Microsoft Homework Assignment

Name: Roy Atali

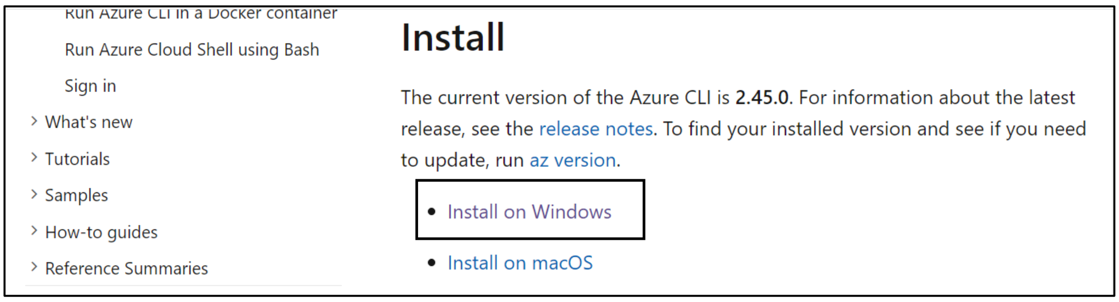
Date: 21/02/2023

Table Of Contents

1. Prerequisites Requirements

1.1 Azure CLI Installation

Go to this link. In the Install section click on **Install on Windows**

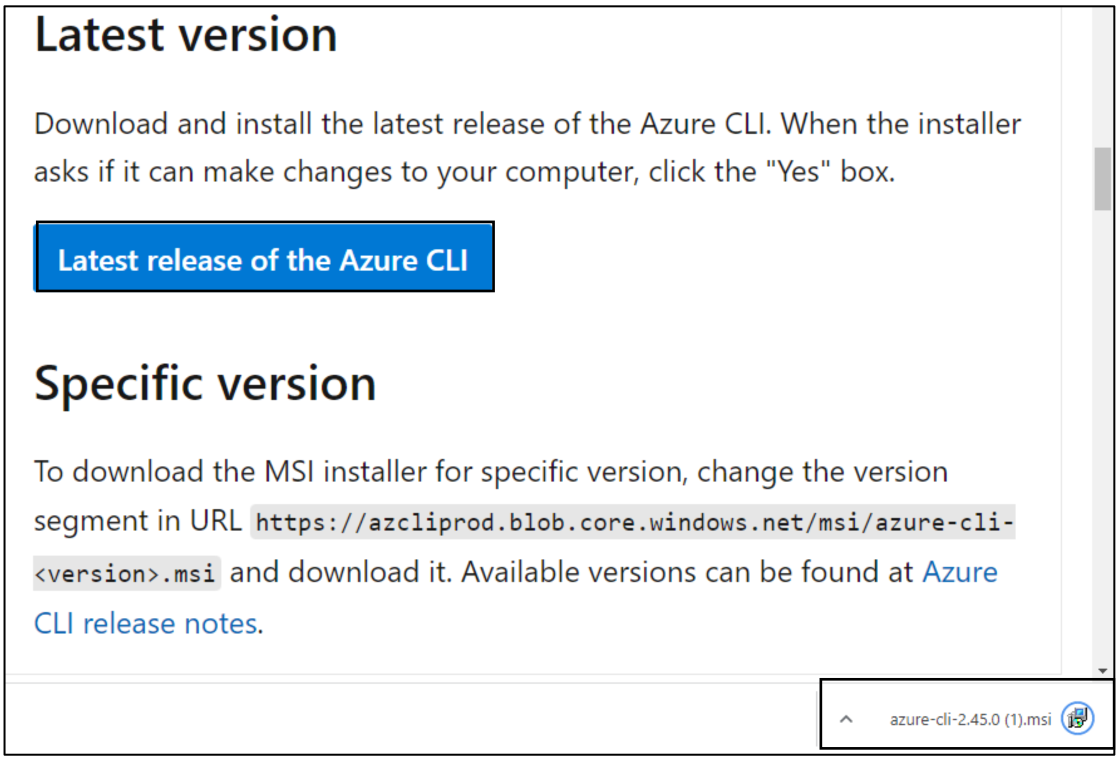

Run Azure CLI in a Docker container
Run Azure Cloud Shell using Bash
Sign in
What's new
Tutorials
Samples
How-to guides
Reference Summaries

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.



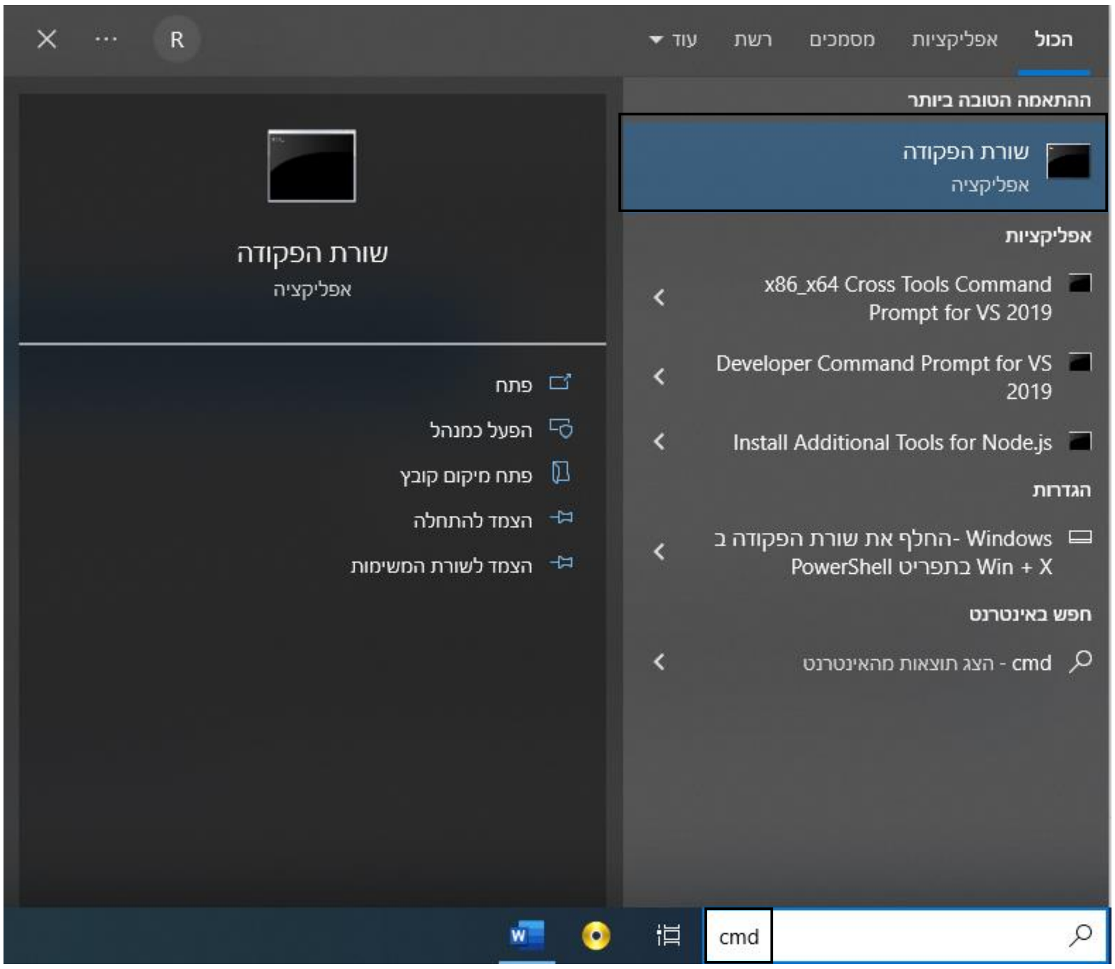
Latest version

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.

Specific version

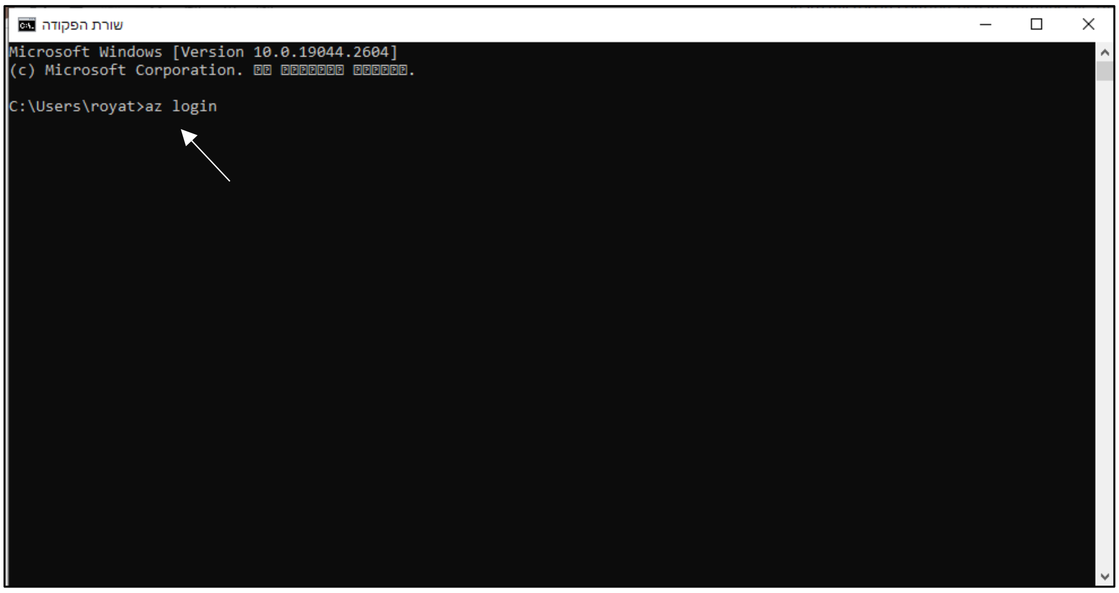
To download the MSI installer for specific version, change the version segment in URL `https://azcliprod.blob.core.windows.net/msi/azure-cli-<version>.msi` and download it. Available versions can be found at Azure CLI release notes.

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



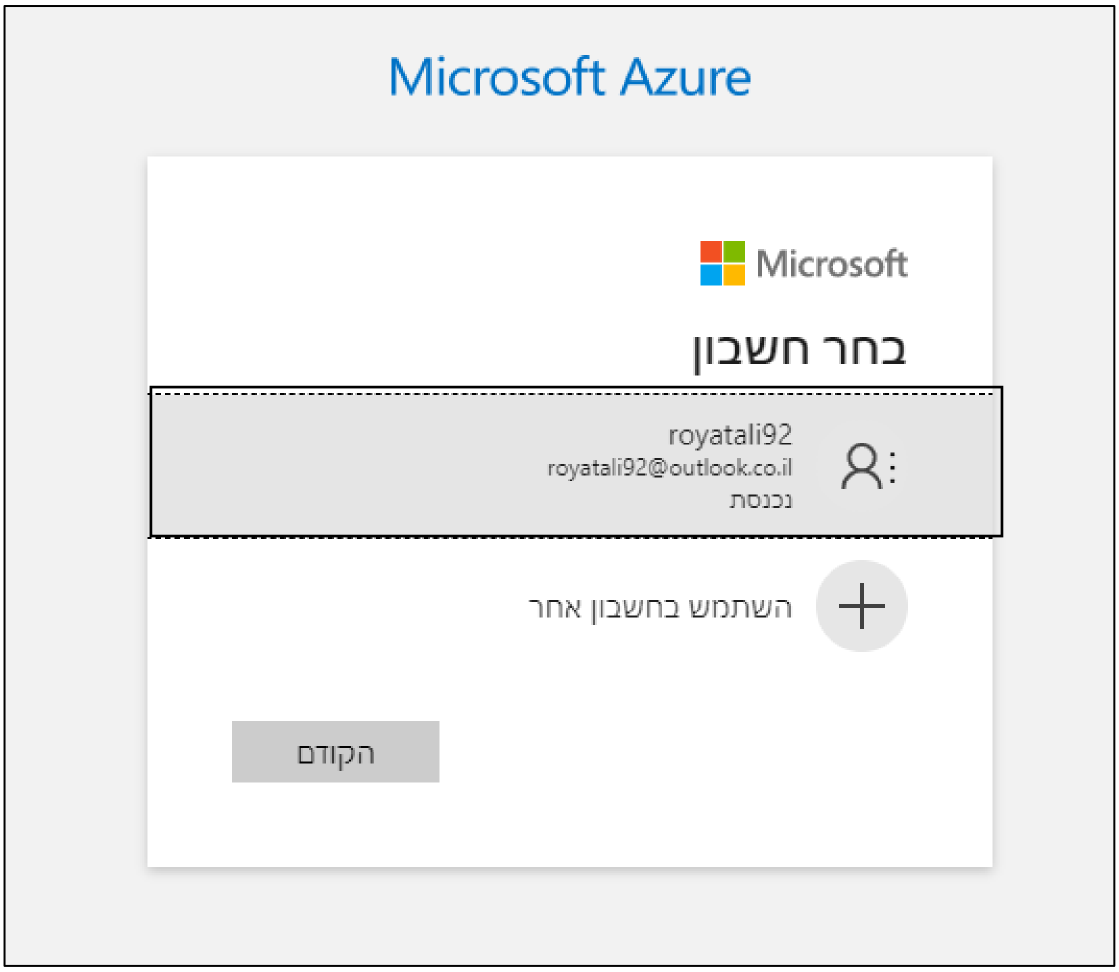
הכול
אפליקציות
מסמכים
רשת
עוד
ההתאמה הטובה ביותר
שורת הפקודה
אפליקציה
אפליקציות
x86_x64 Cross Tools Command Prompt for VS 2019
Developer Command Prompt for VS 2019
Install Additional Tools for Node.js
הגדרות
החלף את שורת הפקודה ב- Windows PowerShell בתפריט Win + X
חפש באינטרנט
cmd - הצג תוצאות מהאינטרנט
שורת הפקודה
אפליקציה
פתח
הפעל כמנהל
פתח מיקום קובץ
הצמד להתחלה
הצמד לשורת המשימות
cmd

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

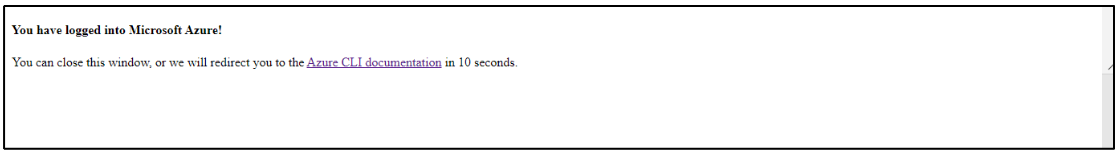


```
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. ▯▯ ▯▯▯▯▯▯▯ ▯▯▯▯▯▯.

C:\Users\royat>az login
```

In the browser select your Microsoft outlook account



This step will authenticate your azure CLI with your account

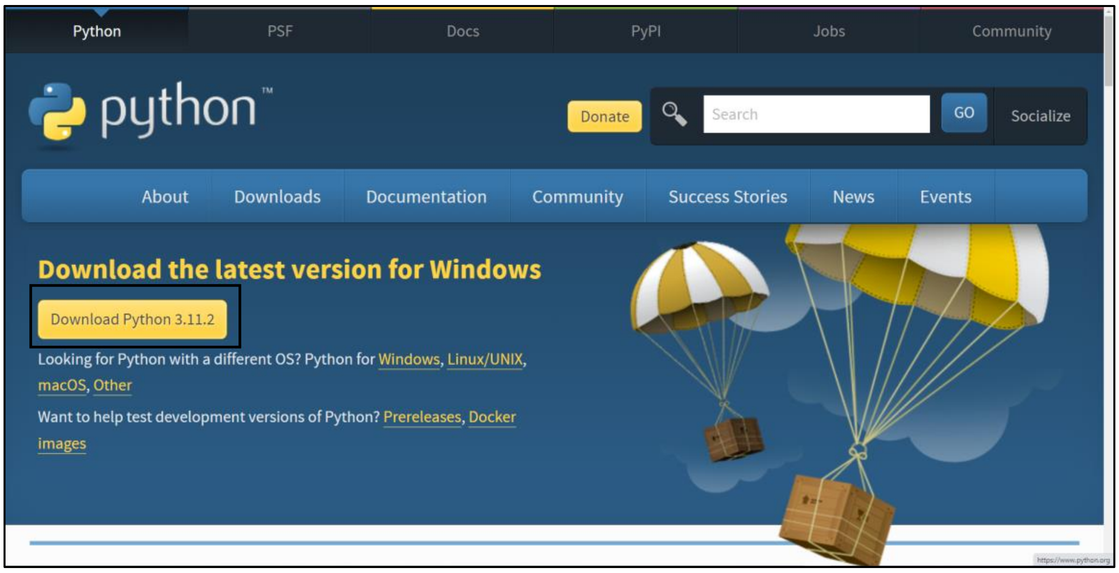

You have logged into Microsoft Azure!

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.

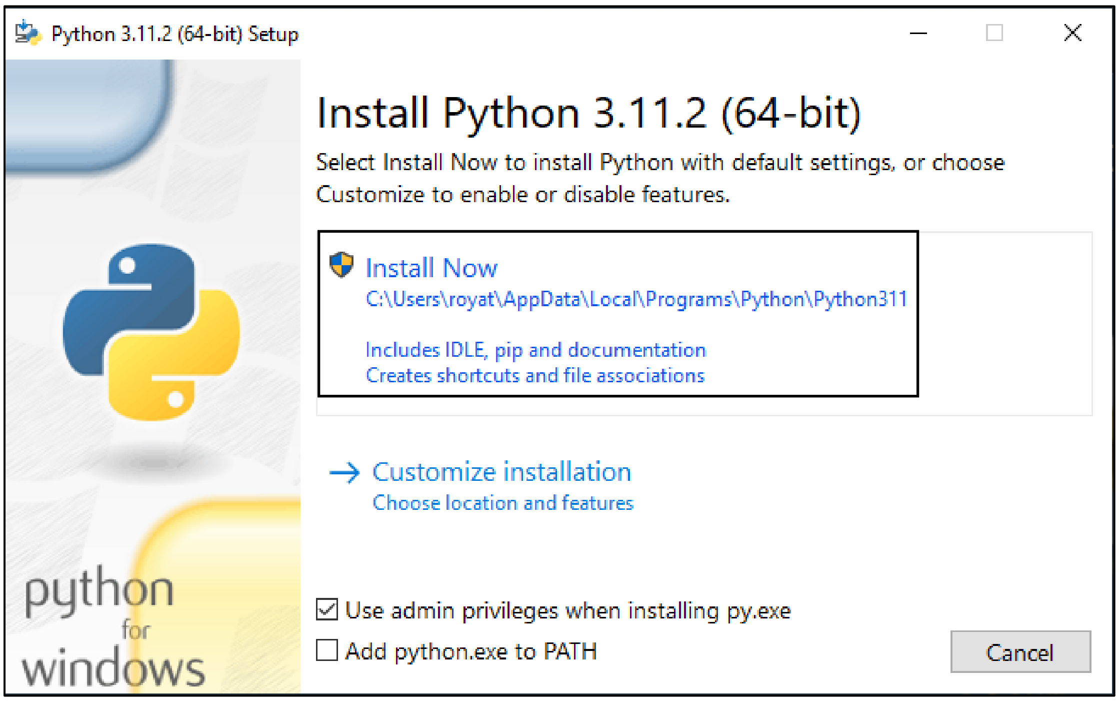
1. Go to this link and download the latest version of python for your machine



Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python? Prereleases, Docker images

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



Python 3.11.2 (64-bit) Setup

Install Python 3.11.2 (64-bit)

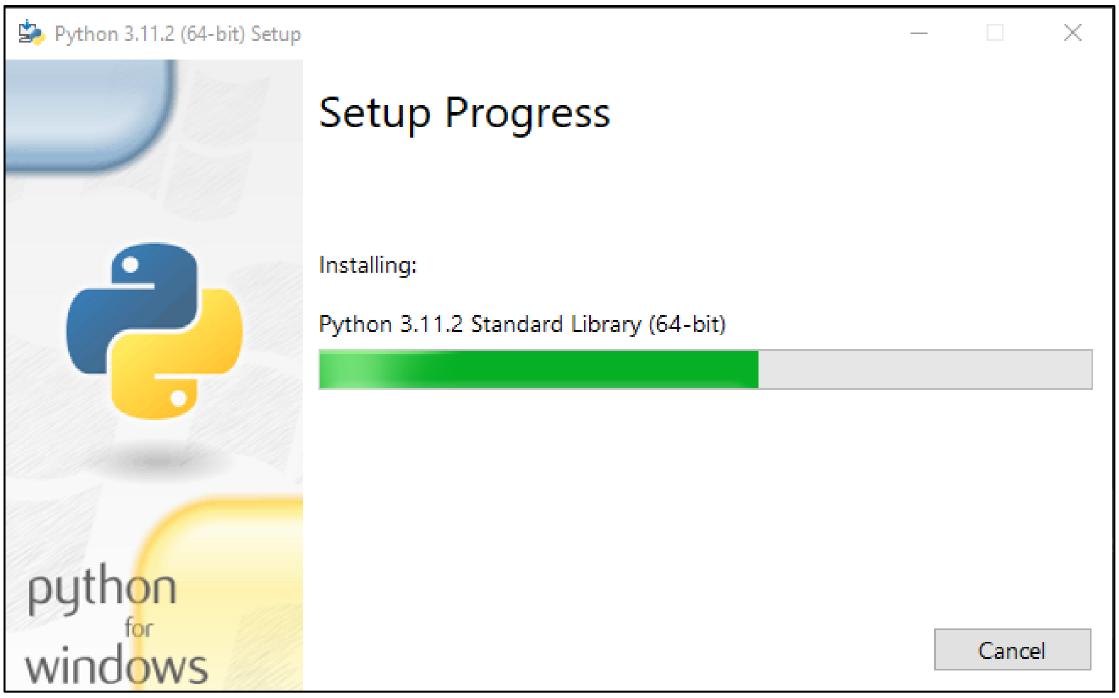
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\royat\AppData\Local\Programs\Python\Python311

Includes IDLE, pip and documentation
Creates shortcuts and file associations

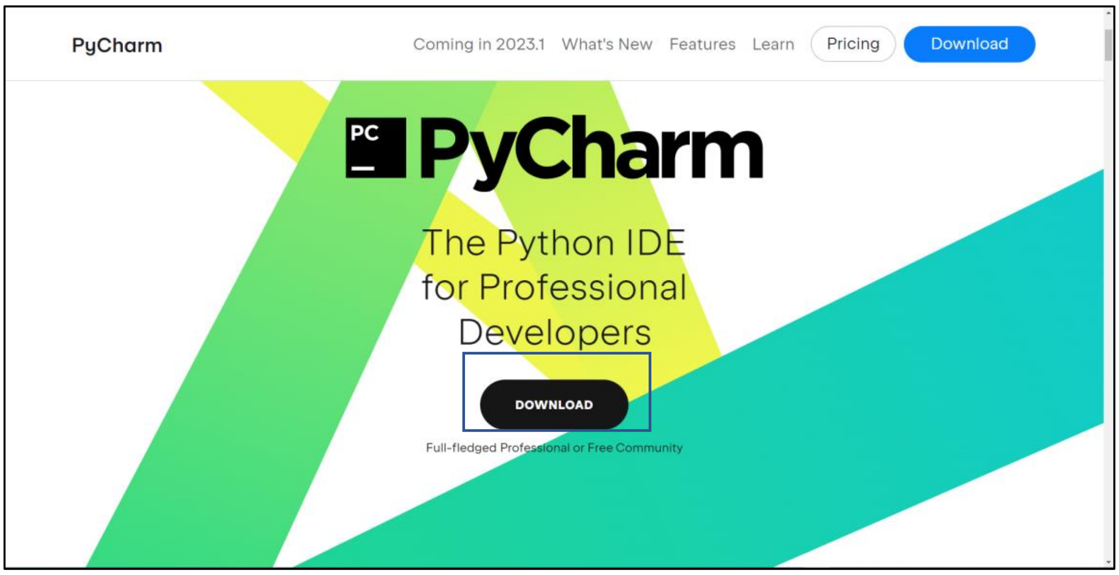
Customize installation
Choose location and features

☑ Use admin privileges when installing py.exe
☐ Add python.exe to PATH

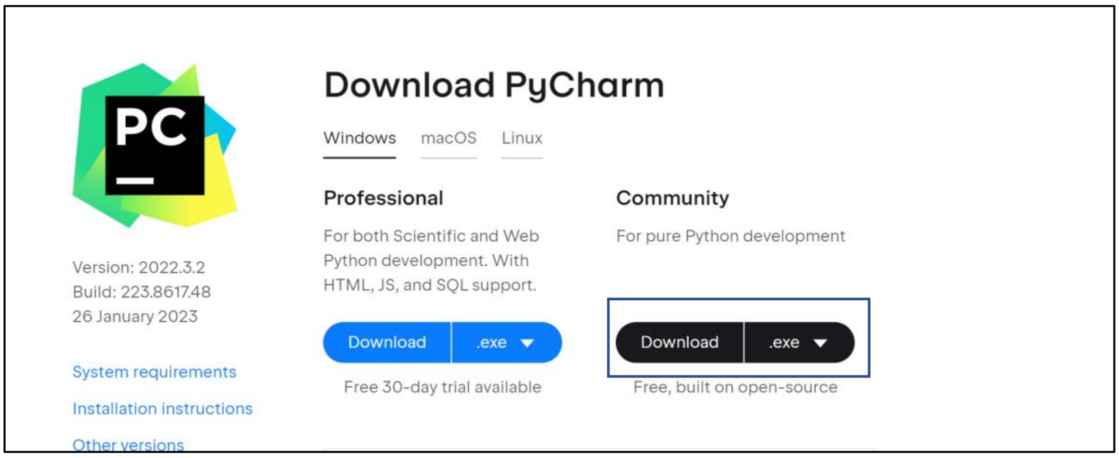

Python 3.11.2 (64-bit) Setup
Setup Progress
Installing:
Python 3.11.2 Standard Library (64-bit)
python for windows
Cancel

After that you will need **PyCharm**

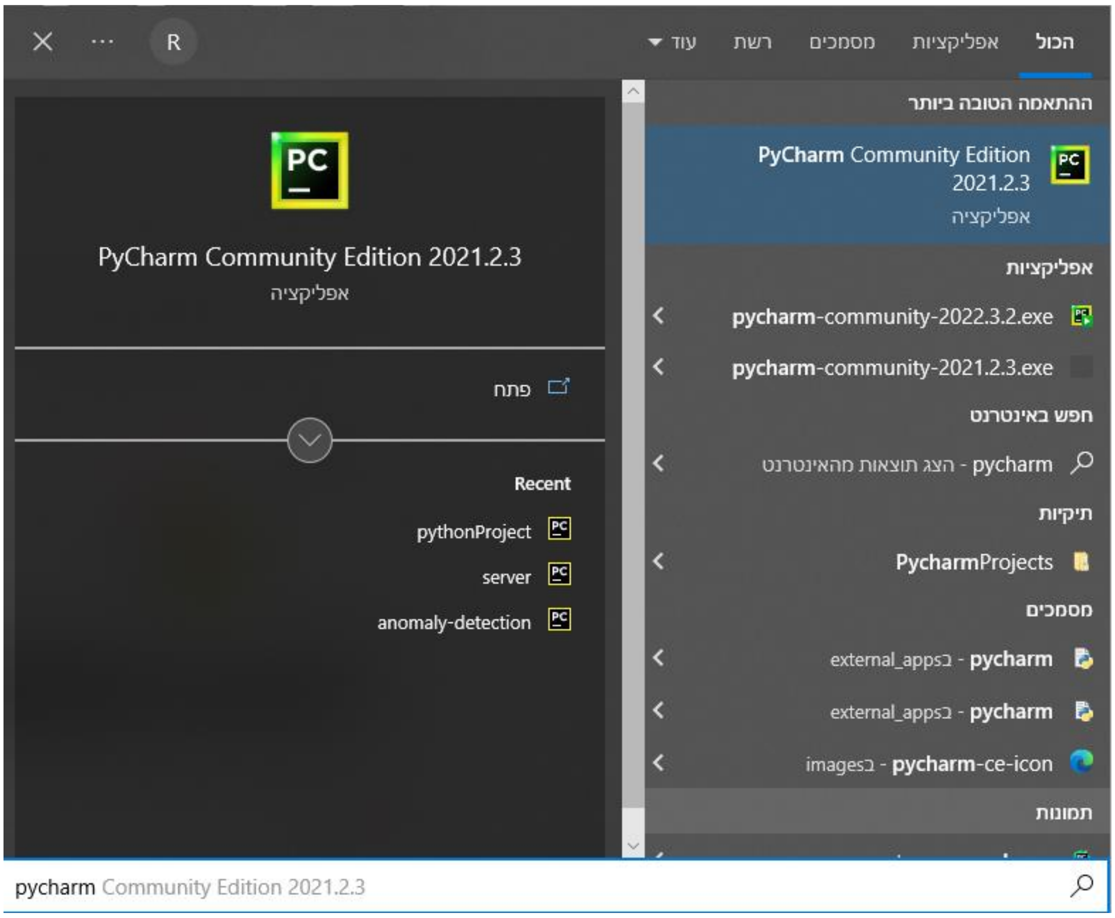
Go to this link and download **PyCharm** for your machine


PyCharm
Coming in 2023.1
What's New
Features
Learn
Pricing
Download
PC PyCharm
The Python IDE for Professional Developers
DOWNLOAD
Full-fledged Professional or Free Community

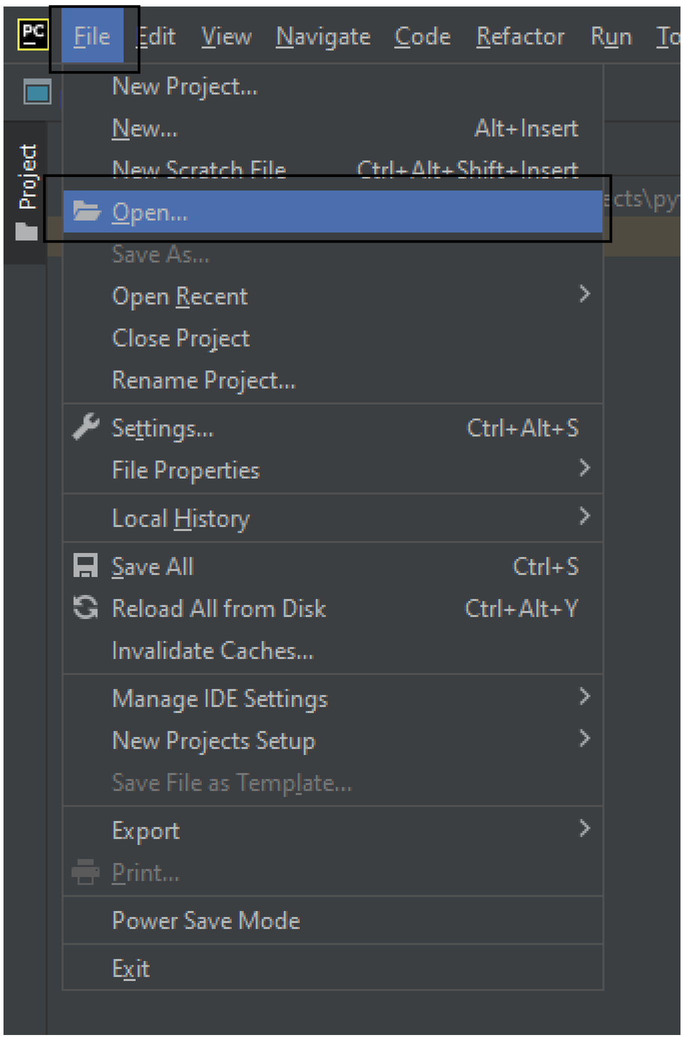
Click on download and download the community version and install it afterwards


Download PyCharm
Windows
macOS
Linux
Professional
For both Scientific and Web Python development. With HTML, JS, and SQL support.
Download
.exe
Free 30-day trial available
Community
For pure Python development
Download
.exe
Free, built on open-source
Version: 2022.3.2
Build: 223.8617.48
26 January 2023
System requirements
Installation instructions
Other versions

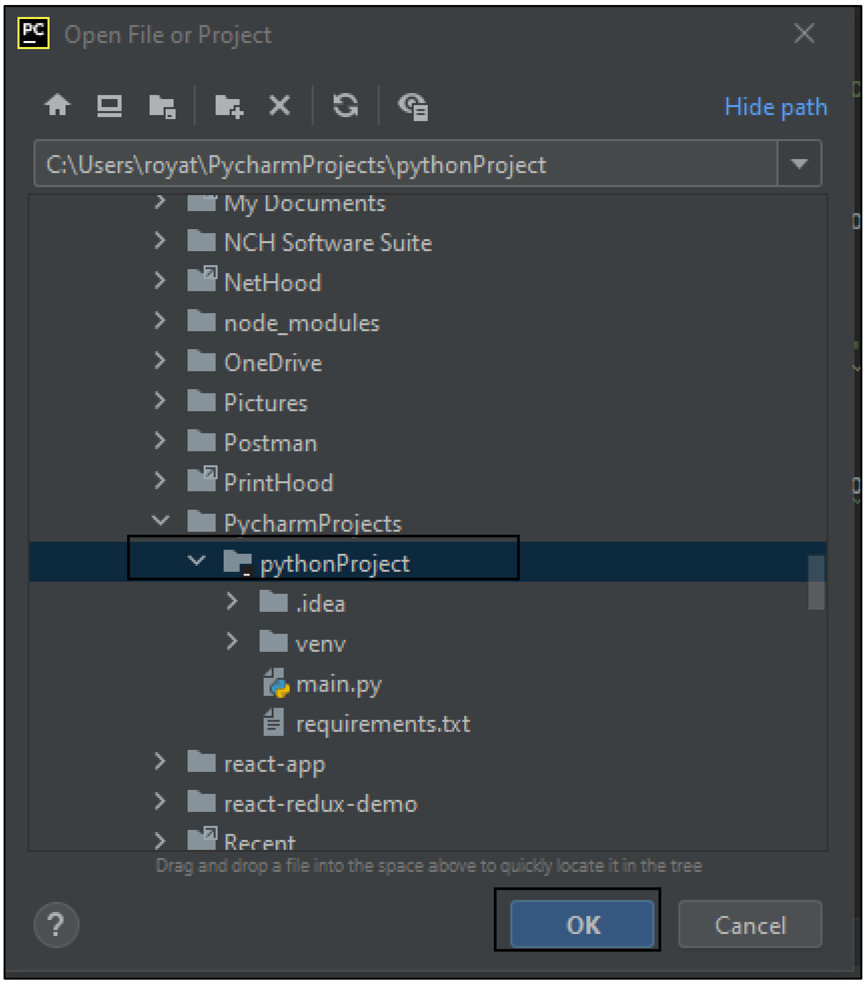
Once installed open **PyCharm** by typing **PyCharm** in the windows search bar and click to open it


הכול
אפליקציות
מסמכים
רשת
עוד
ההתאמה הטובה ביותר
PyCharm Community Edition 2021.2.3
אפליקציה
אפליקציות
pycharm-community-2022.3.2.exe
pycharm-community-2021.2.3.exe
חפש באינטרנט
pycharm - הצג תוצאות מהאינטרנט
תיקיות
PycharmProjects
מסמכים
pycharm - בexternal_apps
pycharm - בexternal_apps
pycharm-ce-icon - בimages
תמונות
PyCharm Community Edition 2021.2.3
אפליקציה
פתח
Recent
pythonProject
server
anomaly-detection
pycharm Community Edition 2021.2.3

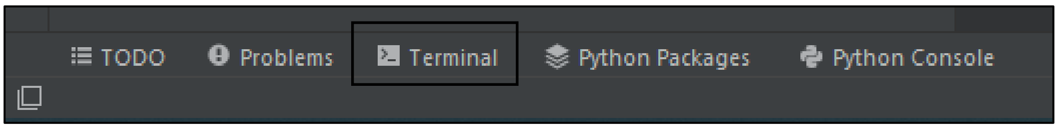
Once it opens click on **file->open**



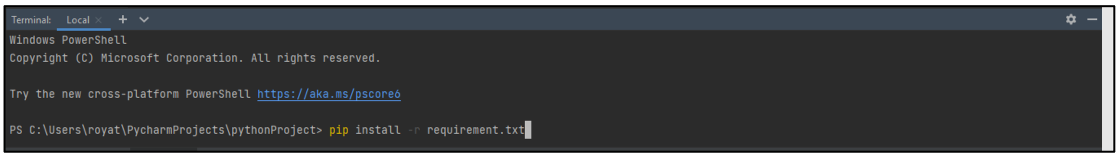
In the window that just opened select the folder pythonProject where it located and press **OK**



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

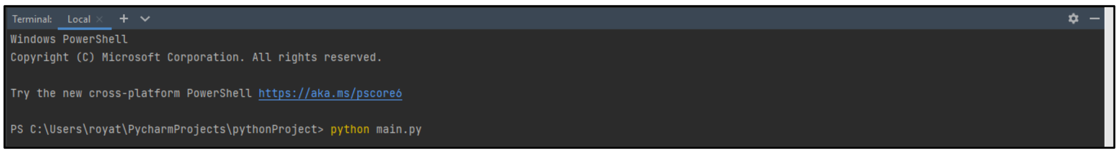


In the terminal type **pip install -r requirements.txt.** that will install all the necessary packages related to the project.



```
Terminal:   Local ×   +
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\royat\PycharmProjects\pythonProject> pip install -r requirement.txt
```

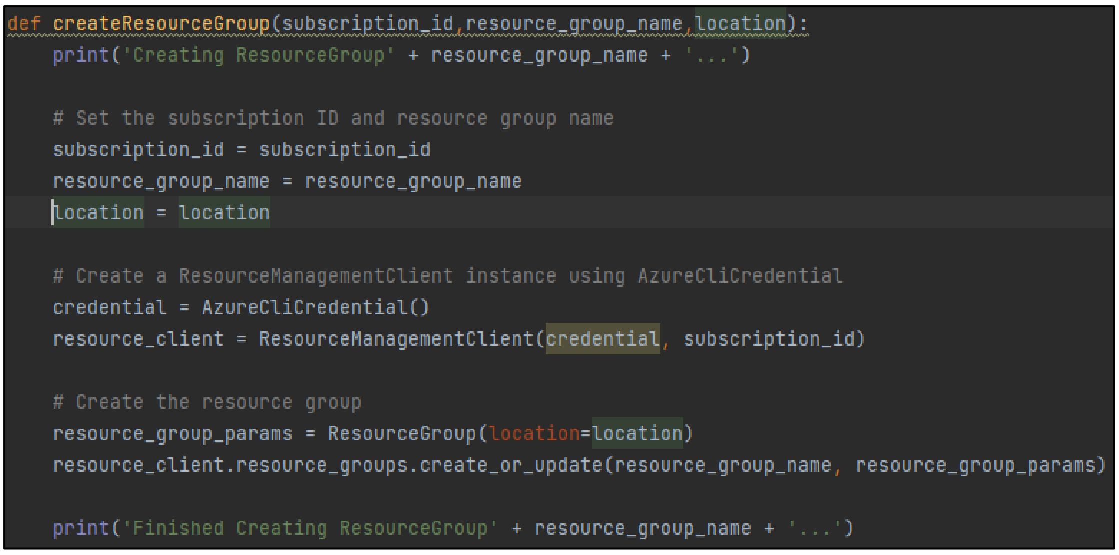
After all the requirements installed. Type in the terminal **python main.py** to run the script.



```
Terminal:   Local ×   +
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\royat\PycharmProjects\pythonProject> python main.py
```

2. Code Snippets

2.1 createResourceGroup



```python
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:

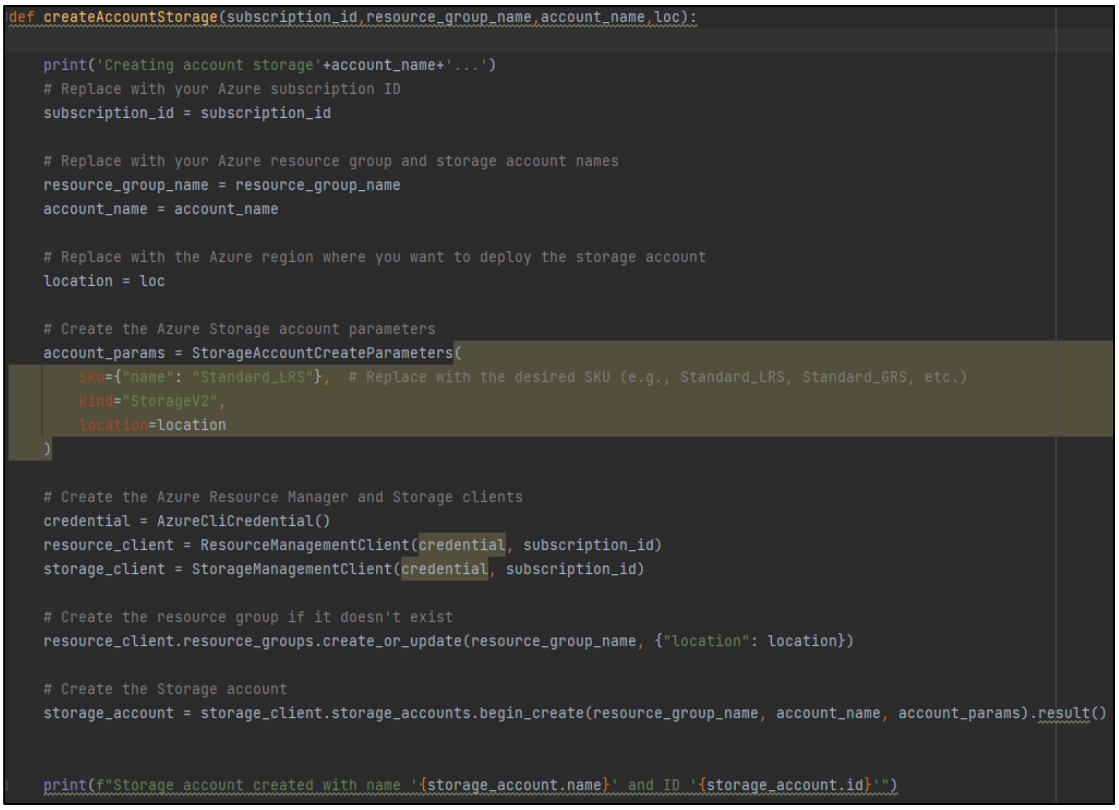
- subscription_id: The ID of your Azure subscription
- resource_group_name: The name of the resource group to be created
- location: The Azure region where you want to deploy the resource group

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



```python
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.

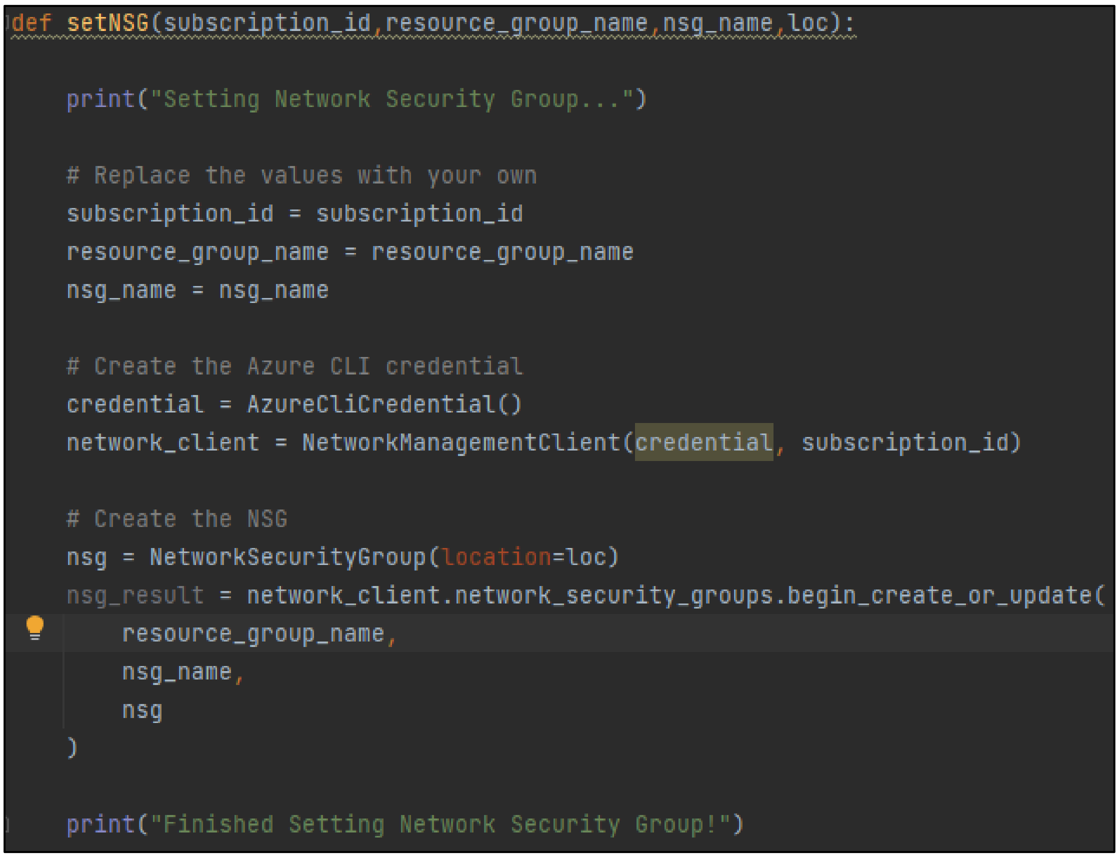
2.4 createWindowsVirtualMachine

```
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



```python
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!")
```

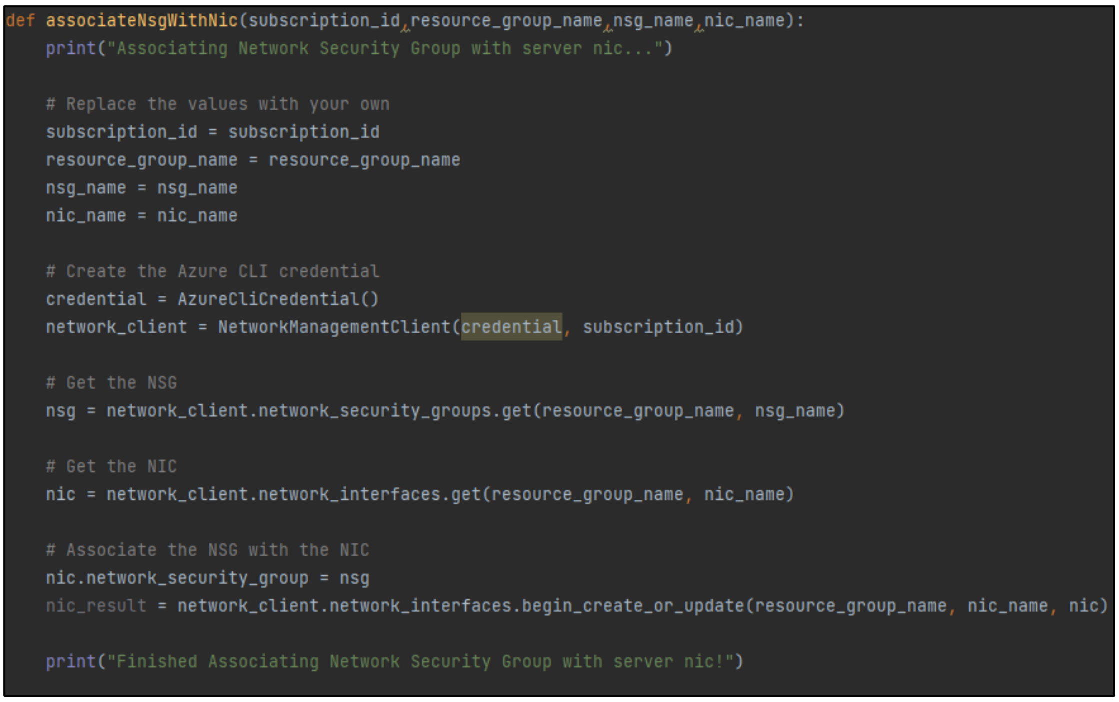
This Python code defines a function setNSG that sets up a new Network Security Group (NSG) in Azure using the Azure SDK for Python. Here's what the code does:

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



```python
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(resourc
e_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.

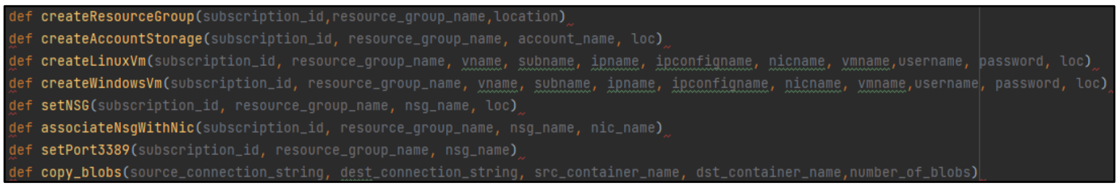
2.8 CopyBlobsFromStorageAccountAToB

```
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_strin
g)

    # 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_containe
r_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):
```

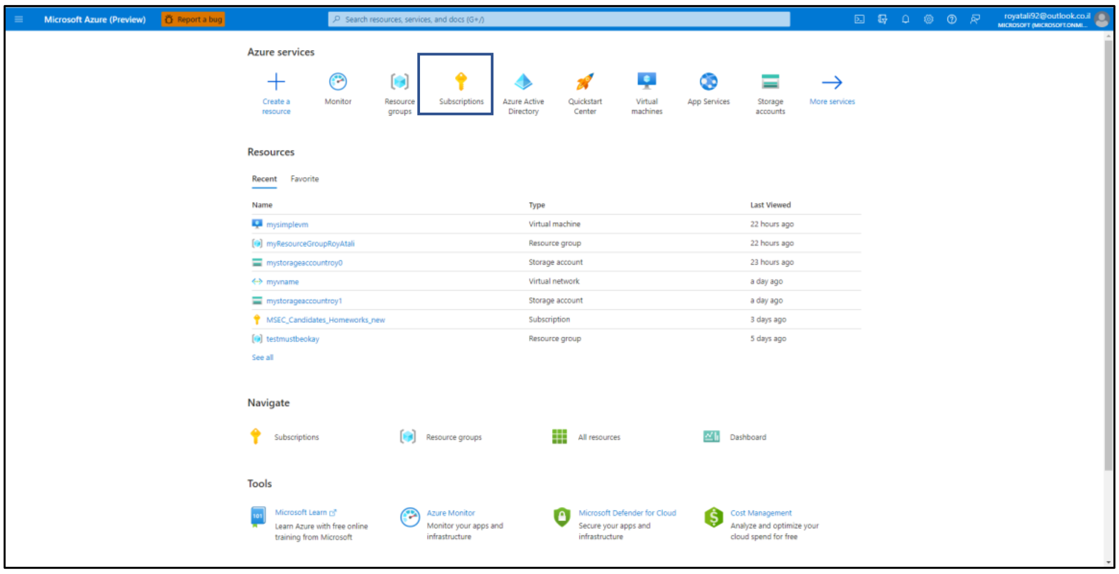
4. Parameters Obtaining

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**



Name	Type	Last Viewed
mysimplevm	Virtual machine	22 hours ago
myResourceGroupRoyAtali	Resource group	22 hours ago
mystorageaccountroy0	Storage account	23 hours ago
myvname	Virtual network	a day ago
mystorageaccountroy1	Storage account	a day ago
MSEC_Candidates_Homeworks_new	Subscription	3 days ago
testmustbeokay	Resource group	5 days ago

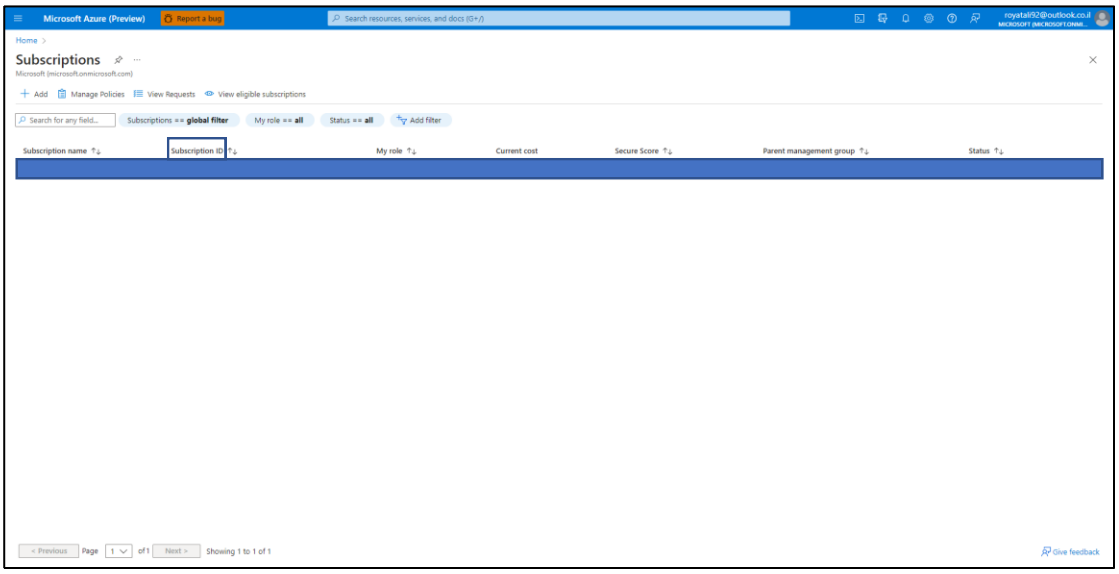
Microsoft Learn — Learn Azure with free online training from Microsoft

Azure Monitor — Monitor your apps and infrastructure

Microsoft Defender for Cloud — Secure your apps and infrastructure

Cost Management — Analyze and optimize your cloud spend for free

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



Subscription name	Subscription ID	My role	Current cost	Secure Score	Parent management group	Status
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

5. Automating the task copy_blobs

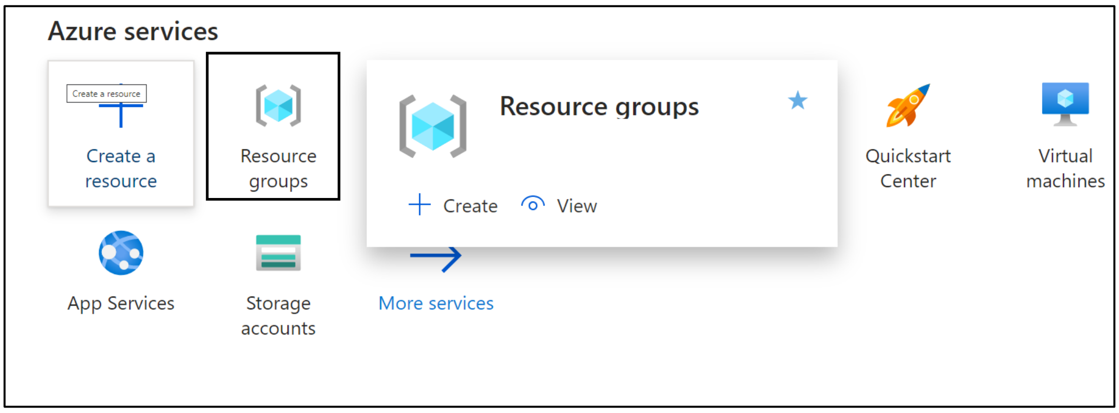
As part of the assignment the function copy_blobs that creates, uploads and copy 100 blobs from one storage account to another (source to destination) needs to run on the virtual machine (server) created previously and it needs to be automated.

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

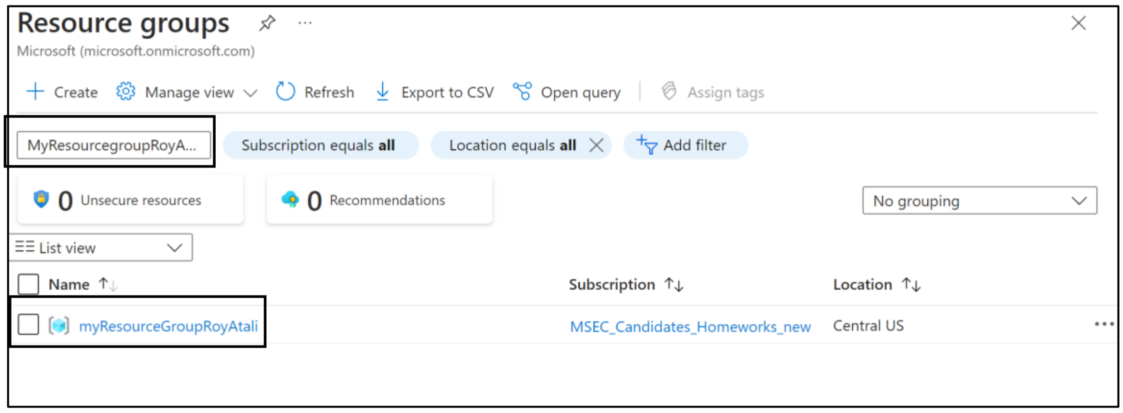
Here is how to do it:

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


Resource groups

Microsoft (microsoft.onmicrosoft.com)

MyResourcegroupRoyA...

Subscription equals **all**

Location equals **all**

0 Unsecure resources

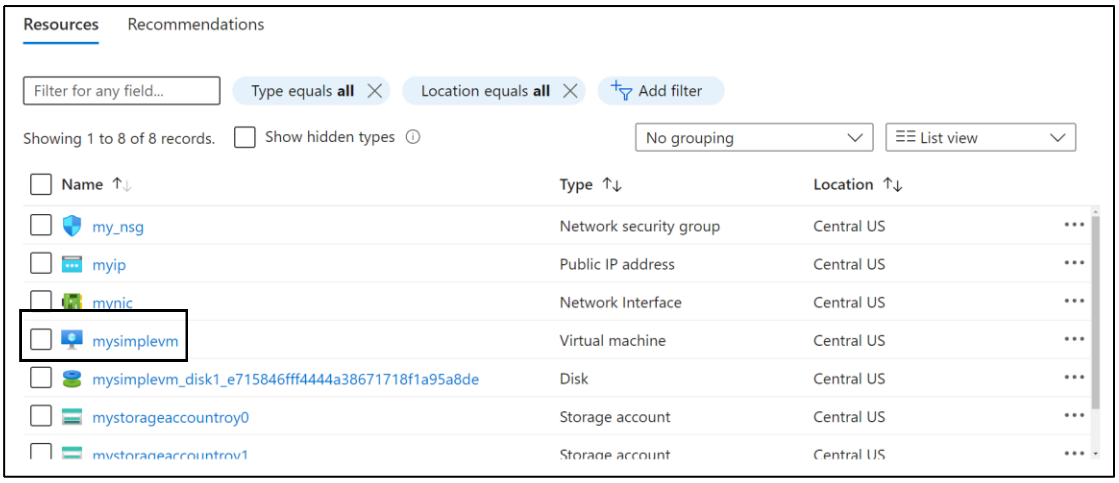
0 Recommendations

No grouping

List view

Name	Subscription	Location
myResourceGroupRoyAtali	MSEC_Candidates_Homeworks_new	Central US

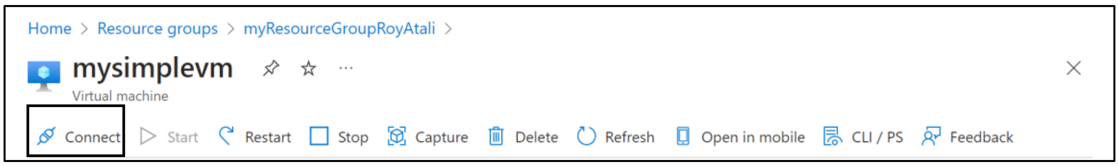
In the resources click on your virtual machine



Showing 1 to 8 of 8 records. Show hidden types

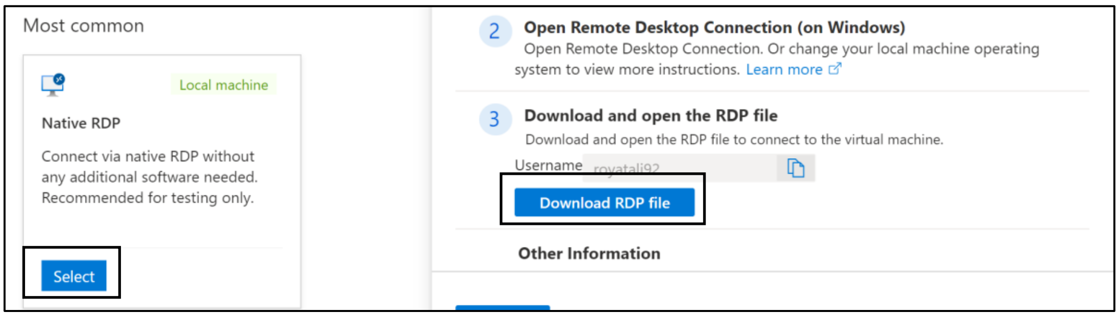
Name	Type	Location
my_nsg	Network security group	Central US
myip	Public IP address	Central US
mynic	Network Interface	Central US
mysimplevm	Virtual machine	Central US
mysimplevm_disk1_e715846fff4444a38671718f1a95a8de	Disk	Central US
mystorageaccountroy0	Storage account	Central US
mystorageaccountroy1	Storage account	Central US

Click on **Connect**



Click on **Select**

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



Native RDP

Connect via native RDP without any additional software needed. Recommended for testing only.

2 Open Remote Desktop Connection (on Windows)

Open Remote Desktop Connection. Or change your local machine operating system to view more instructions. Learn more

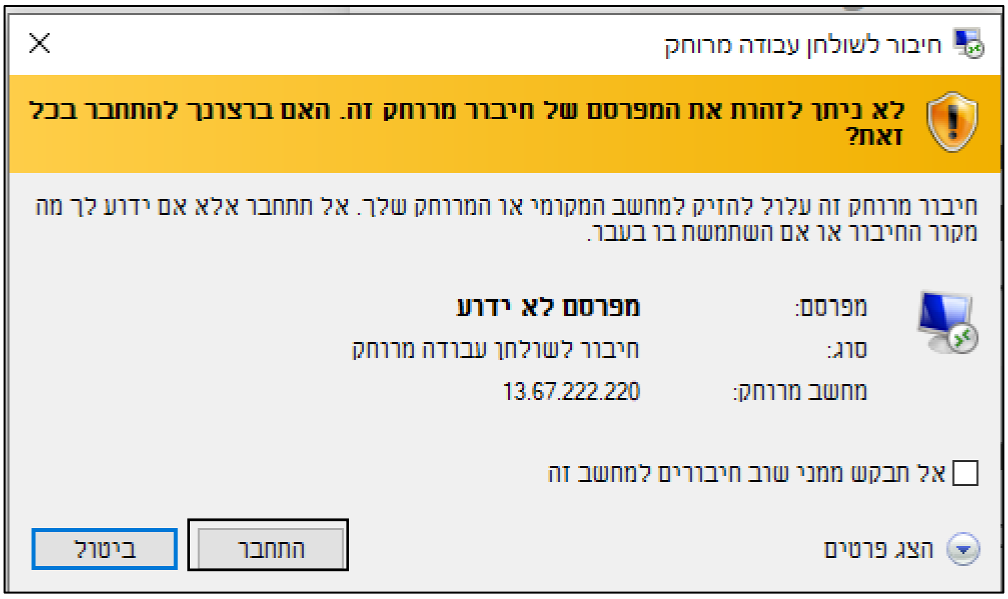
3 Download and open the RDP file

Download and open the RDP file to connect to the virtual machine.

Username royatali92

Other Information

Click on **התחבר/connect**



חיבור לשולחן עבודה מרוחק

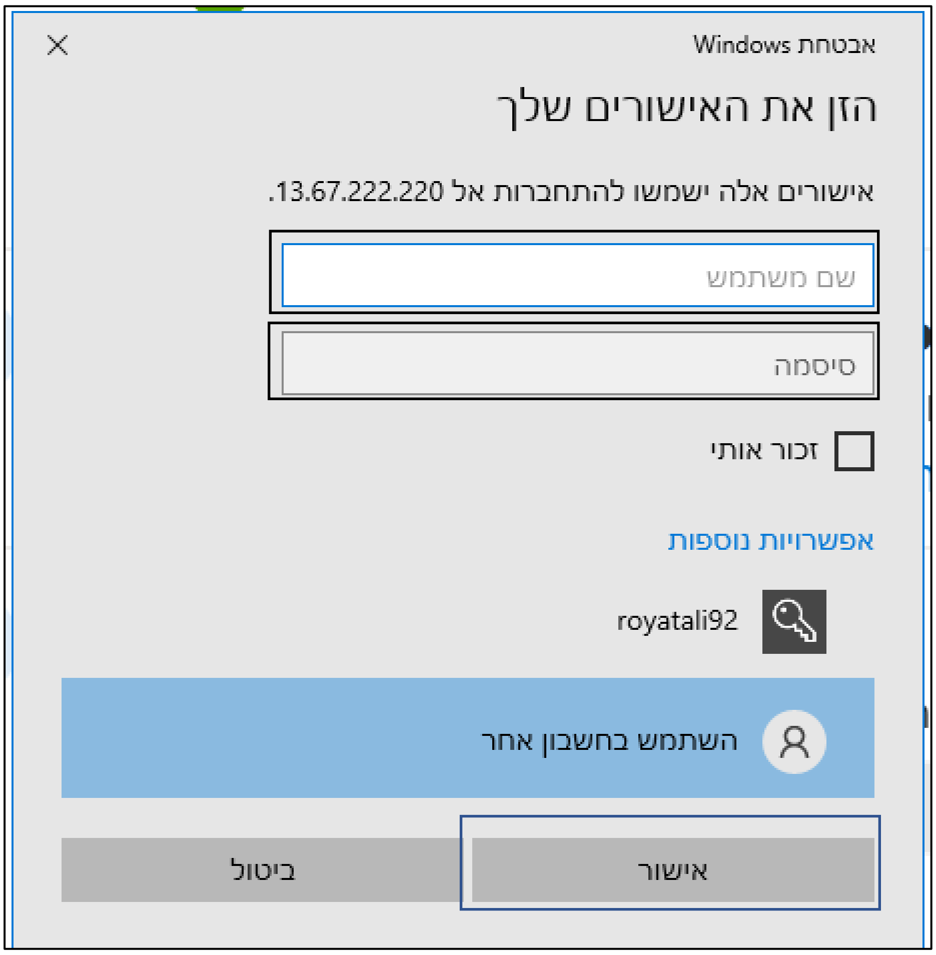
לא ניתן לזהות את המפרסם של חיבור מרוחק זה. האם ברצונך להתחבר בכל זאת?

חיבור מרוחק זה עלול להזיק למחשב המקומי או המרוחק שלך. אל תתחבר אלא אם ידוע לך מה מקור החיבור או אם השתמשת בו בעבר.

מפרסם:	מפרסם לא ידוע
סוג:	חיבור לשולחן עבודה מרוחק
מחשב מרוחק:	13.67.222.220

☐ אל תבקש ממני שוב חיבורים למחשב זה

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



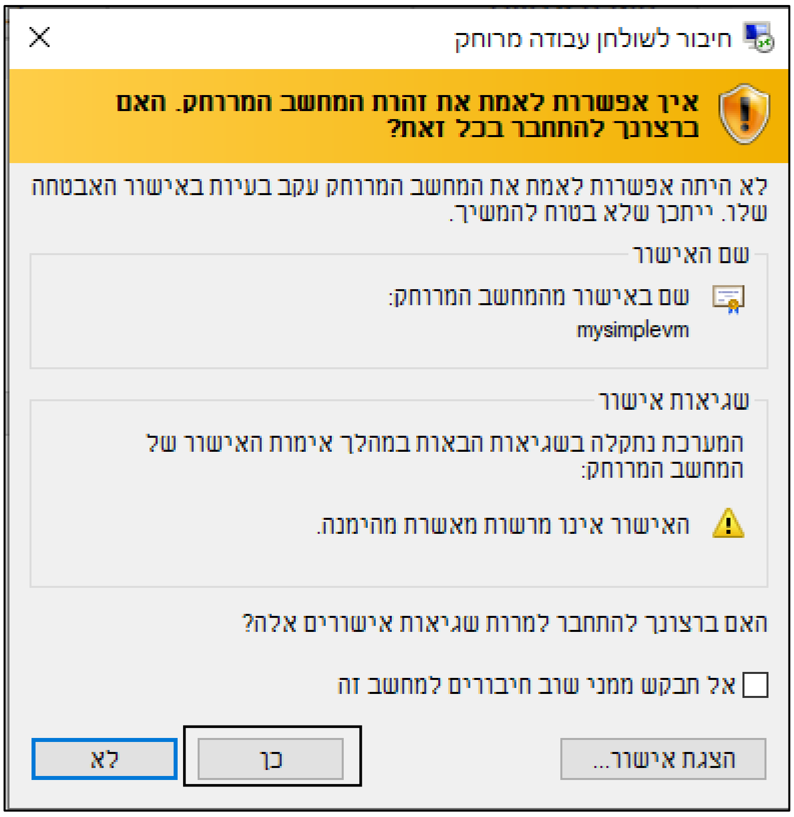
אבטחת Windows

הזן את האישורים שלך

אישורים אלה ישמשו להתחברות אל 13.67.222.220.

And click **אישור / OK**

click **כן / Yes**



חיבור לשולחן עבודה מרוחק

אין אפשרות לאמת את זהות המחשב המרוחק. האם ברצונך להתחבר בכל זאת?

לא היתה אפשרות לאמת את המחשב המרוחק עקב בעיות באישור האבטחה שלו. ייתכן שלא בטוח להמשיך.

שם האישור

שם באישור מהמחשב המרוחק:
mysimplevm

שגיאות אישור

המערכת נתקלה בשגיאות הבאות במהלך אימות האישור של המחשב המרוחק:

האישור אינו מרשות מאשרת מהימנה.

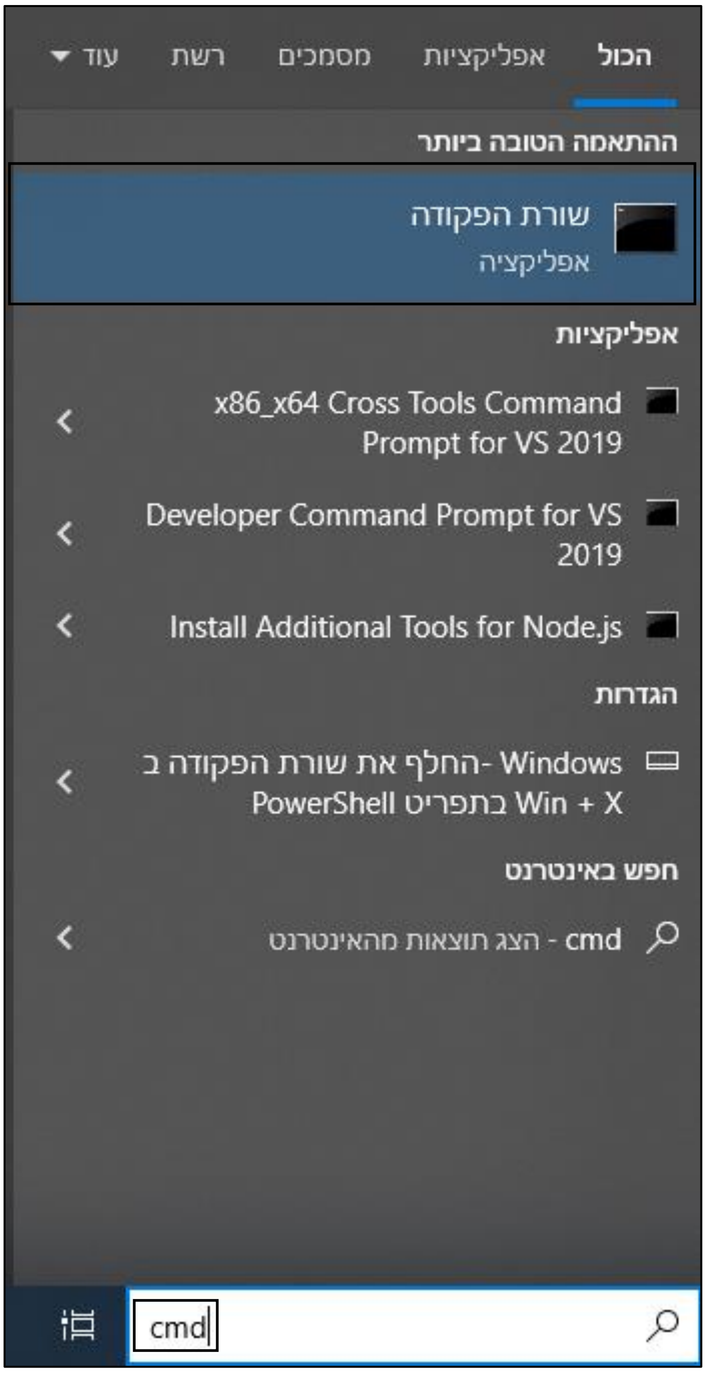
האם ברצונך להתחבר למרות שגיאות אישורים אלה?

☐ אל תבקש ממני שוב חיבורים למחשב זה

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.

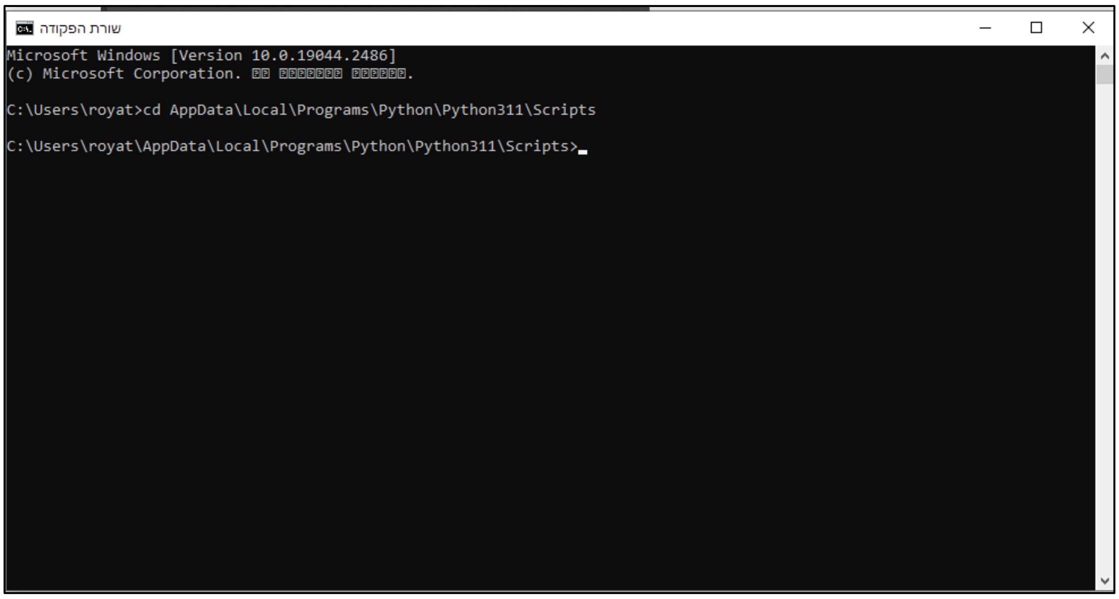
After installed go to the search bar on the Virtual Machine and type **cmd**.


הכול
אפליקציות
מסמכים
רשת
עוד
ההתאמה הטובה ביותר
שורת הפקודה
אפליקציה
אפליקציות
x86_x64 Cross Tools Command Prompt for VS 2019
Developer Command Prompt for VS 2019
Install Additional Tools for Node.js
הגדרות
החלף את שורת הפקודה ב- Windows PowerShell בתפריט Win + X
חפש באינטרנט
cmd - הצג תוצאות מהאינטרנט
cmd

In the command prompt type the command:

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

and press enter.

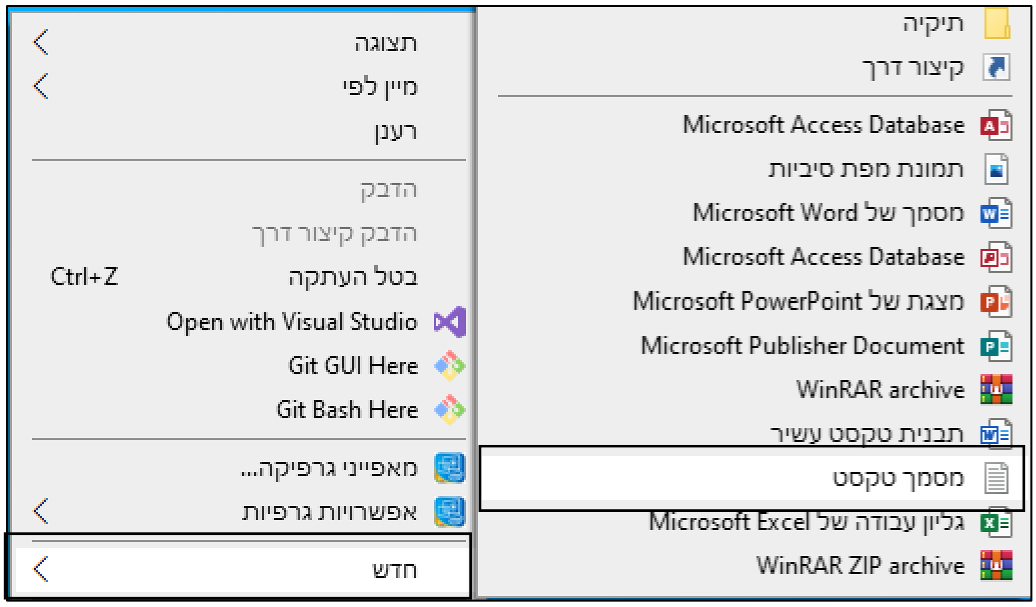


```
Microsoft Windows [Version 10.0.19044.2486]
(c) Microsoft Corporation. ?? ??????? ??????.

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

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

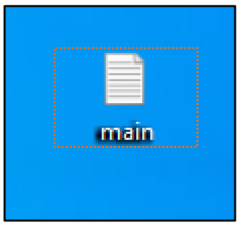
Then type these commands one by one and then press Enter after each one:

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->מסמך טקסט/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_na
me)


    # 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_na
me):

    # 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

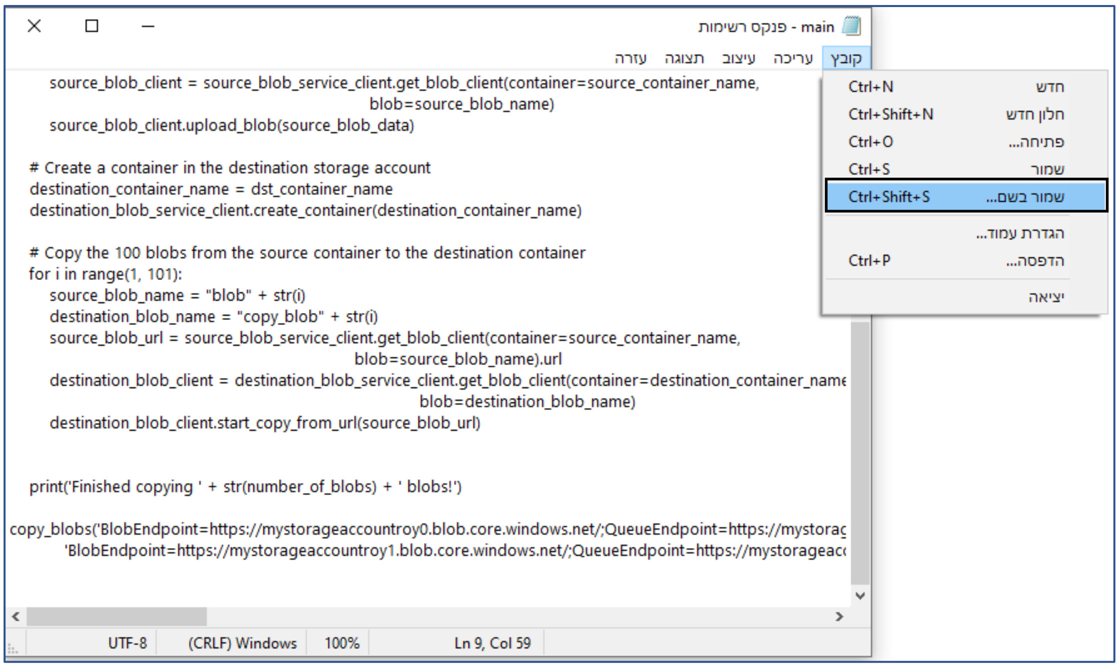
2. resource_group_name

3. src_container_name

4. dst_container_name

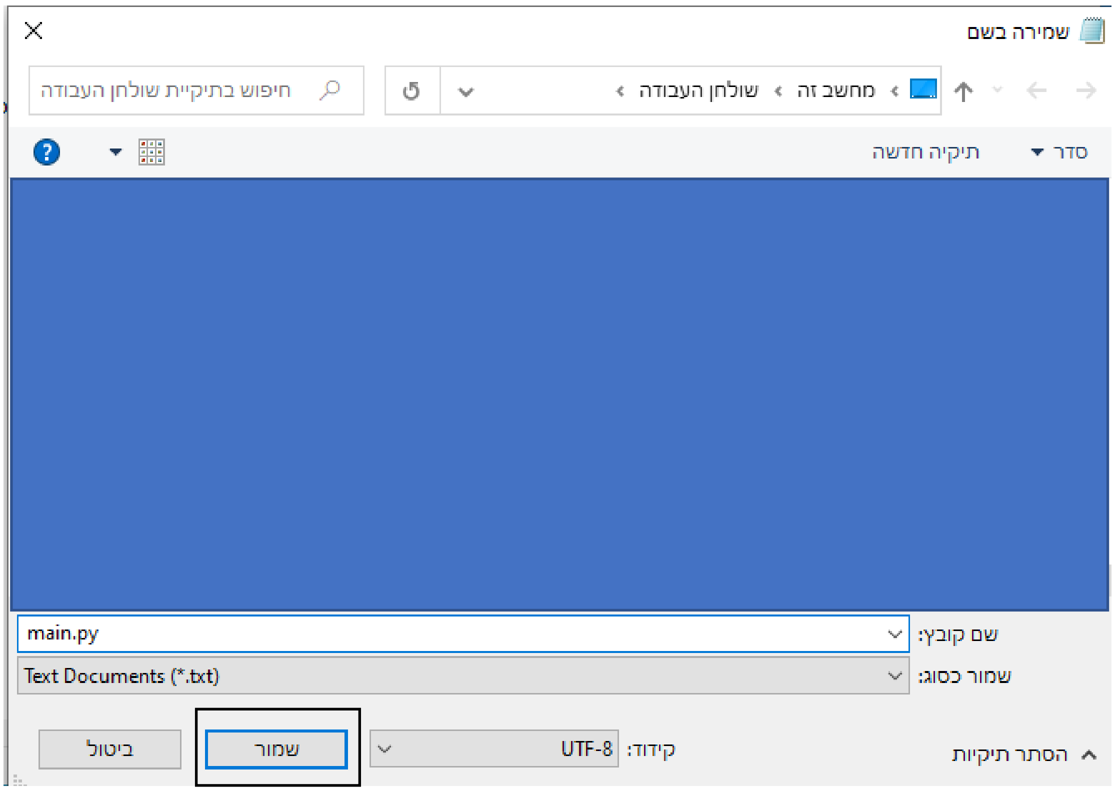
5. number_of_blobs

When everything is done click on קובץ/file->שמור בשם/save as



```
    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!')

copy_blobs('BlobEndpoint=https://mystorageaccountroy0.blob.core.windows.net/;QueueEndpoint=https://mystorag
       'BlobEndpoint=https://mystorageaccountroy1.blob.core.windows.net/;QueueEndpoint=https://mystorageacc
```

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

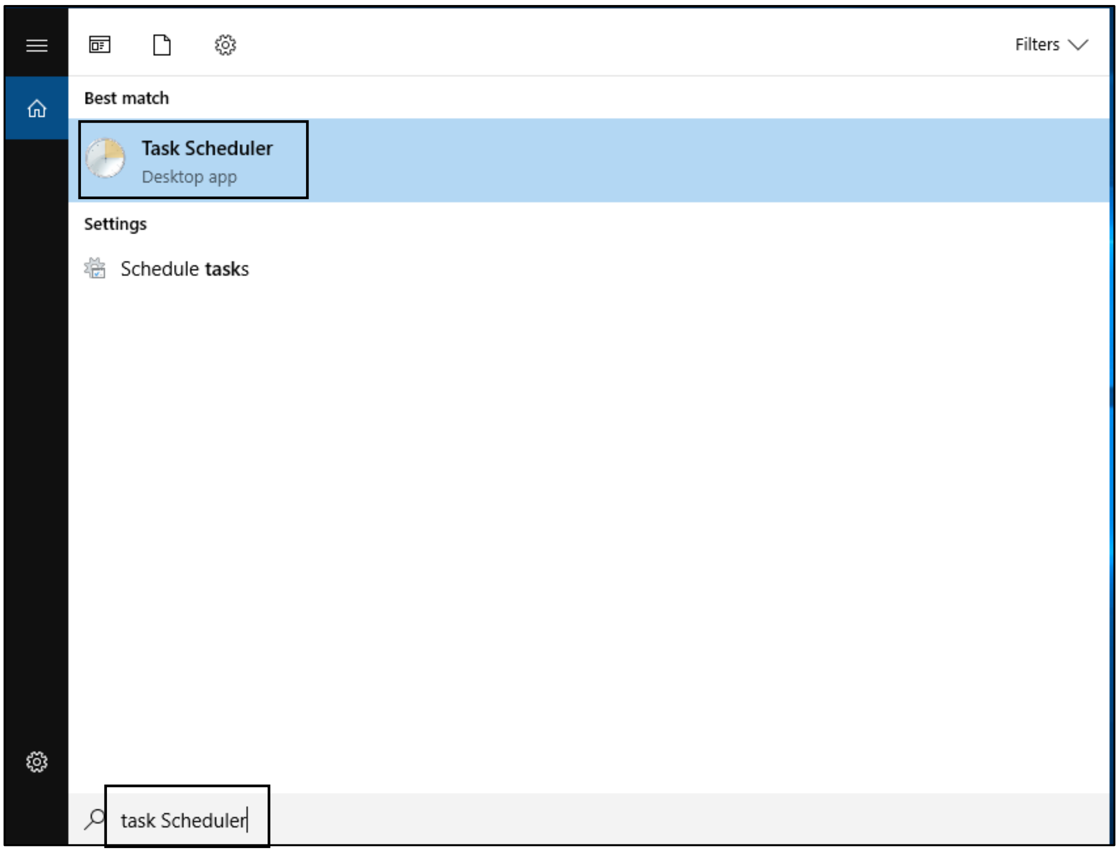


שם קובץ: main.py

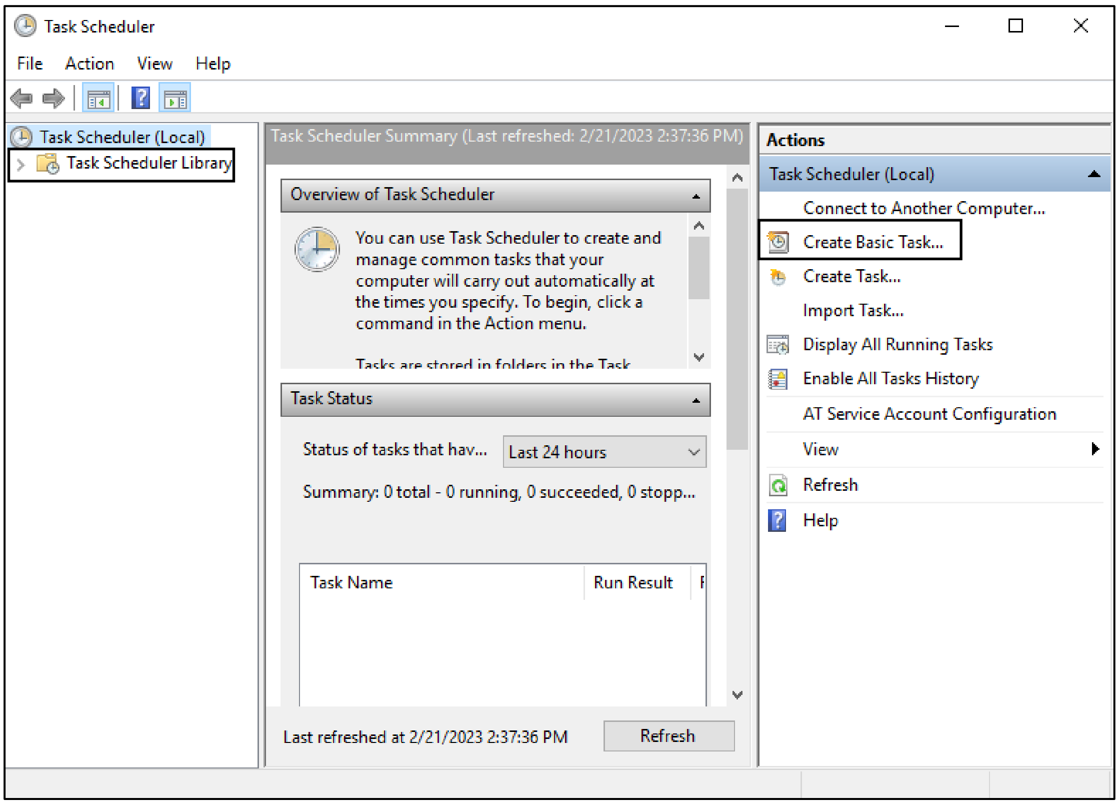
שמור כסוג: Text Documents (*.txt)

קידוד: UTF-8

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

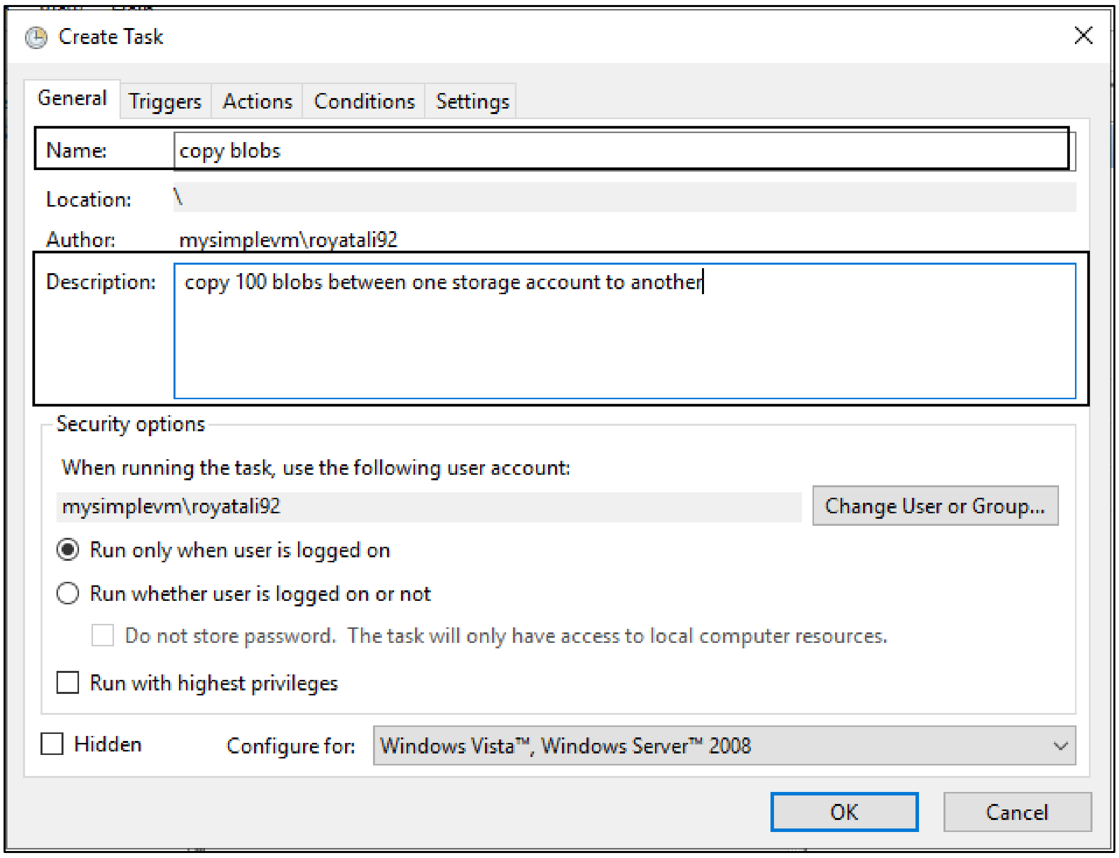

Task Scheduler
File Action View Help
Task Scheduler (Local)
Task Scheduler Library
Task Scheduler Summary (Last refreshed: 2/21/2023 2:37:36 PM)
Overview of Task Scheduler

You can use Task Scheduler to create and manage common tasks that your computer will carry out automatically at the times you specify. To begin, click a command in the Action menu.

Tasks are stored in folders in the Task

Task Status
Status of tasks that hav...
Last 24 hours
Summary: 0 total - 0 running, 0 succeeded, 0 stopp...
Task Name
Run Result
Last refreshed at 2/21/2023 2:37:36 PM
Refresh
Actions
Task Scheduler (Local)
Connect to Another Computer...
Create Basic Task...
Create Task...
Import Task...
Display All Running Tasks
Enable All Tasks History
AT Service Account Configuration
View
Refresh
Help

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


Create Task

General | Triggers | Actions | Conditions | Settings

Name: copy blobs

Location: \

Author: mysimplevm\royatali92

Description: copy 100 blobs between one storage account to another

Security options

When running the task, use the following user account:

mysimplevm\royatali92

[✓ Run only when user is logged on]

Run whether user is logged on or not

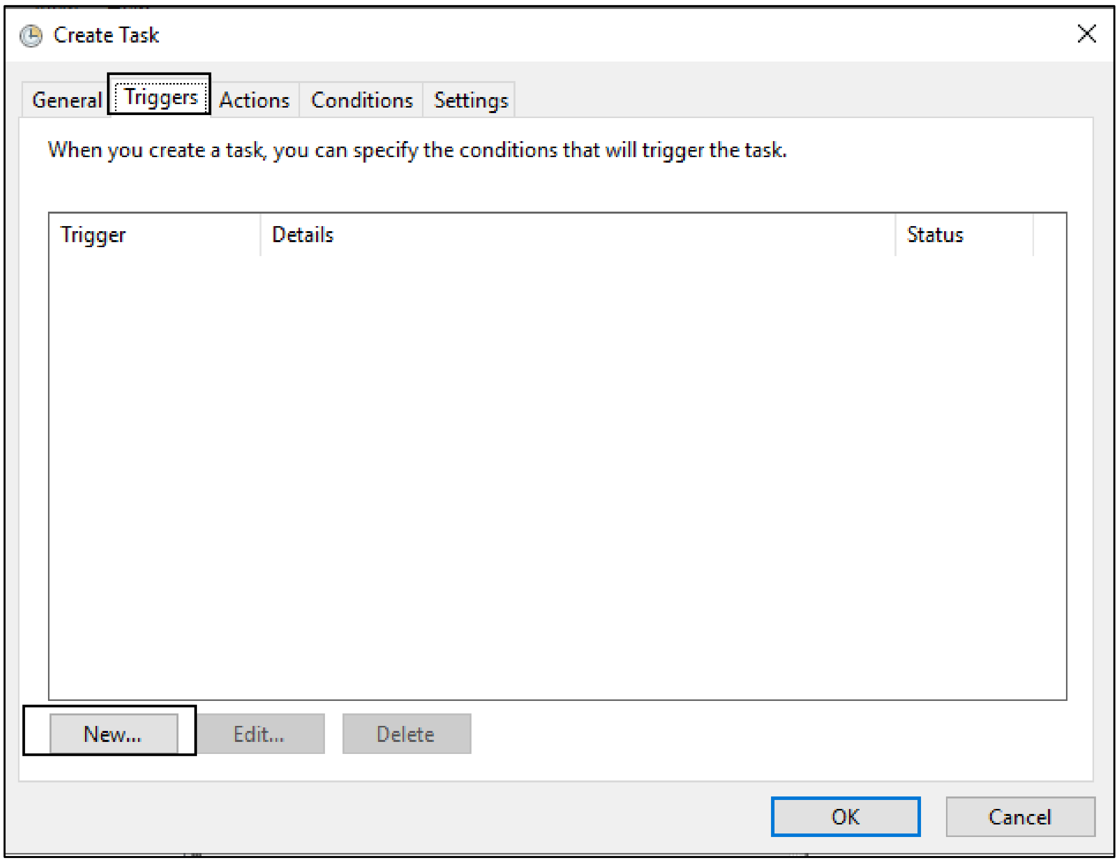
Do not store password. The task will only have access to local computer resources.

Run with highest privileges

Hidden

Configure for: Windows Vista™, Windows Server™ 2008

Move to Triggers Tab and select New...

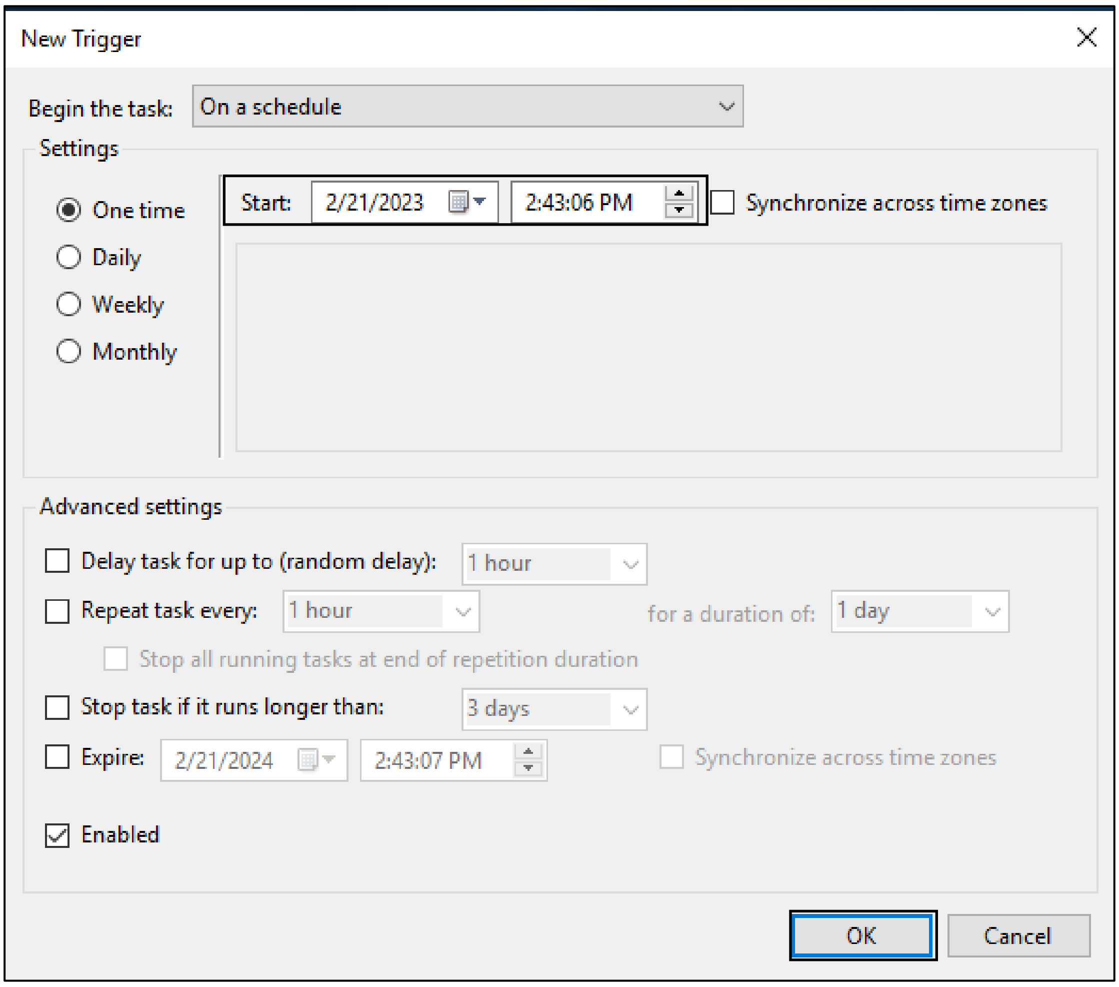

Create Task

General | Triggers | Actions | Conditions | Settings

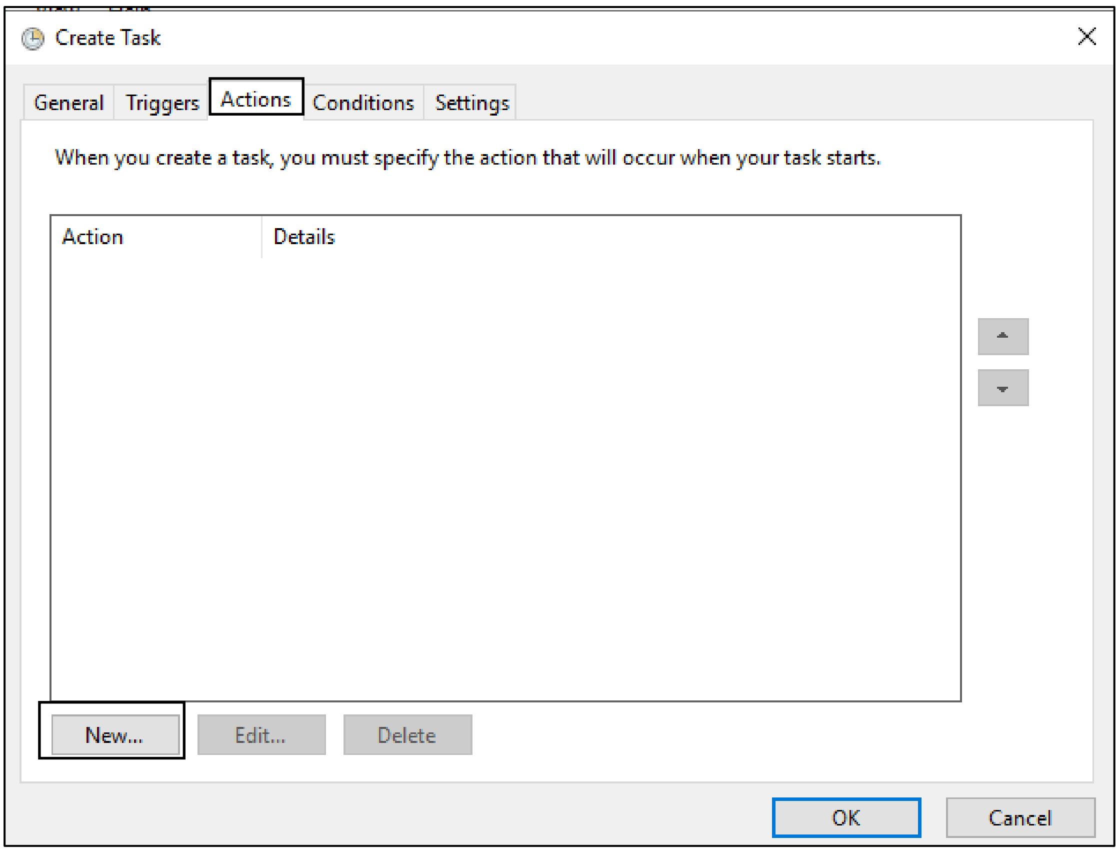
When you create a task, you can specify the conditions that will trigger the task.

Trigger	Details	Status

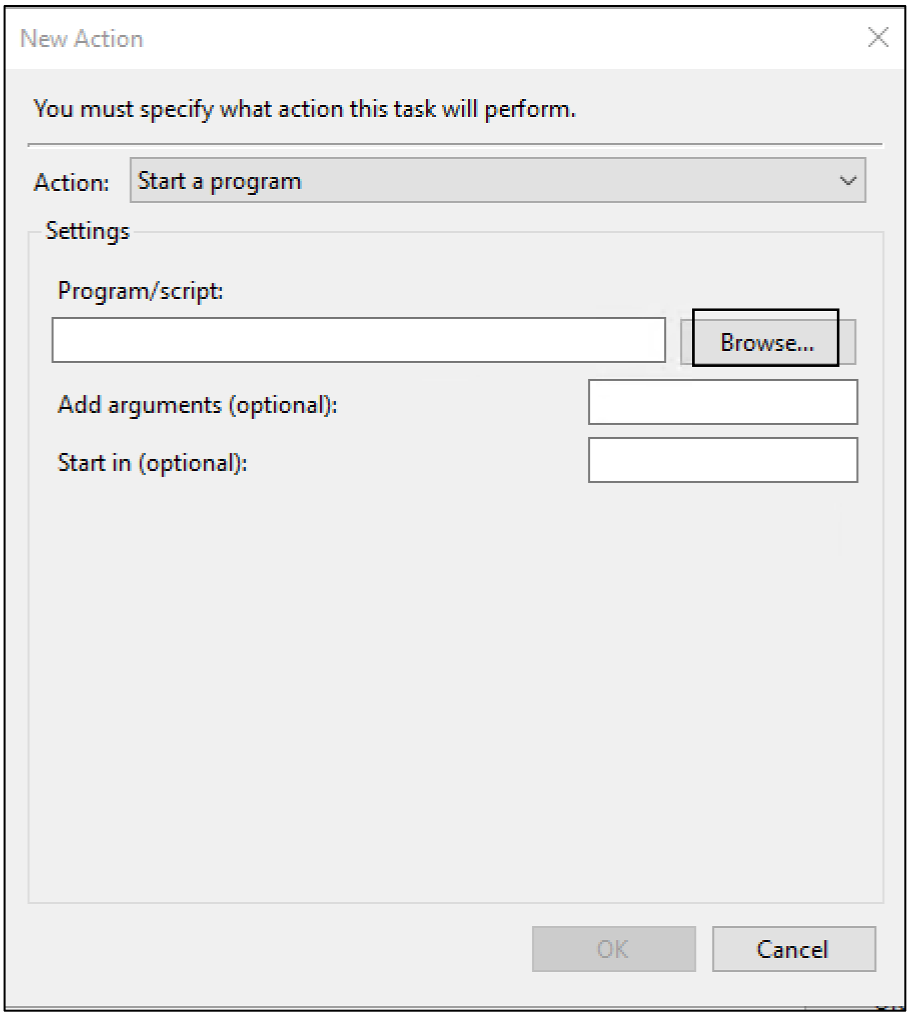
In the New Trigger Window choose the appropriate start date/time to run the script and click OK


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



You must specify what action this task will perform.

Action: Start a program

Settings

Program/script:

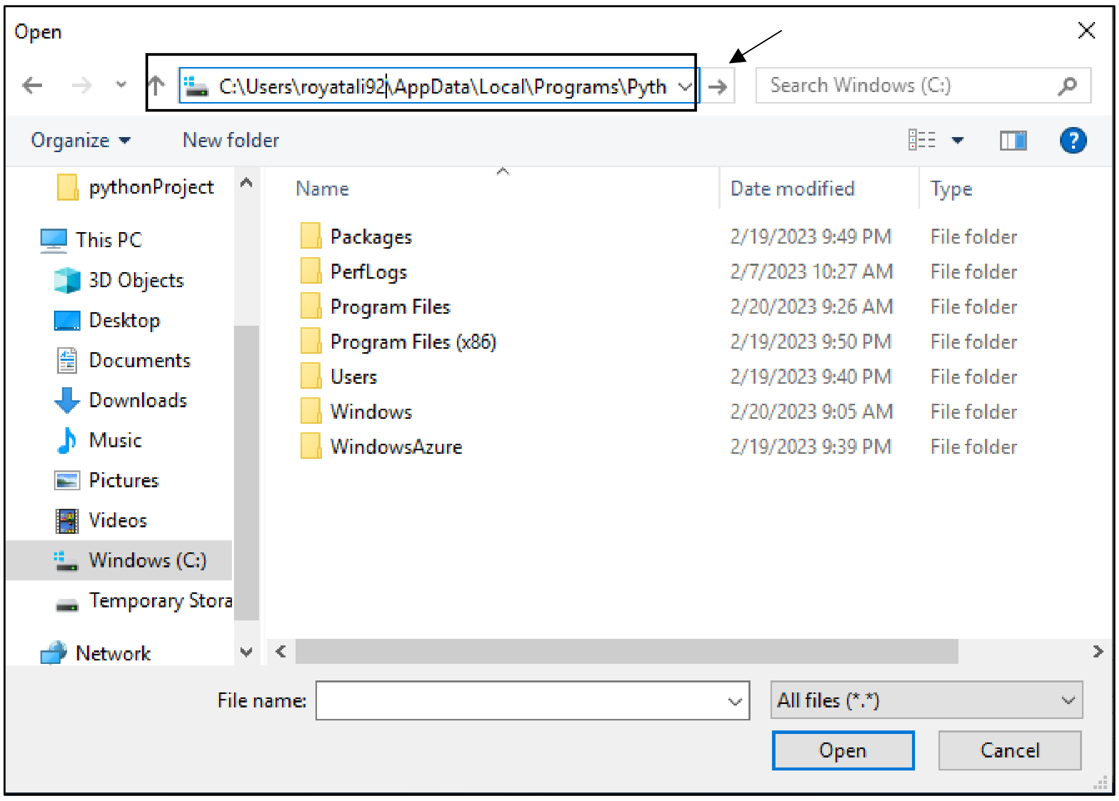
Add arguments (optional):

Start in (optional):

In the browser enter this path:

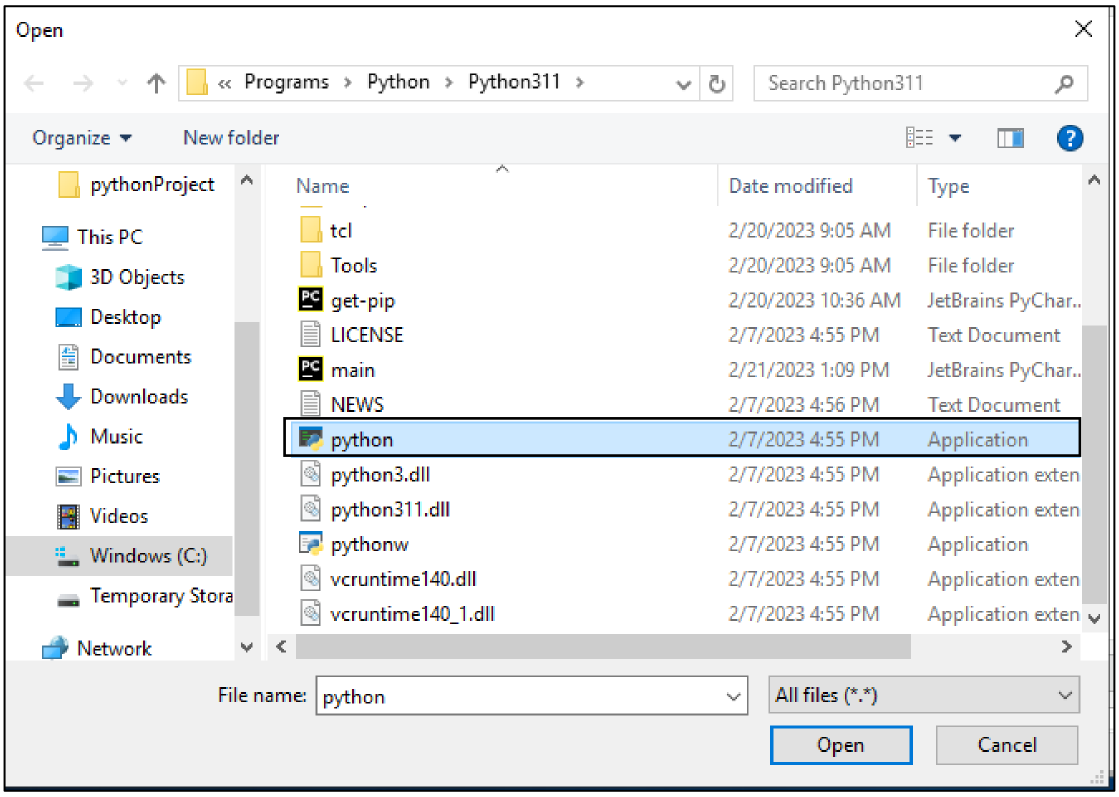
C:\Users\yourUserName\AppData\Local\Programs\Python\Python311\

And press the arrow button in the picture below



Name	Date modified	Type
Packages	2/19/2023 9:49 PM	File folder
PerfLogs	2/7/2023 10:27 AM	File folder
Program Files	2/20/2023 9:26 AM	File folder
Program Files (x86)	2/19/2023 9:50 PM	File folder
Users	2/19/2023 9:40 PM	File folder
Windows	2/20/2023 9:05 AM	File folder
WindowsAzure	2/19/2023 9:39 PM	File folder

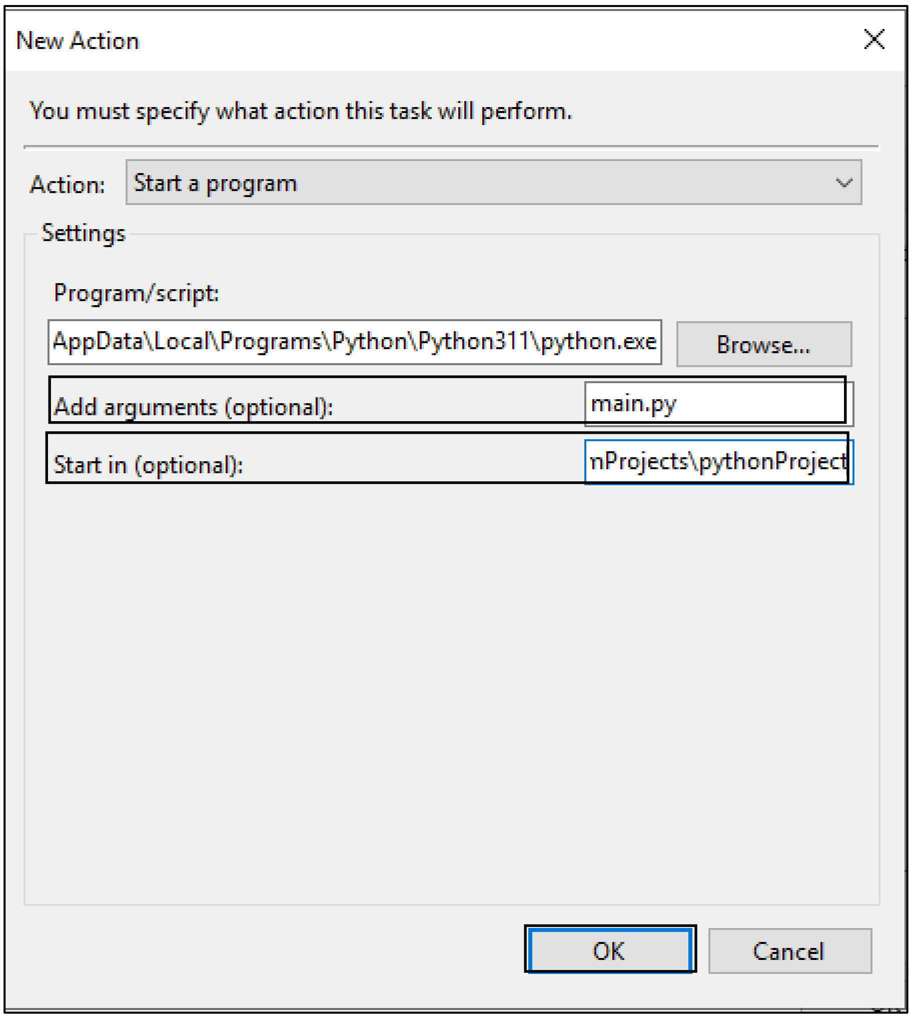
Choose python from the list and press Open



Name	Date modified	Type
tcl	2/20/2023 9:05 AM	File folder
Tools	2/20/2023 9:05 AM	File folder
get-pip	2/20/2023 10:36 AM	JetBrains PyChar..
LICENSE	2/7/2023 4:55 PM	Text Document
main	2/21/2023 1:09 PM	JetBrains PyChar..
NEWS	2/7/2023 4:56 PM	Text Document
python	2/7/2023 4:55 PM	Application
python3.dll	2/7/2023 4:55 PM	Application exten
python311.dll	2/7/2023 4:55 PM	Application exten
pythonw	2/7/2023 4:55 PM	Application
vcruntime140.dll	2/7/2023 4:55 PM	Application exten
vcruntime140_1.dll	2/7/2023 4:55 PM	Application exten

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


New Action

You must specify what action this task will perform.

Action: Start a program

Settings

Program/script: AppData\Local\Programs\Python\Python311\python.exe

Add arguments (optional): main.py

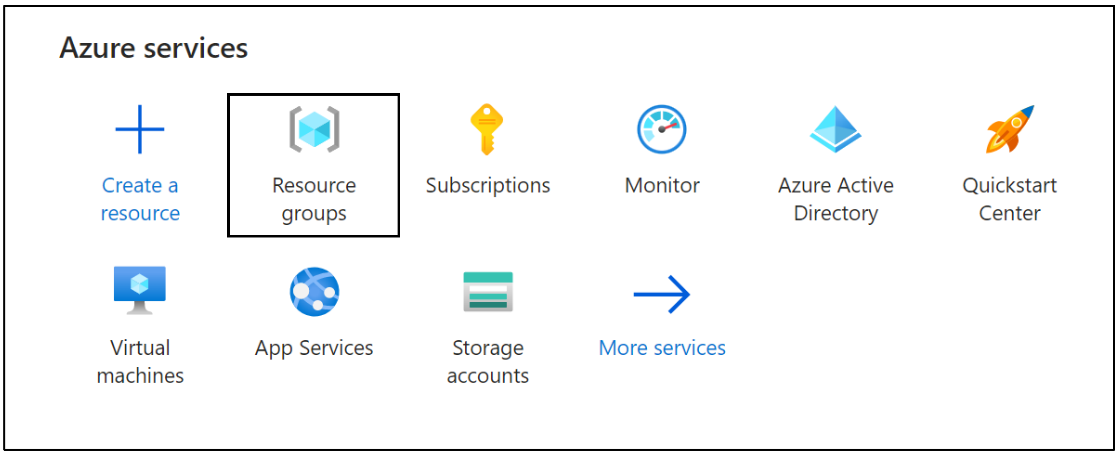
Start in (optional): nProjects\pythonProject

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

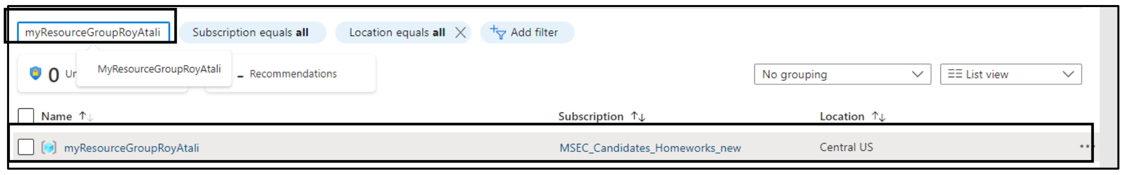
You can see the blobs in the source and destination containers at the Azure Portal

Go to the link it will open the Azure portal.

Click on Resource Groups

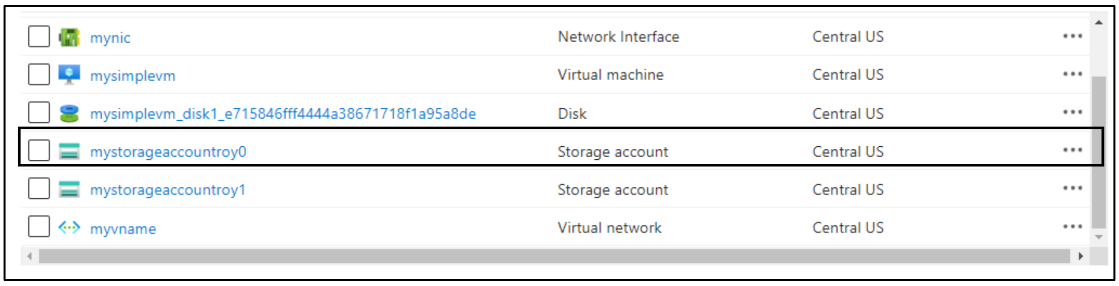


In the search bar type your resource group and click on it



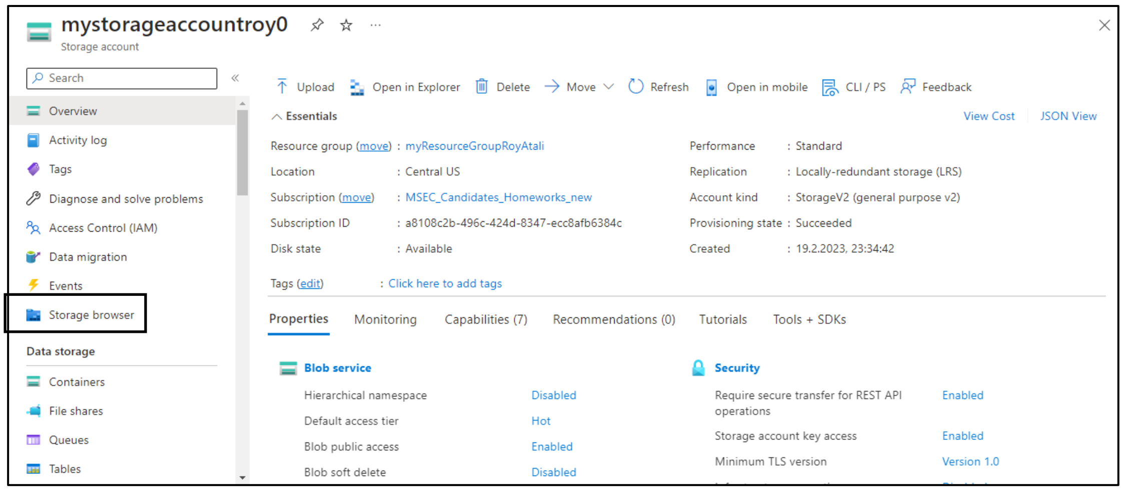
Name	Subscription	Location
myResourceGroupRoyAtali	MSEC_Candidates_Homeworks_new	Central US

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



mynic	Network Interface	Central US
mysimplevm	Virtual machine	Central US
mysimplevm_disk1_e715846fff4444a38671718f1a95a8de	Disk	Central US
mystorageaccountroy0	Storage account	Central US
mystorageaccountroy1	Storage account	Central US
myvname	Virtual network	Central US

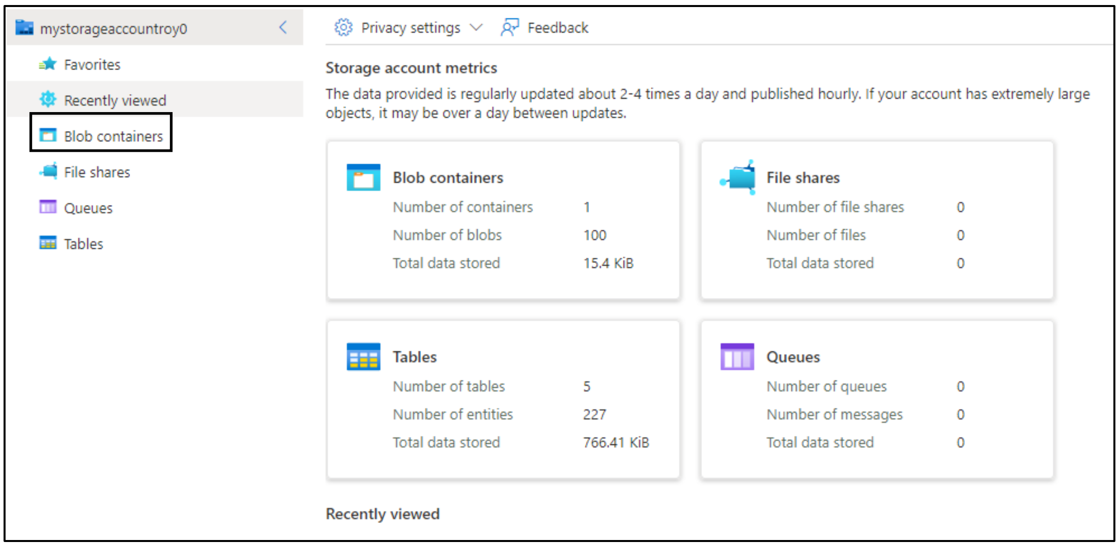
On the side menu choose Storage Browser



Resource group (move)	: myResourceGroupRoyAtali	Performance	: Standard
Location	: Central US	Replication	: Locally-redundant storage (LRS)
Subscription (move)	: MSEC_Candidates_Homeworks_new	Account kind	: StorageV2 (general purpose v2)
Subscription ID	: a8108c2b-496c-424d-8347-ecc8afb6384c	Provisioning state	: Succeeded
Disk state	: Available	Created	: 19.2.2023, 23:34:42
Tags (edit)	: Click here to add tags		

Blob service		Security	
Hierarchical namespace	Disabled	Require secure transfer for REST API operations	Enabled
Default access tier	Hot	Storage account key access	Enabled
Blob public access	Enabled	Minimum TLS version	Version 1.0
Blob soft delete	Disabled		

And then Blob Containers



mystorageaccountroy0

- Favorites
- Recently viewed
- Blob containers
- File shares
- Queues
- Tables

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

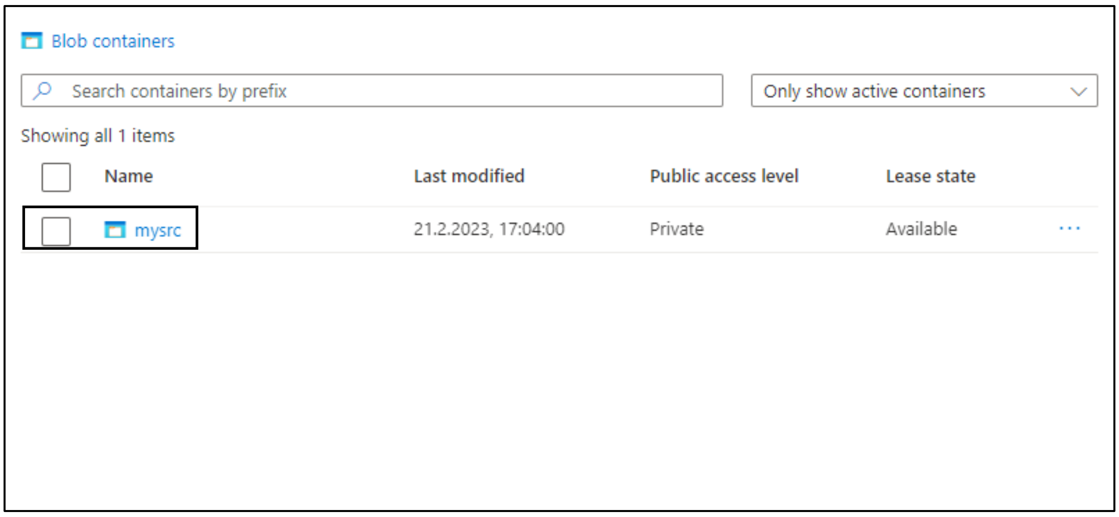
File shares	
Number of file shares	0
Number of files	0
Total data stored	0

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



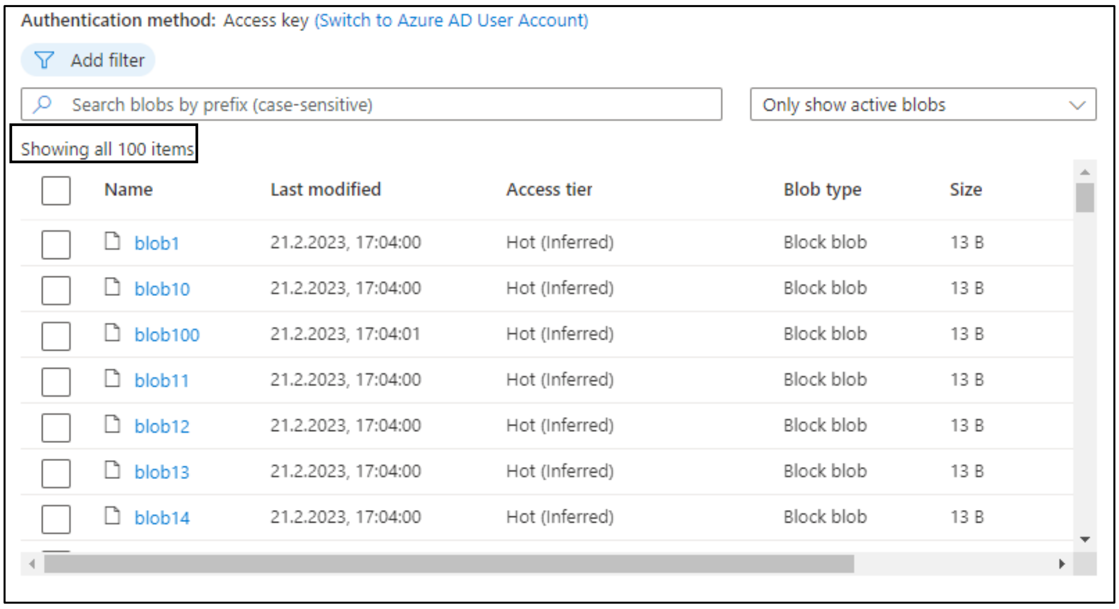
Blob containers

Search containers by prefix · Only show active containers

Showing all 1 items

Name	Last modified	Public access level	Lease state
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)

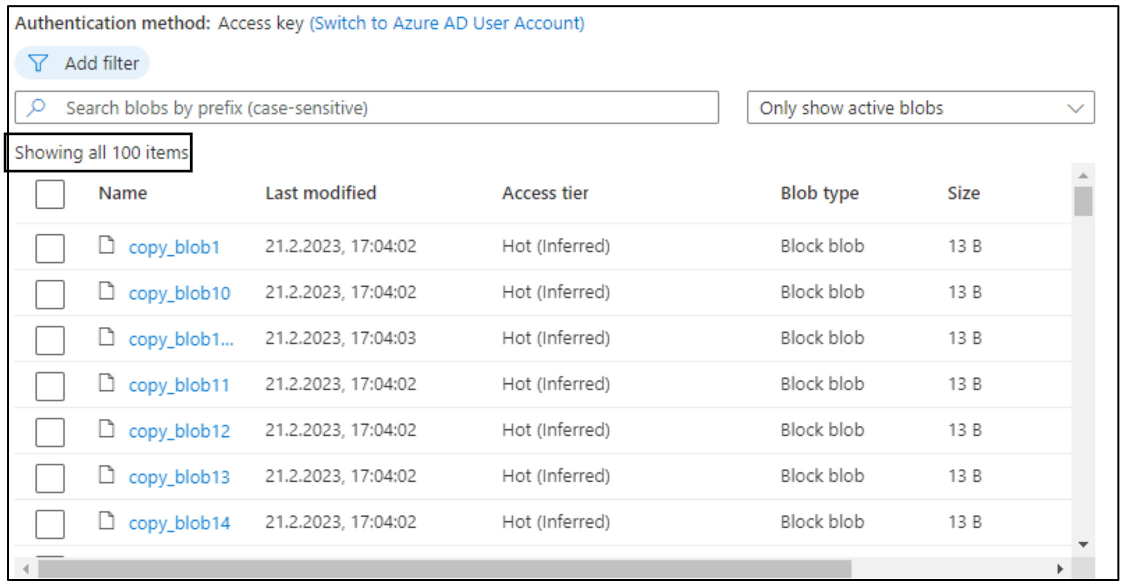
Add filter

Search blobs by prefix (case-sensitive) · Only show active blobs

Showing all 100 items

Name	Last modified	Access tier	Blob type	Size
blob1	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
blob10	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
blob100	21.2.2023, 17:04:01	Hot (Inferred)	Block blob	13 B
blob11	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
blob12	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
blob13	21.2.2023, 17:04:00	Hot (Inferred)	Block blob	13 B
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

Search blobs by prefix (case-sensitive) | Only show active blobs

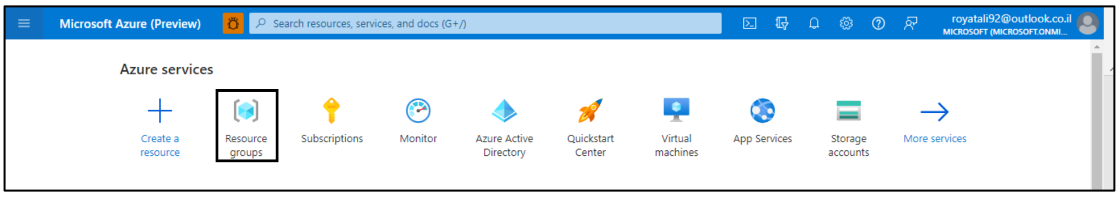
Showing all 100 items

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

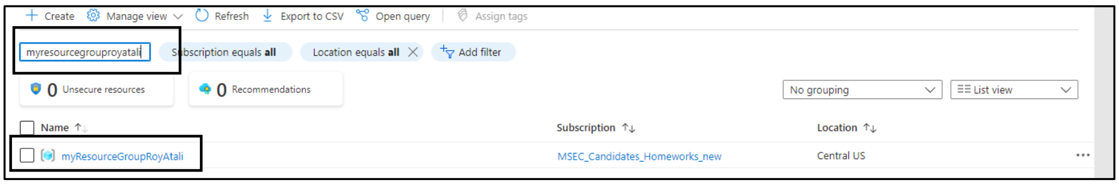
6. Monitoring

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

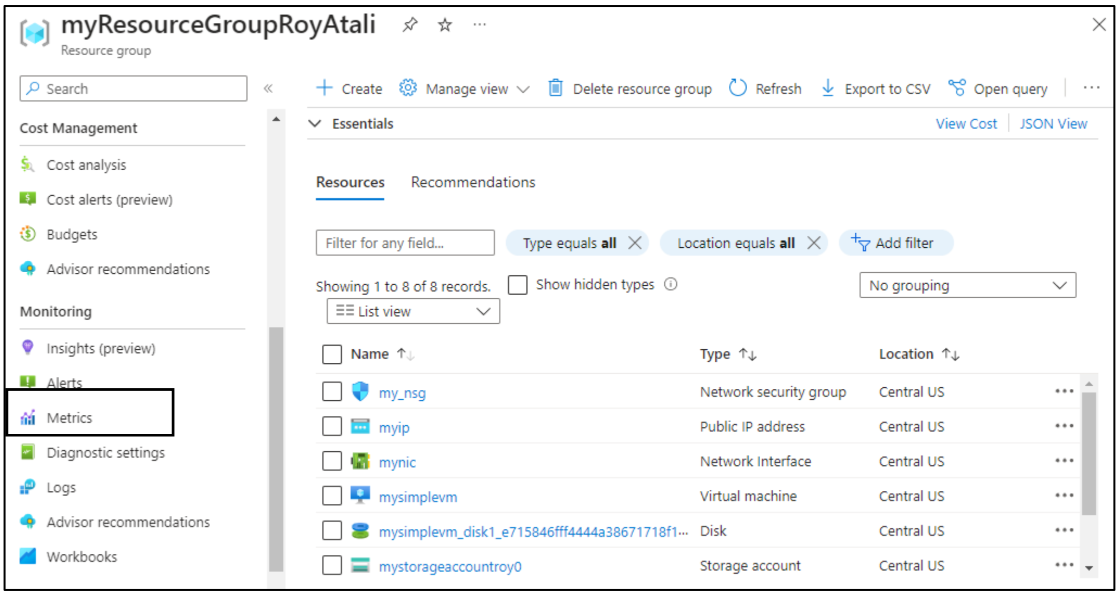


myresourcegrouproyatali | Subscription equals **all** | Location equals **all** | Add filter

0 Unsecure resources | 0 Recommendations | No grouping | List view

Name	Subscription	Location
myResourceGroupRoyAtali	MSEC_Candidates_Homeworks_new	Central US

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

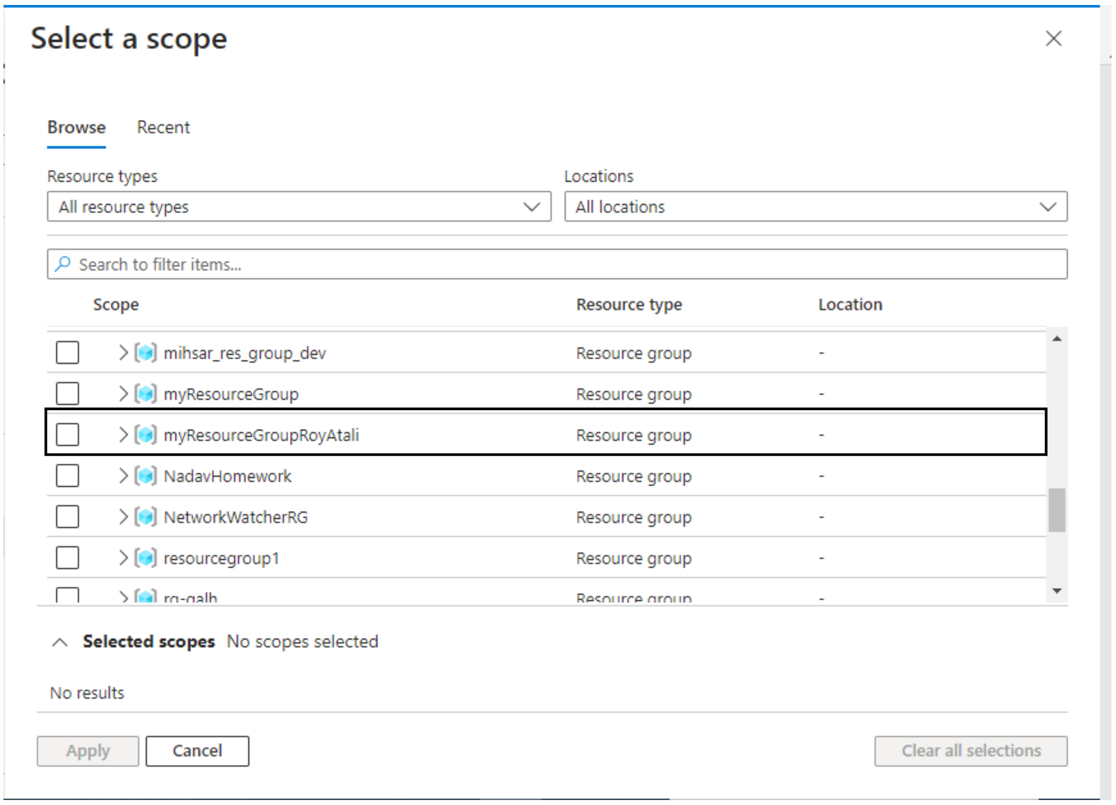


myResourceGroupRoyAtali
Resource group

Showing 1 to 8 of 8 records.

Name	Type	Location
my_nsg	Network security group	Central US
myip	Public IP address	Central US
mynic	Network Interface	Central US
mysimplevm	Virtual machine	Central US
mysimplevm_disk1_e715846fff4444a38671718f1...	Disk	Central US
mystorageaccountroy0	Storage account	Central US

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



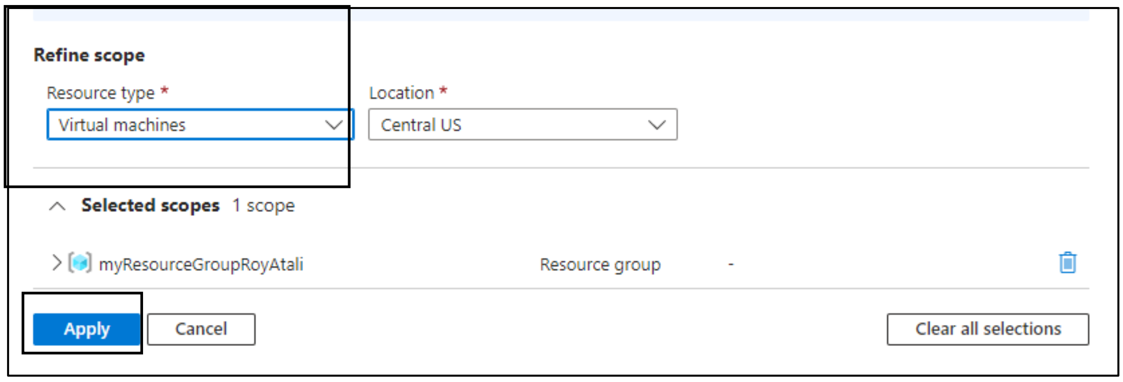
Select a scope

Scope	Resource type	Location
mihsar_res_group_dev	Resource group	-
myResourceGroup	Resource group	-
myResourceGroupRoyAtali	Resource group	-
NadavHomework	Resource group	-
NetworkWatcherRG	Resource group	-
resourcegroup1	Resource group	-
rg-galb	Resource group	-

Selected scopes No scopes selected

No results

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

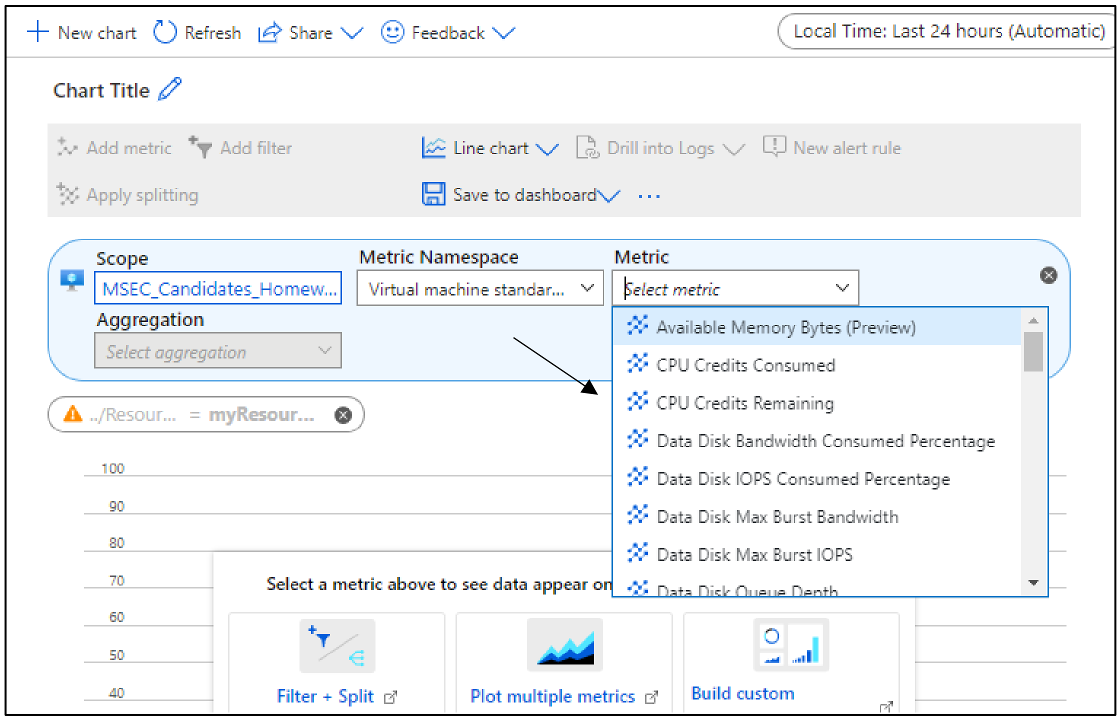

New chart Refresh Share Feedback Local Time: Last 24 hours (Automatic)

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

- 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

../Resour... = myResour...

Select a metric above to see data appear on

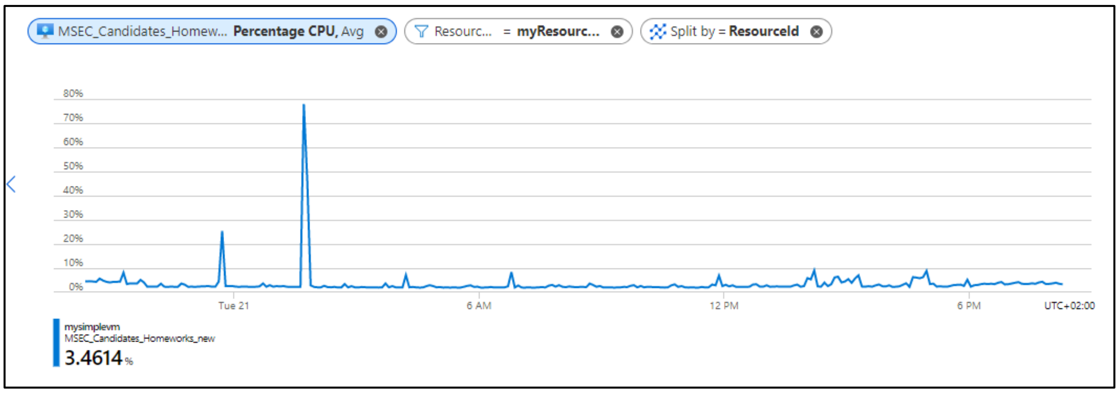
Filter + Split Plot multiple metrics Build custom

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

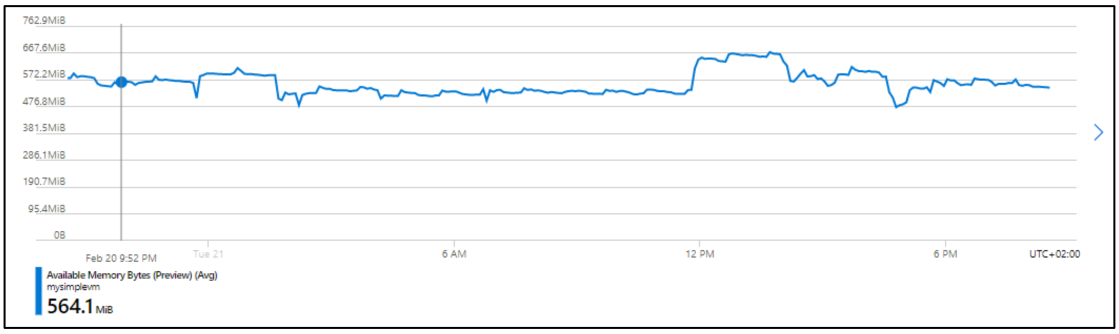
1. **Server CPU utilization:** Monitor the percentage of CPU usage on the server. High CPU utilization could indicate performance issues or resource constraints.
2. **Server Available Memory:** This metric represents the amount of memory available for use by applications and processes. You can monitor 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

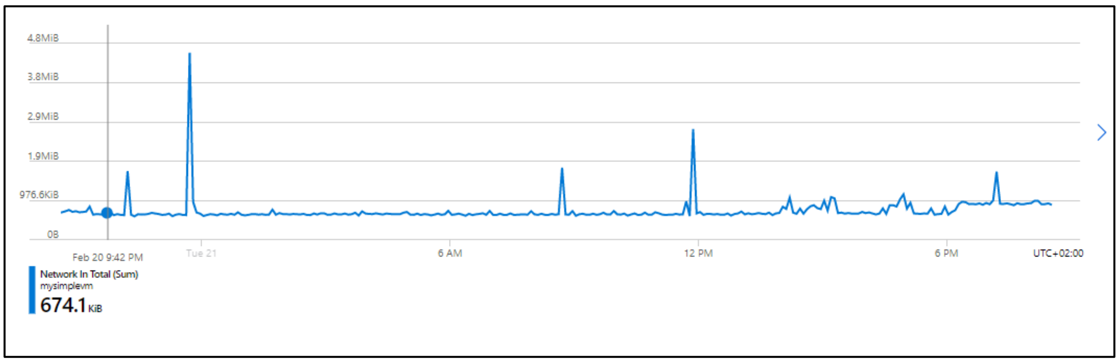


6.1.2 Server Available Memory

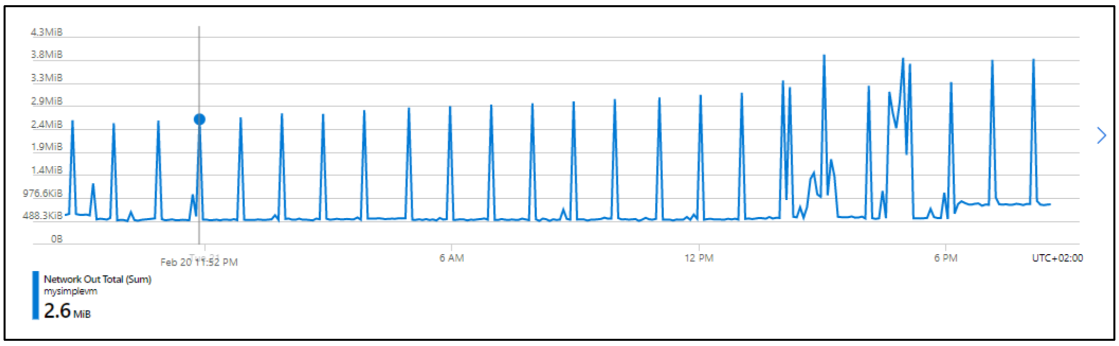


6.1.3 Server Network Traffic

6.1.3.1 Inbound Traffic



6.1.3.2 Outbound Traffic

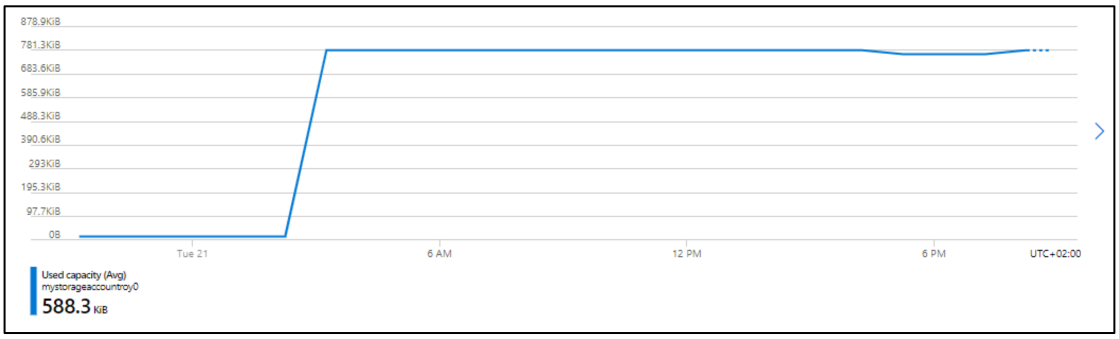


The different metrics for the storage accounts I chose to focus on are:

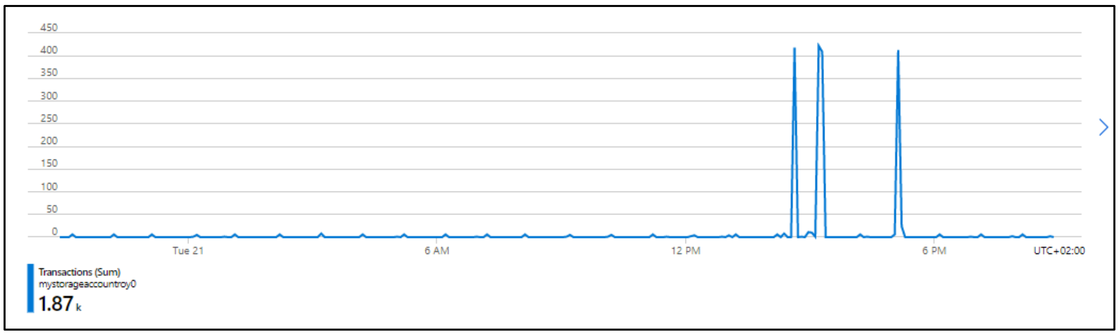
1. **Used capacity:** Used capacity refers to the amount of storage space that is being used by the data stored in your storage account. By monitoring used capacity, you can identify when you are approaching the storage limit of your account and take action to ensure that you have sufficient storage 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

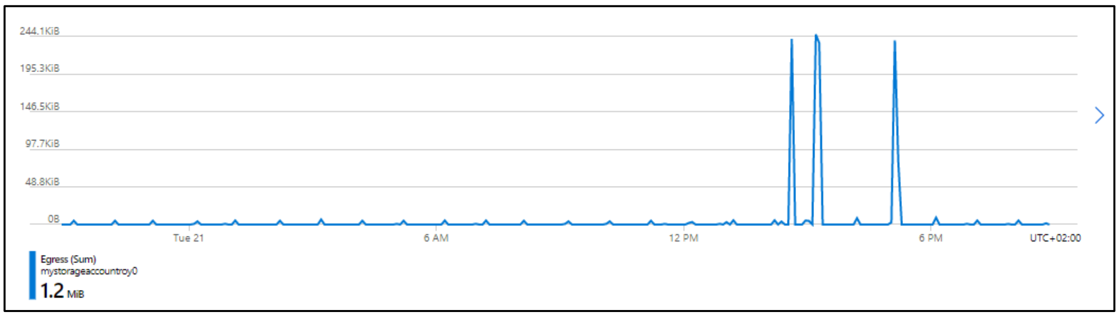
6.2.1 Used capacity



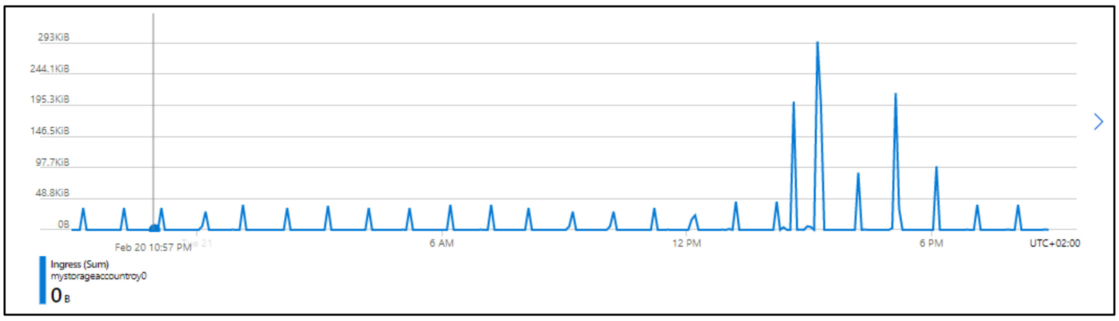
6.2.2 Transactions



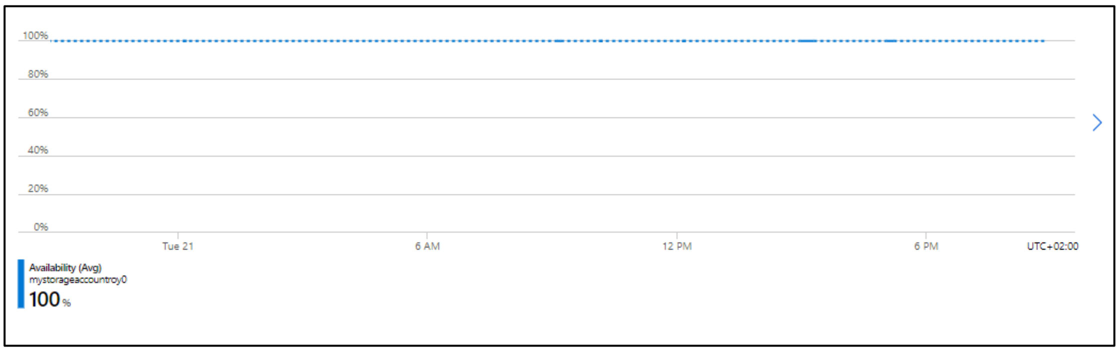
6.2.3 Egress



6.2.4 Ingress

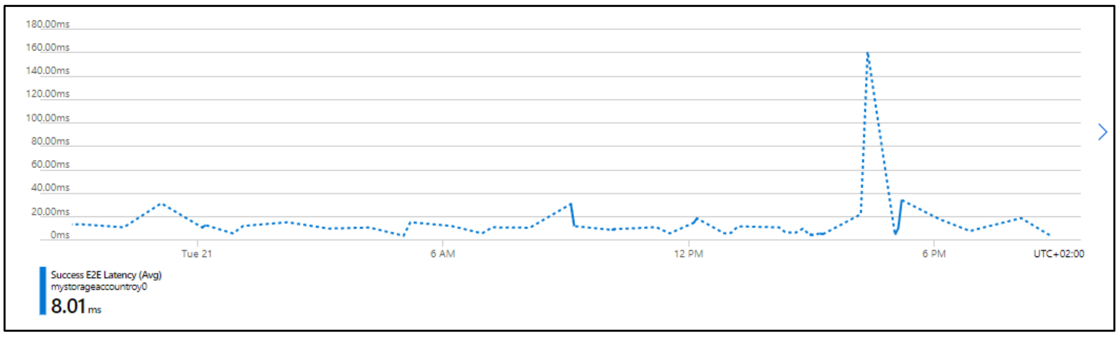


6.2.5 Availability

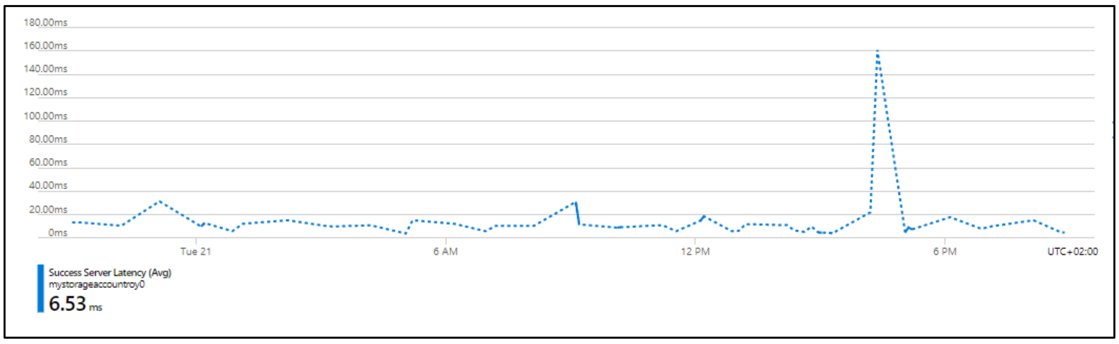


6.2.6 Latency

6.2.6.1 E2E Latency

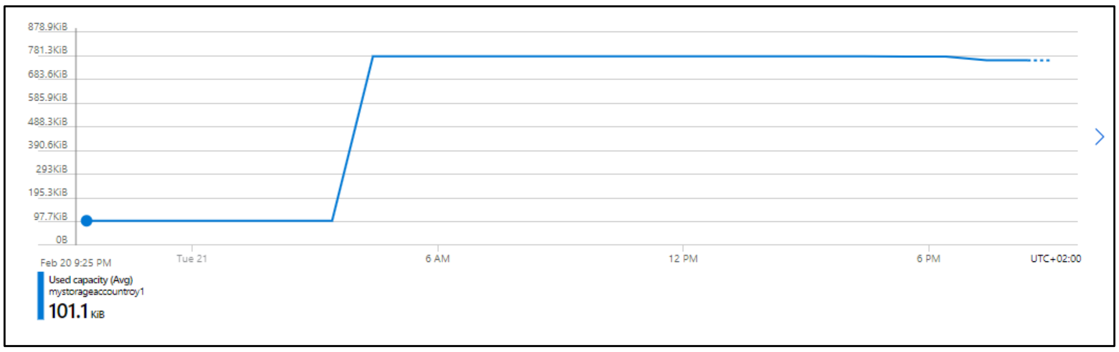


6.2.6.2 Success Server Latency

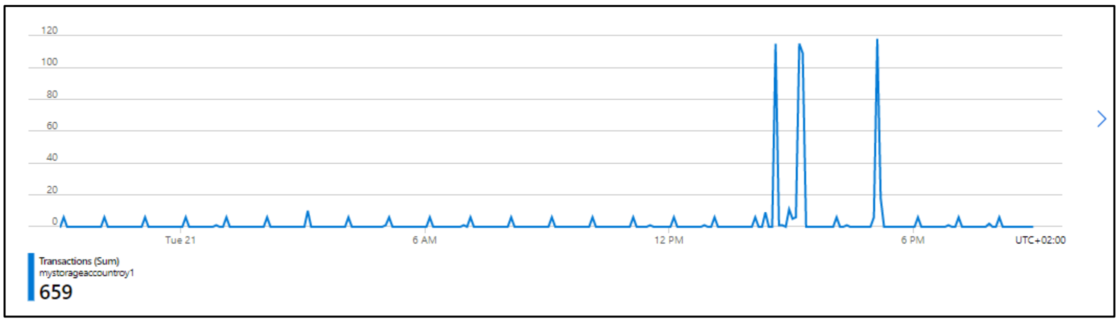


6.3 Destination Storage Account Metrics

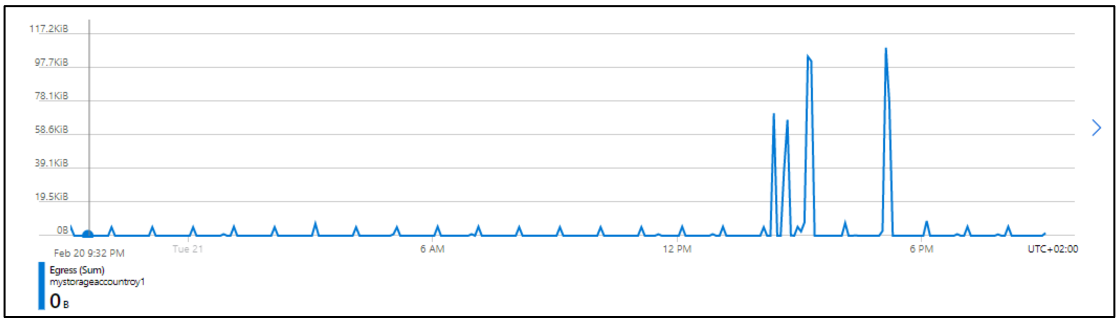
6.3.1 Used capacity



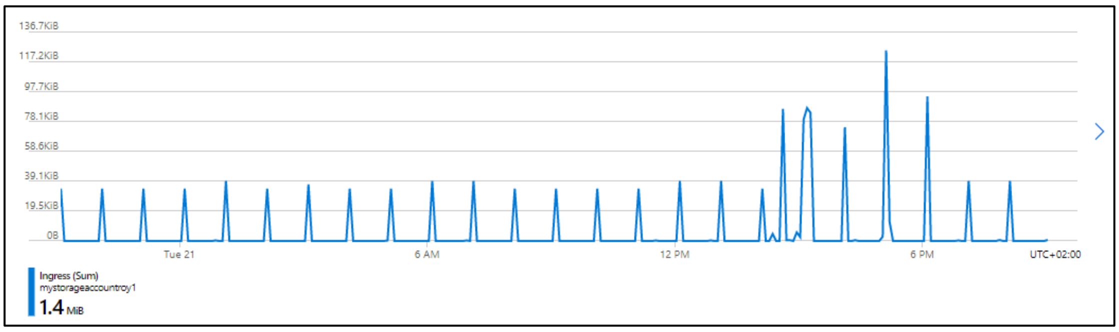
6.3.2 Transactions



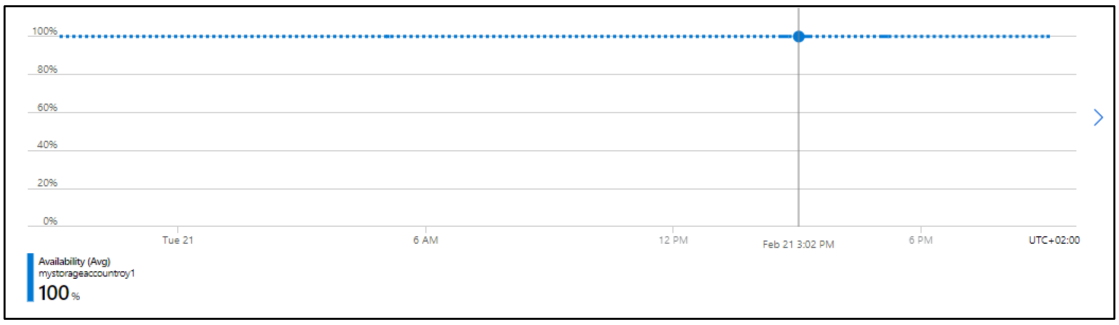
6.3.3 Egress



6.3.4 Ingress

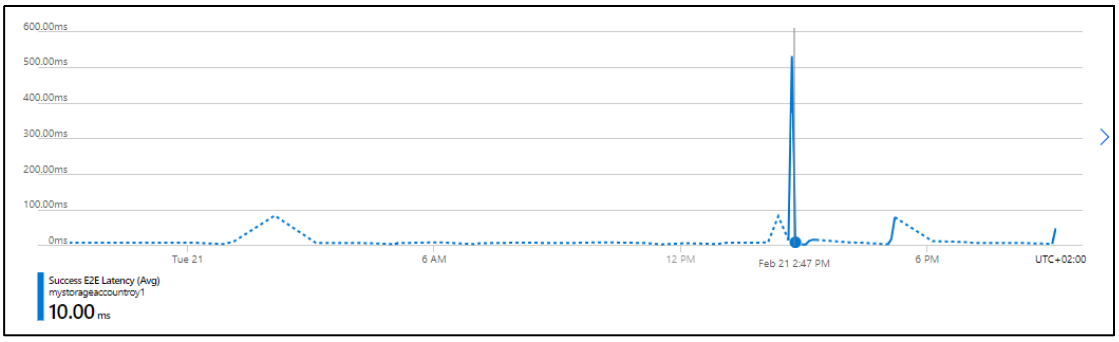


6.3.5 Availability

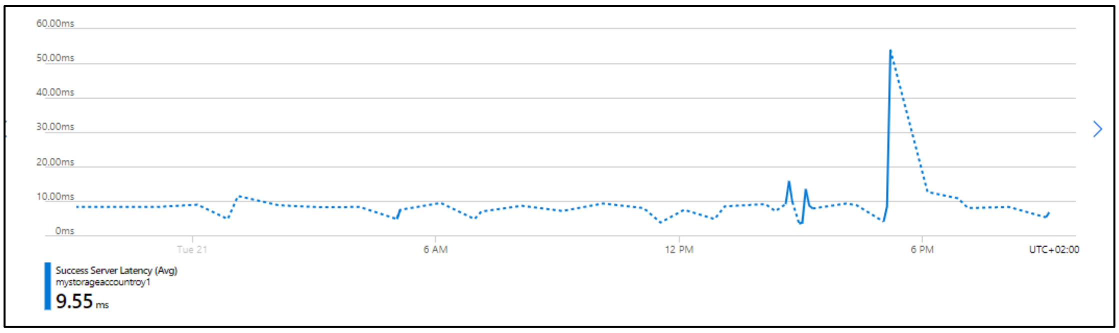


6.3.6 Latency

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.