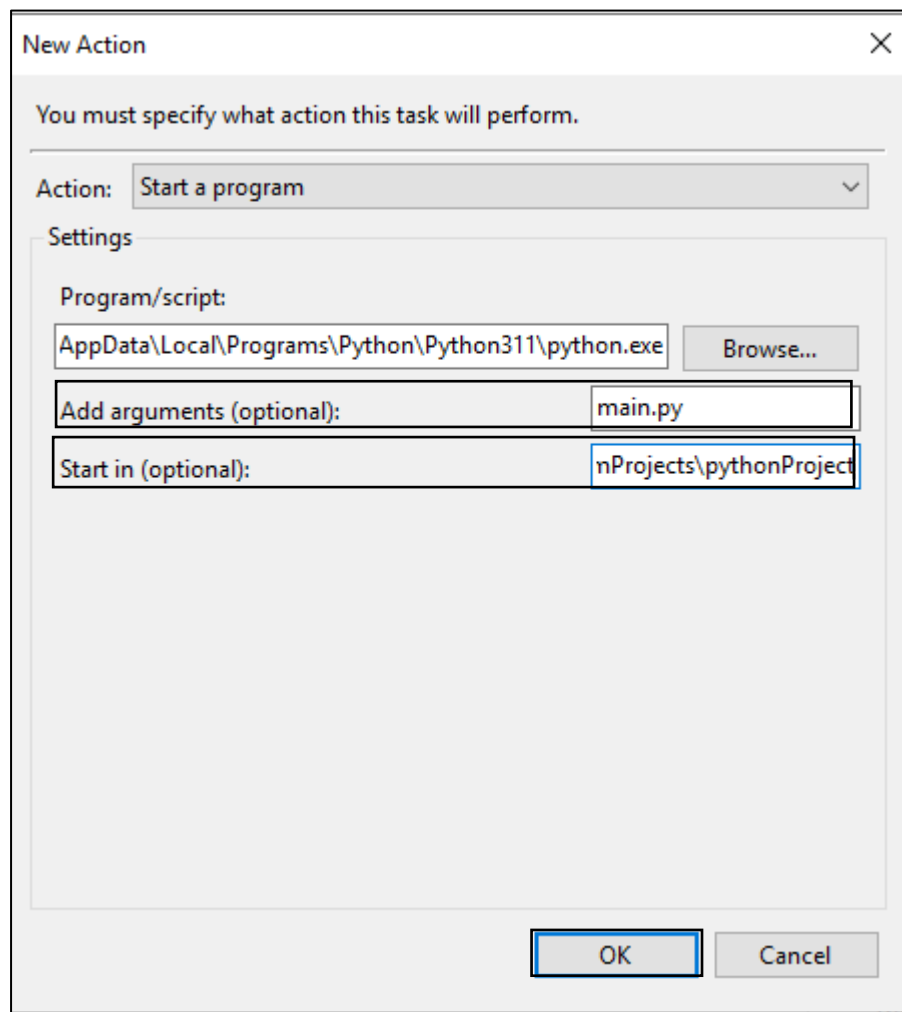


Choose python from the list and press Open



Back to the New Action Window type the file name with the extension of .py in the Add arguments(optional) field

And type the where the file is located in the Start in (optional) field and click OK

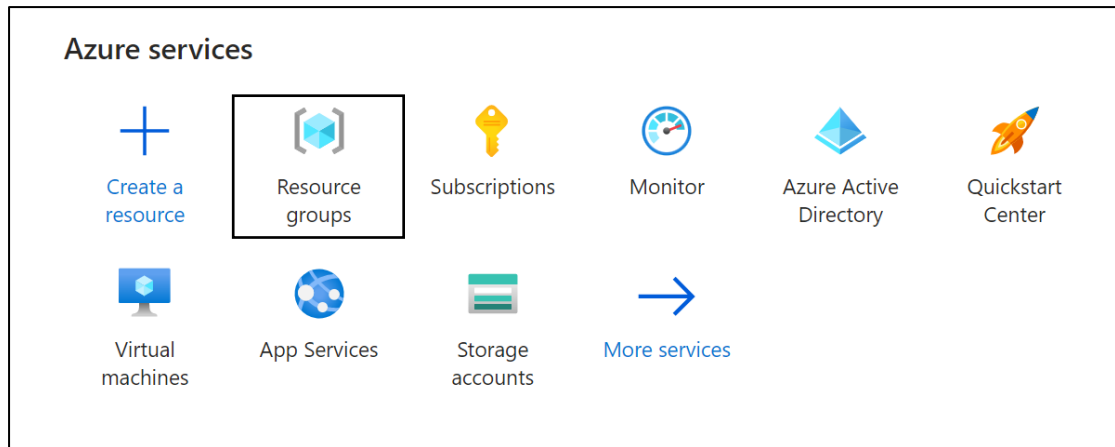


Press OK in the Actions Window to confirm and the script will run at the given time/date.

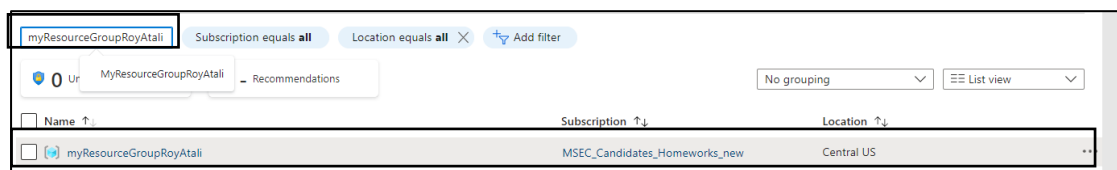
You can see the blobs in the source and destination containers at the Azure Portal

Go to the [link](#) it will open the Azure portal.

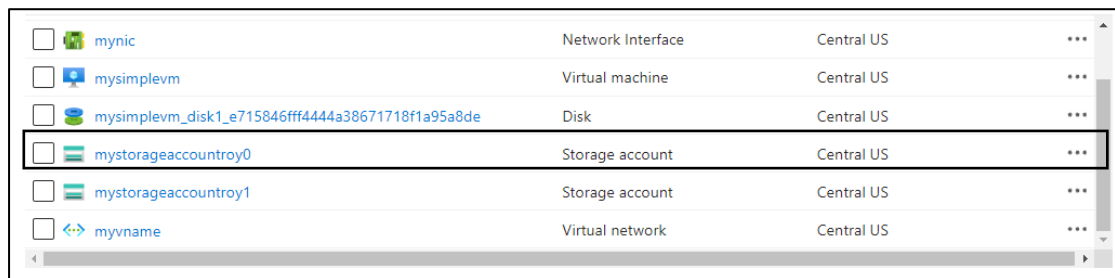
Click on Resource Groups



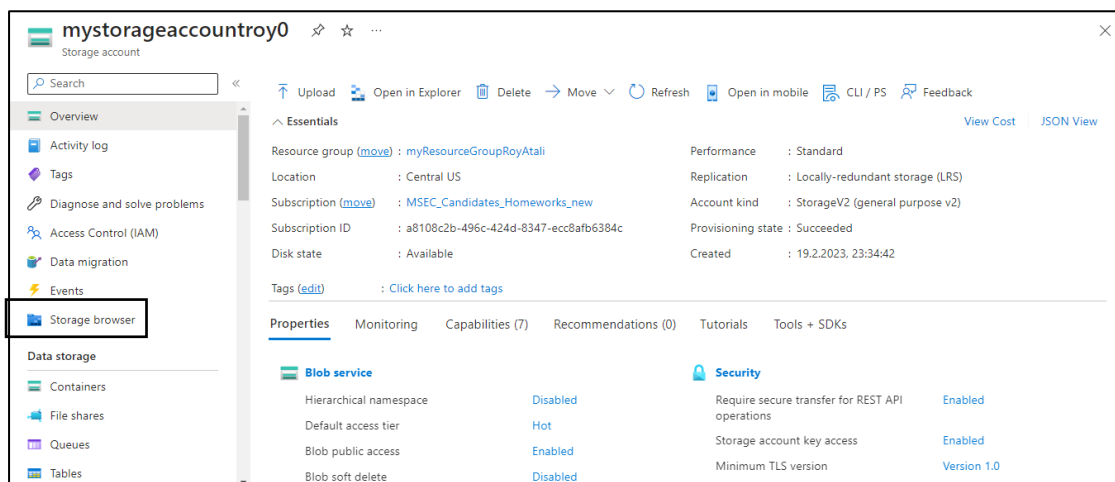
In the search bar type your resource group and click on it



Choose the source storage account from the list. For example mystorageaccountroy0



On the side menu choose Storage Browser



And then Blob Containers

mystorageaccountroy0

Privacy settings Feedback

Storage account metrics

The data provided is regularly updated about 2-4 times a day and published hourly. If your account has extremely large objects, it may be over a day between updates.

Blob containers

Number of containers	1
Number of blobs	100
Total data stored	15.4 KiB

File shares

Number of file shares	0
Number of files	0
Total data stored	0

Tables

Number of tables	5
Number of entities	227
Total data stored	766.41 KiB

Queues

Number of queues	0
Number of messages	0
Total data stored	0

Recently viewed

Choose the container we just created with the 100 blobs in it

Blob containers

Search containers by prefix

Only show active containers

Showing all 1 items

Name	Last modified	Public access level	Lease state
<input type="checkbox"/> mysrc	21.2.2023, 17:04:00	Private	Available

And we can see there are 100 blobs created

Authentication method: Access key (Switch to Azure AD User Account)

Add filter

Search blobs by prefix (case-sensitive)

Only show active blobs

Showing all 100 items

Name	Last modified	Access tier	Blob type	Size
<input type="checkbox"/> blob1	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/> blob10	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/> blob100	21.2.2023, 17:04:01	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/> blob11	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/> blob12	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/> blob13	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/> blob14	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B

Now navigate the same way to the destination account storage and we can see there are 100 copies of the source blobs

Authentication method: Access key ([Switch to Azure AD User Account](#))

[Add filter](#)

Only show active blobs

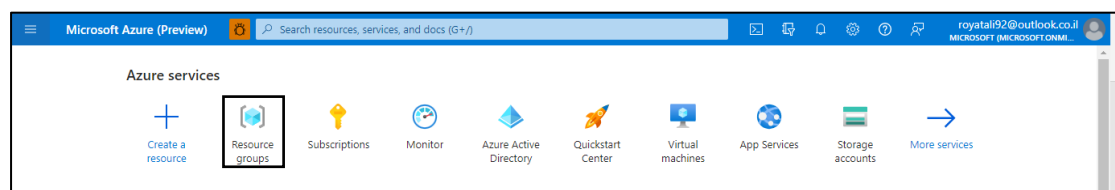
Showing all 100 items

<input type="checkbox"/>	Name	Last modified	Access tier	Blob type	Size
<input type="checkbox"/>	copy_blob1	21.2.2023, 17:04:02	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/>	copy_blob10	21.2.2023, 17:04:02	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/>	copy_blob1...	21.2.2023, 17:04:03	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/>	copy_blob11	21.2.2023, 17:04:02	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/>	copy_blob12	21.2.2023, 17:04:02	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/>	copy_blob13	21.2.2023, 17:04:02	Hot (Inferred)	Block blob	13 B
<input type="checkbox"/>	copy_blob14	21.2.2023, 17:04:02	Hot (Inferred)	Block blob	13 B

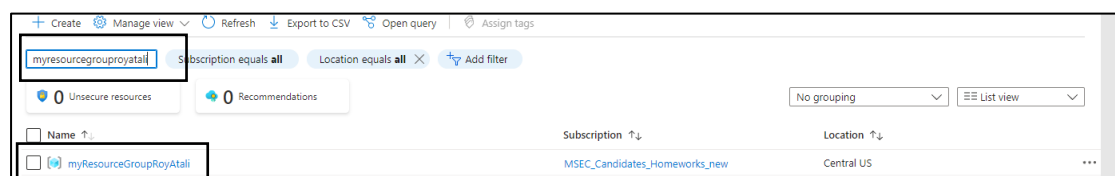
6. Monitoring

In order to monitor our system of server and 2 storage accounts we use the Azure portal **Monitor** blade.

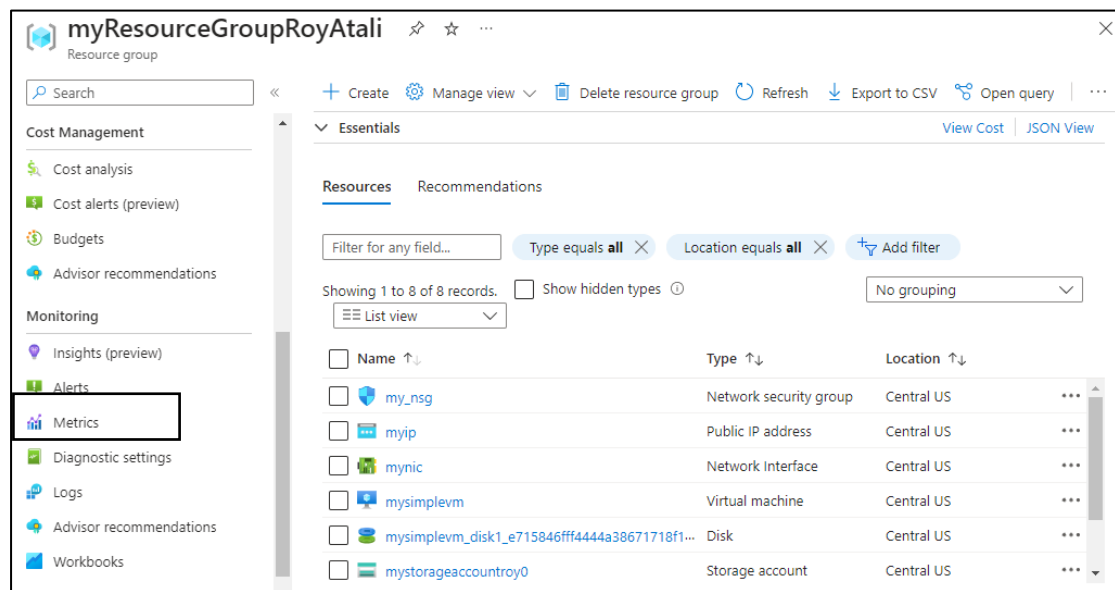
Go to this [link](#) it will open the Azure portal. On the homepage click **Resource groups**.



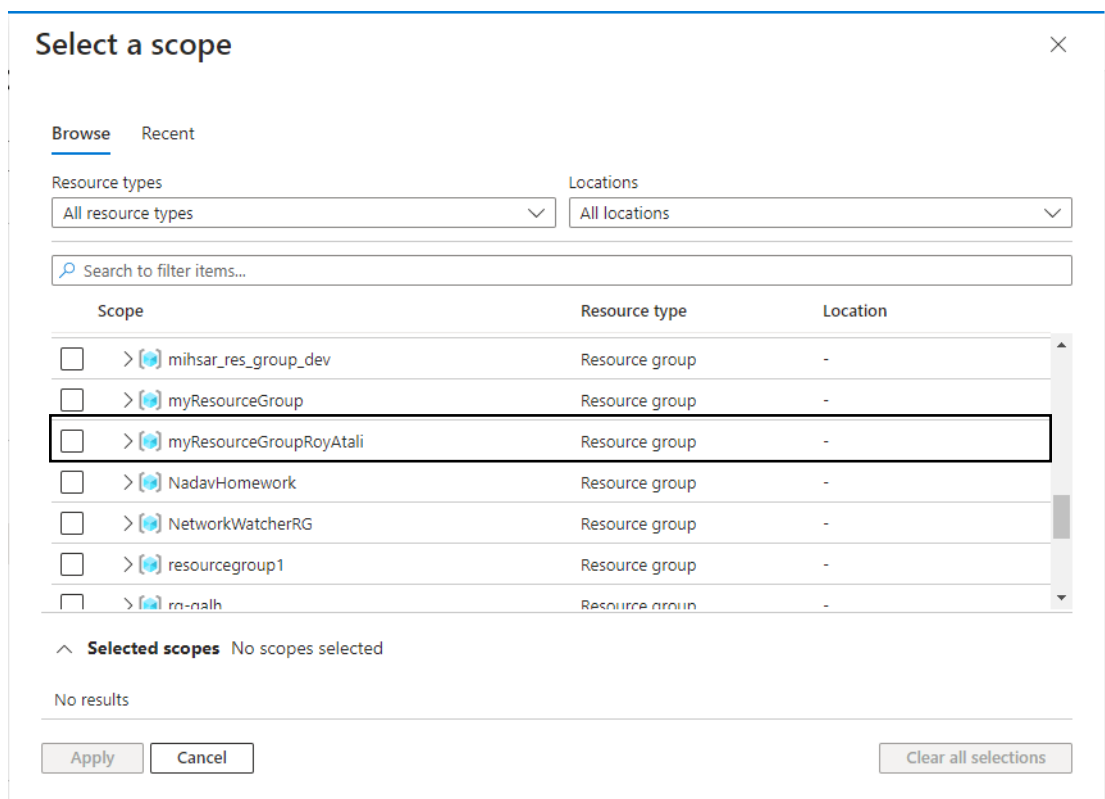
Then in the search bar type the name of your resource group and select it from the list



In the resource group window on the left side menu choose metrics



In the metrics Window select scope – the resource group in which the resources to be monitor are found



Select the resource type you want to monitor and press **Apply**

Refine scope

Resource type *
Virtual machines

Location *
Central US

^ Selected scopes 1 scope

> myResourceGroupRoyAtali Resource group

Apply

Cancel

Clear all selections

Then we can select different metrics we want to monitor about the resource

+ New chart Refresh Share Feedback
Local Time: Last 24 hours (Automatic)

Chart Title

Add metric Add filter Line chart Drill into Logs New alert rule

Apply splitting Save to dashboard

Scope: MSEC_Candidates_Homew...
Metric Namespace: Virtual machine standar...
Metric: select metric

Aggregation: Select aggregation

./Resour... = myResour...

100
90
80
70
60
50
40

Select a metric above to see data appear on

Filter + Split

Plot multiple metrics

Build custom

Available Memory Bytes (Preview)

CPU Credits Consumed

CPU Credits Remaining

Data Disk Bandwidth Consumed Percentage

Data Disk IOPS Consumed Percentage

Data Disk Max Burst Bandwidth

Data Disk Max Burst IOPS

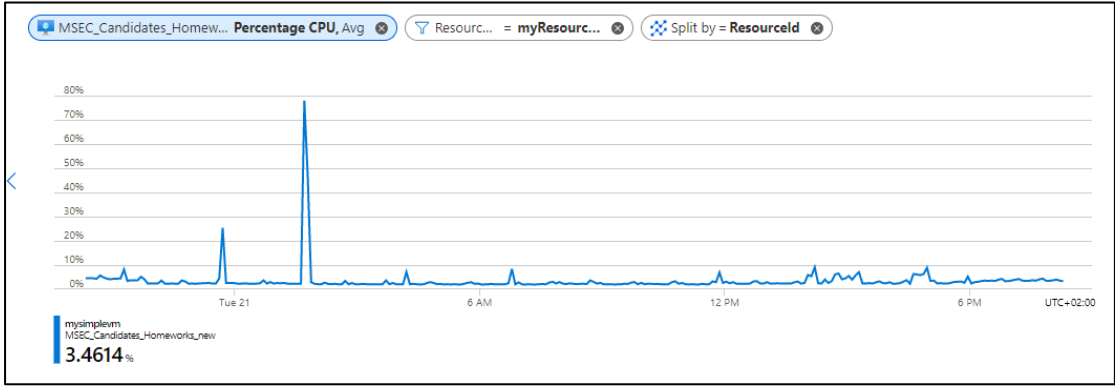
Data Disk Queue Depth

The different metrics for the server (VM) I chose to focus on are:

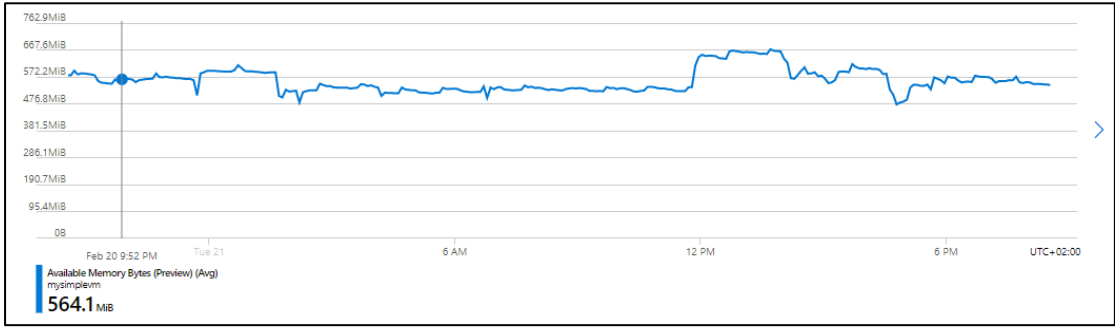
1. **Server CPU utilization:** Monitor the percentage of CPU usage on the server. High CPU utilization could indicate performance issues or resource constraints.
2. **Server Available Memory:** This metric represents the amount of memory available for use by applications and processes. You can monitor it to ensure that the virtual machine has sufficient memory to handle its workload.
3. **Server Network Traffic:** Monitor the amount of inbound and outbound traffic to and from the server and storage accounts. High network traffic could indicate a security threat or network congestion.

6.1 Server (Virtual Machine Metrics)

6.1.1 Server CPU percentage/utilization

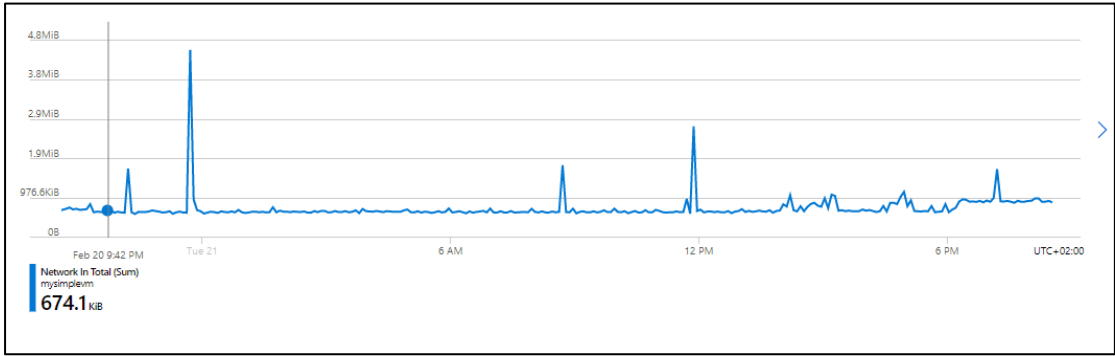


6.1.2 Server Available Memory

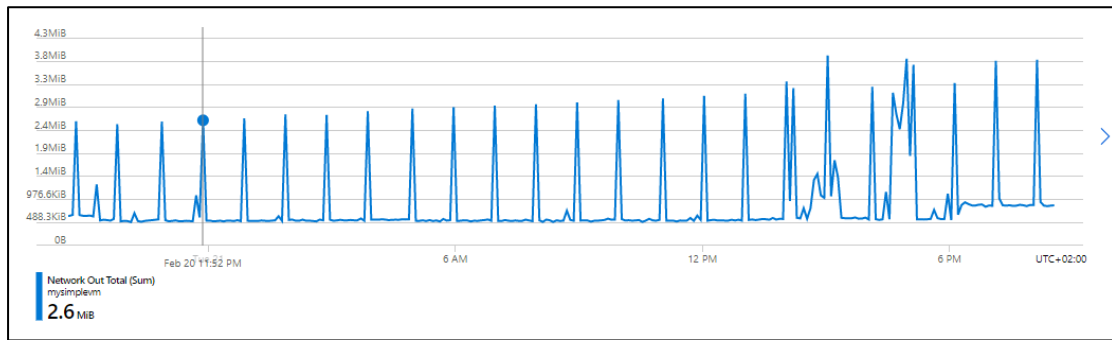


6.1.3 Server Network Traffic

6.1.3.1 Inbound Traffic



6.1.3.2 Outbound Traffic

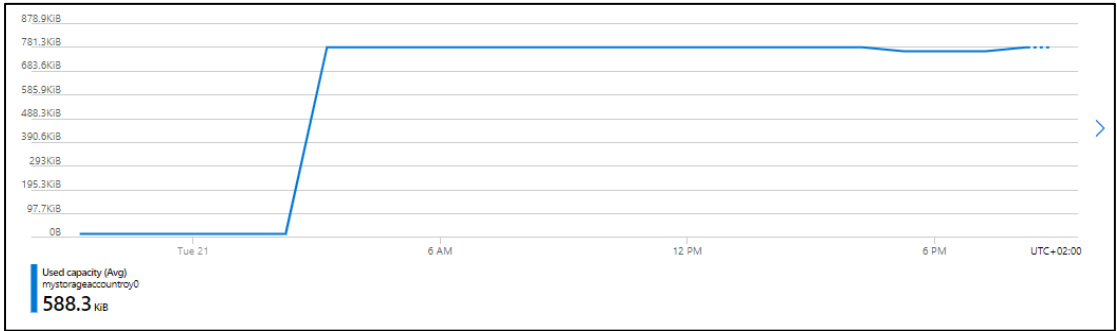


The different metrics for the storage accounts I chose to focus on are:

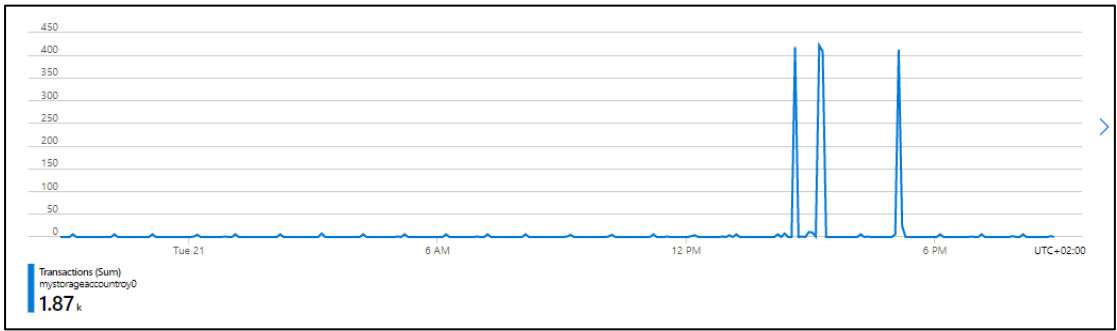
1. **Used capacity:** Used capacity refers to the amount of storage space that is being used by the data stored in your storage account. By monitoring used capacity, you can identify when you are approaching the storage limit of your account and take action to ensure that you have sufficient storage capacity for your data. This can include adding additional storage capacity to your account, optimizing your data storage practices to reduce data usage, or archiving or deleting data that is no longer needed.
2. **Transactions:** This metric measures the number of operations performed on the storage account, such as reads, writes, and deletes. Monitoring this metric can help you understand the workload on the storage account and identify any potential issues related to transaction volume.
3. **Egress:** This metric measures the amount of data that is being sent out of the storage account. Monitoring this metric can help you understand the bandwidth requirements of your applications and identify any potential issues related to data transfer.
4. **Ingress:** This metric measures the amount of data that is being sent into the storage account. Monitoring this metric can help you understand the data transfer requirements of your applications and identify any potential issues related to data ingestion.
5. **Availability:** This metric measures the availability of the storage account over a period of time. Monitoring this metric can help you understand the reliability of the storage account and identify any potential issues related to downtime or service interruptions.
6. **Latency:** This metric measures the time it takes for a request to be processed by the storage account. Monitoring this metric can help you understand the performance of the storage account and identify any potential issues related to slow response times.

6.2 Source Storage Account Metrics

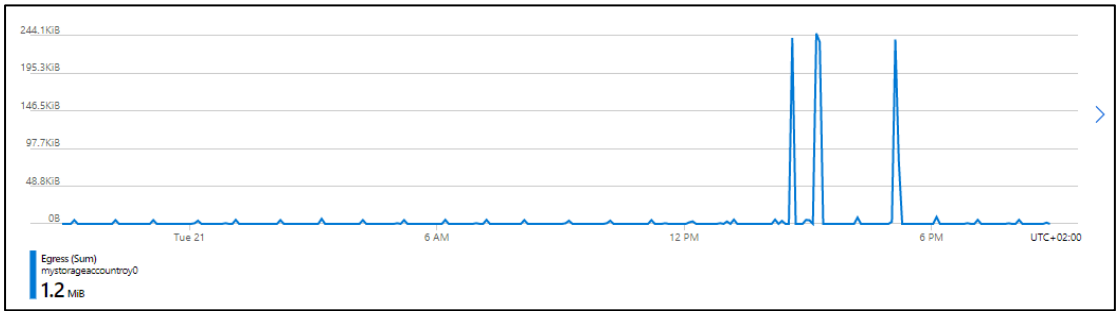
6.2.1 Used capacity



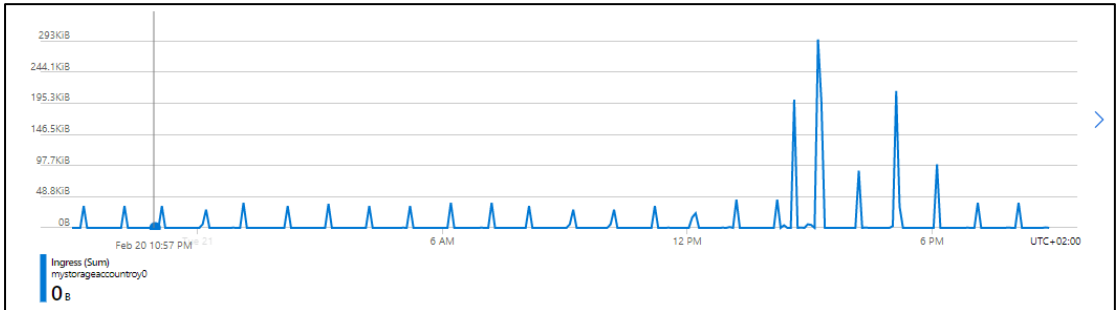
6.2.2 Transactions



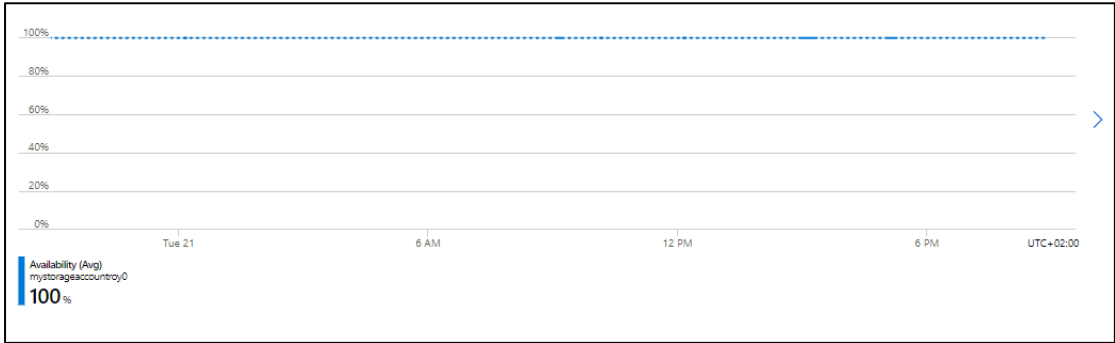
6.2.3 Egress



6.2.4 Ingress

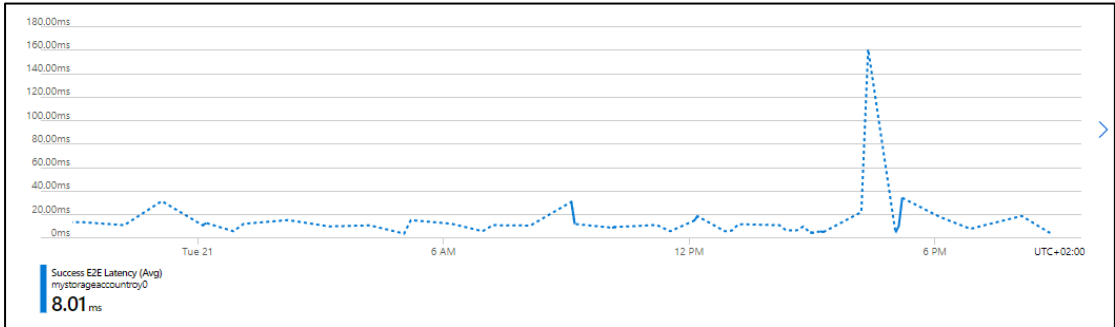


6.2.5 Availability

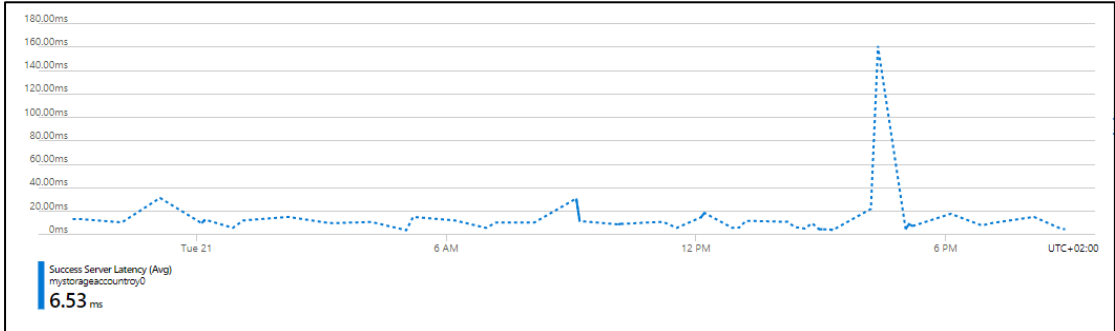


6.2.6 Latency

6.2.6.1 E2E Latency

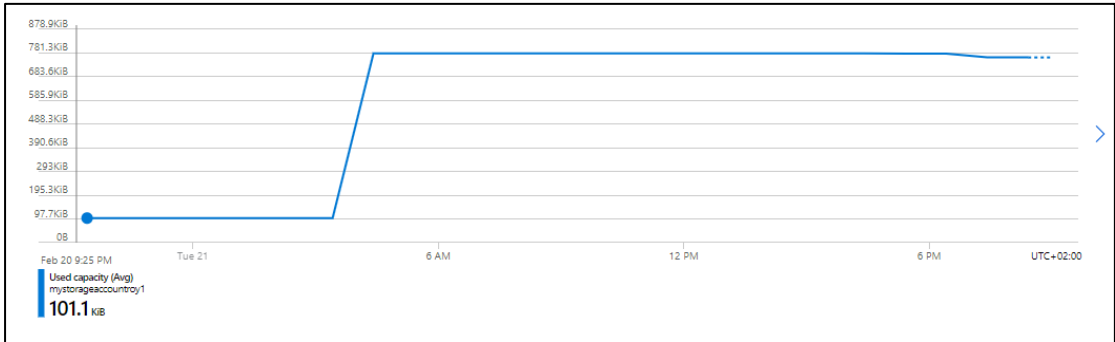


6.2.6.2 Success Server Latency

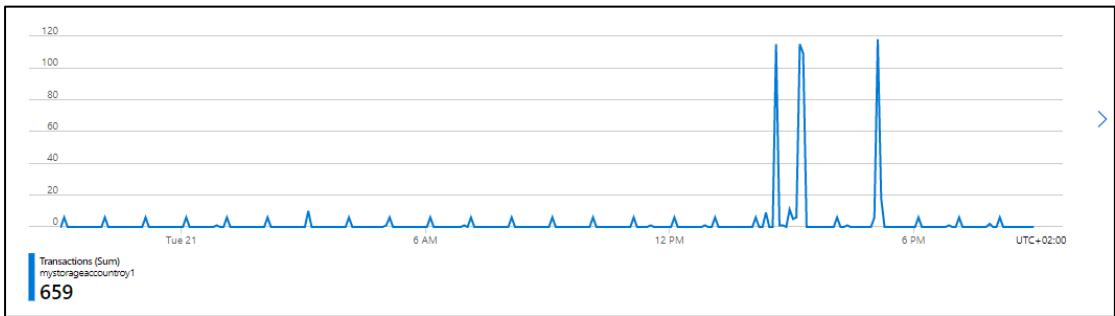


6.3 Destination Storage Account Metrics

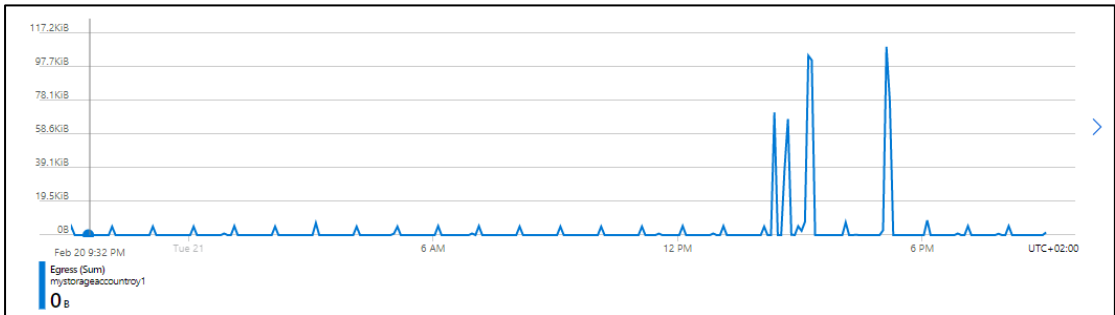
6.3.1 Used capacity



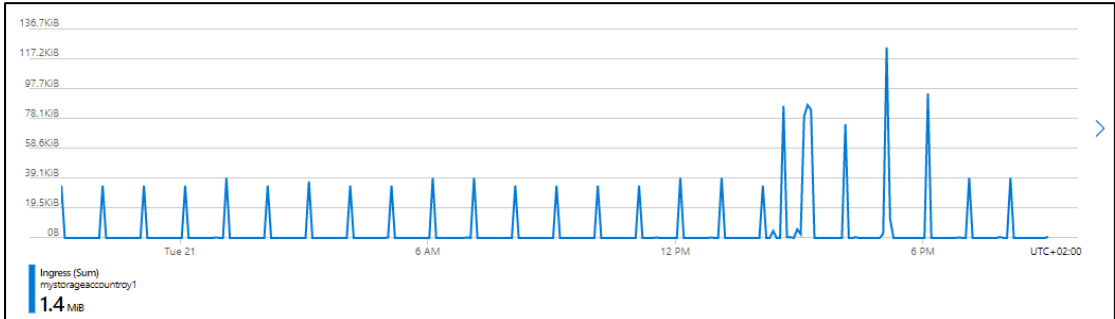
6.3.2 Transactions



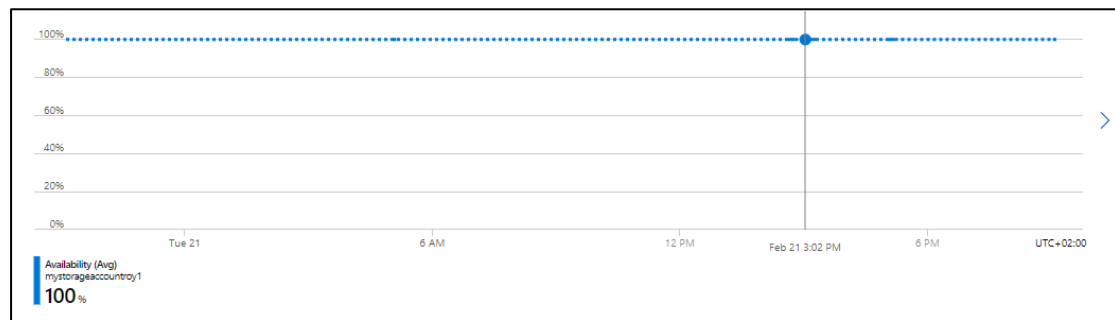
6.3.3 Egress



6.3.4 Ingress

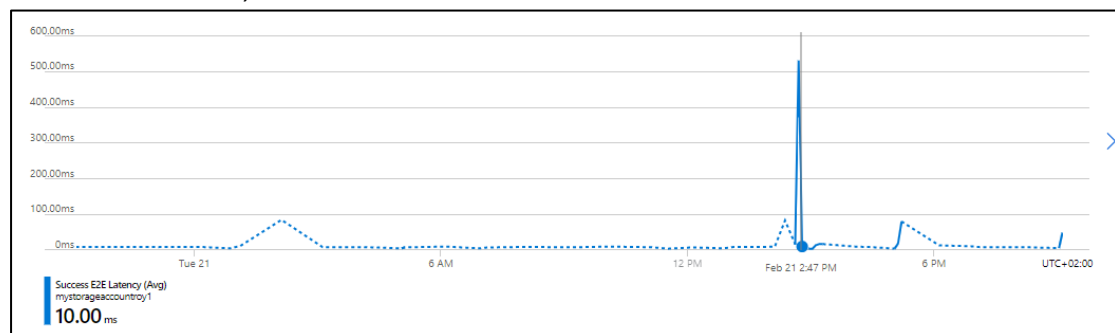


6.3.5 Availability

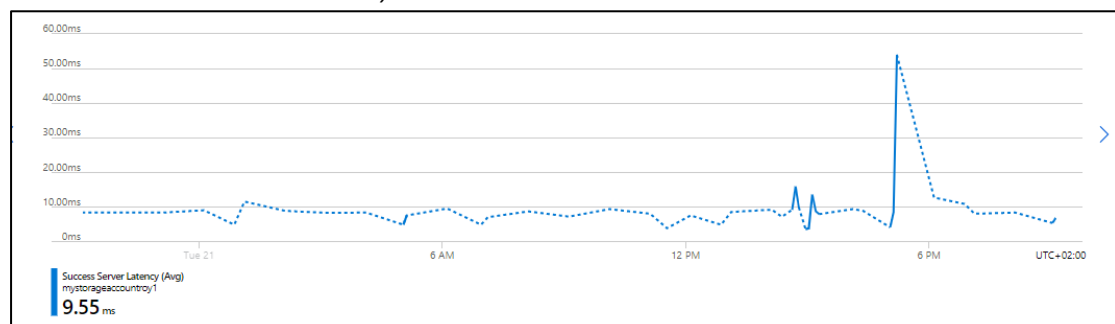


6.3.6 Latency

6.3.6.1 E2E Latency



6.3.6.2 Success Server Latency



7. Github Link

Go to this [link](#). it will open the GitHub repository that contains all the script files for this project.