**IT491** – Cloud Computing **Date** – April 15, 2023

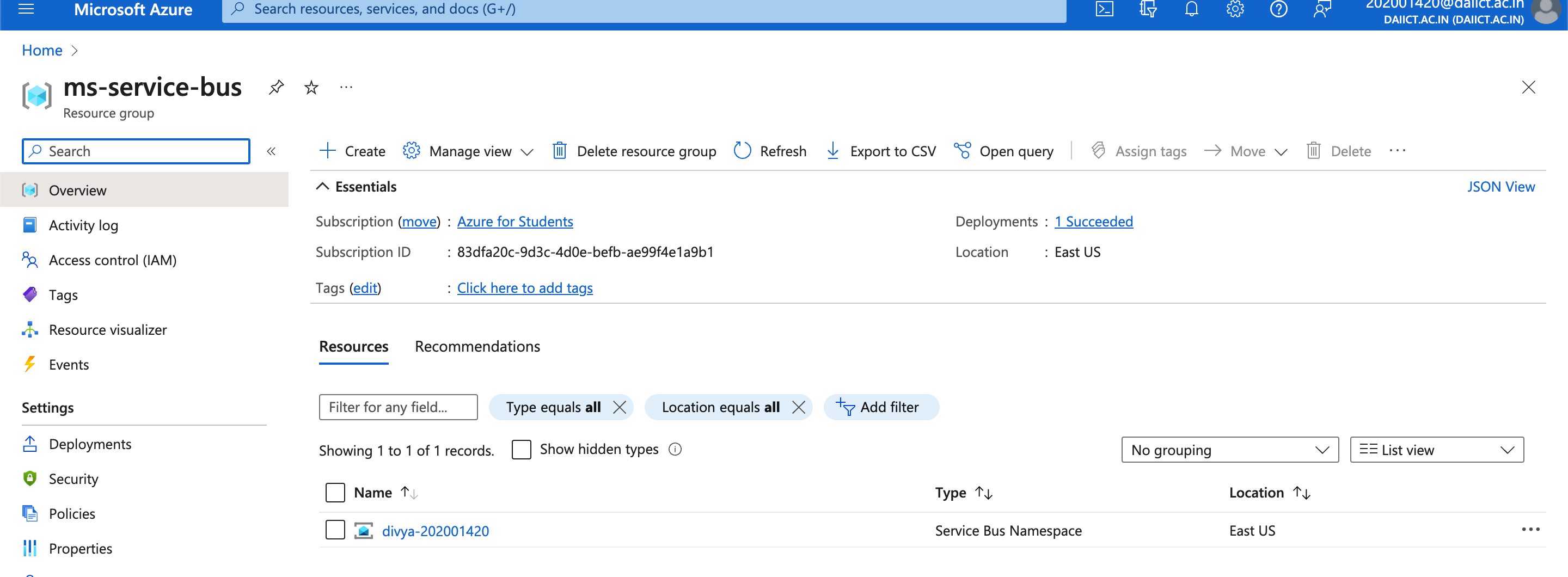
**Lab Report - 6**

**Name** – Divya Kirtikumar Patel

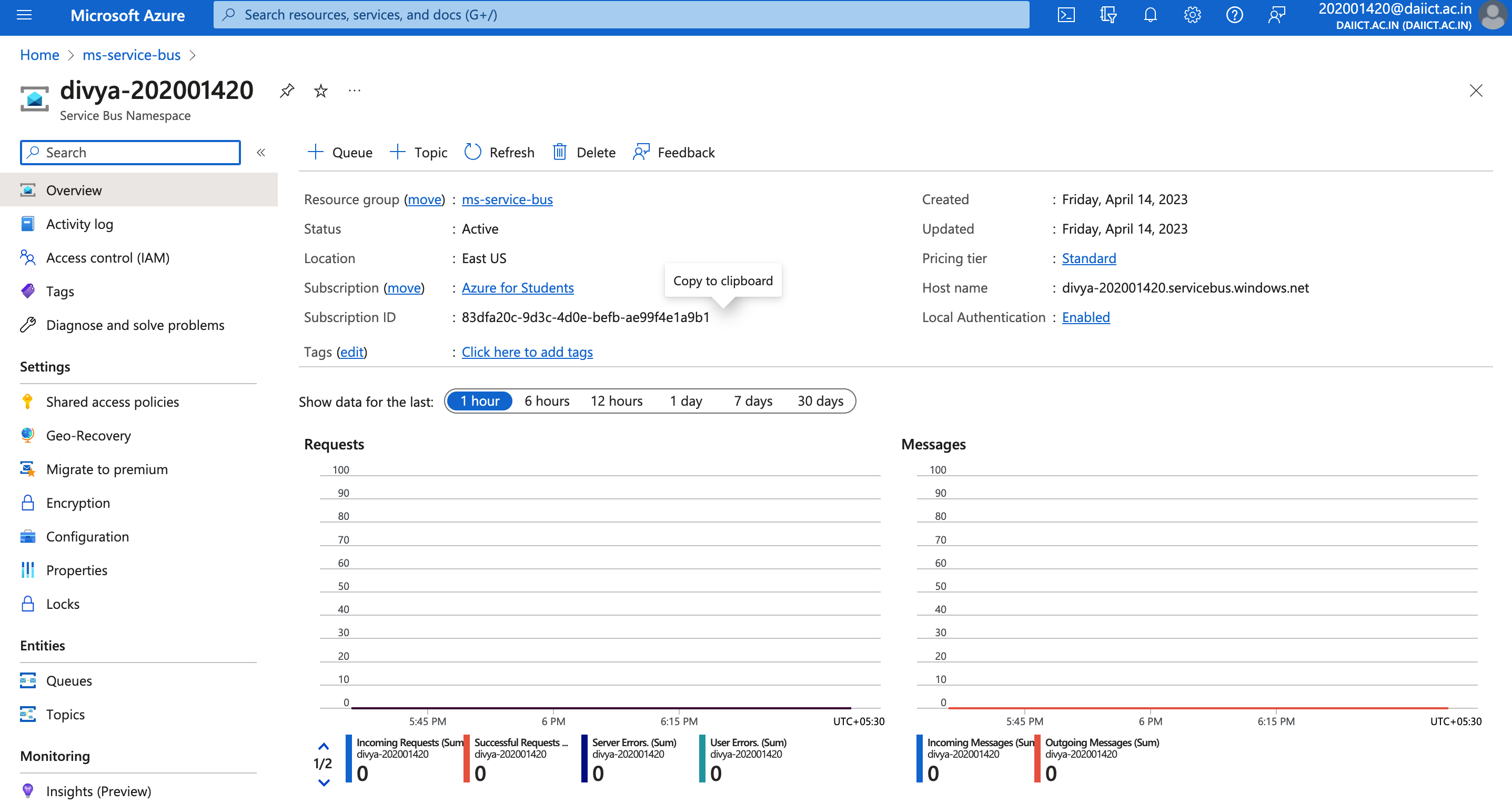
**Student ID** - **202001420**

# 1. Send messages to an Azure Service Bus topic and receive messages from subscriptions to the topic (Python)

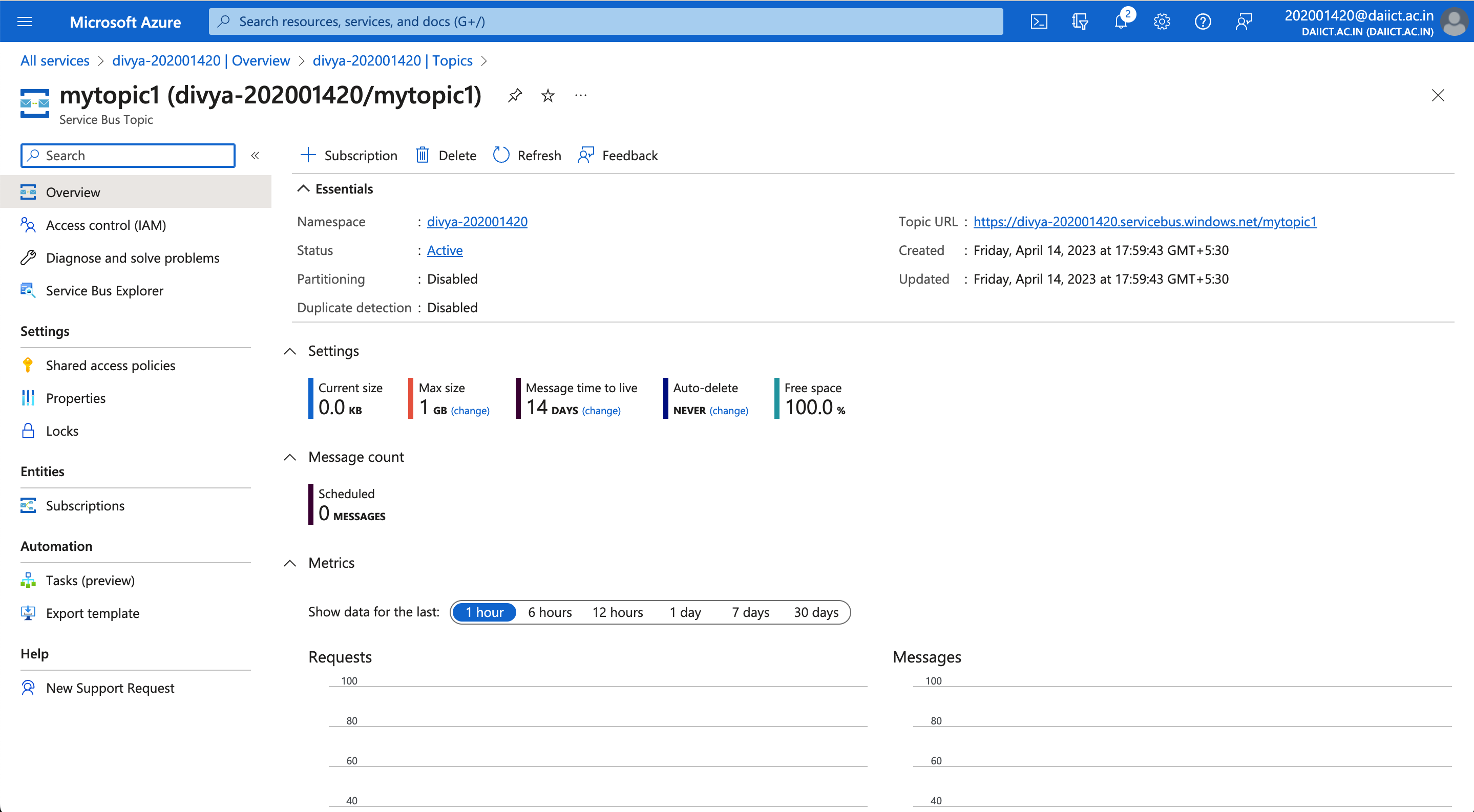
1. Create service Bus



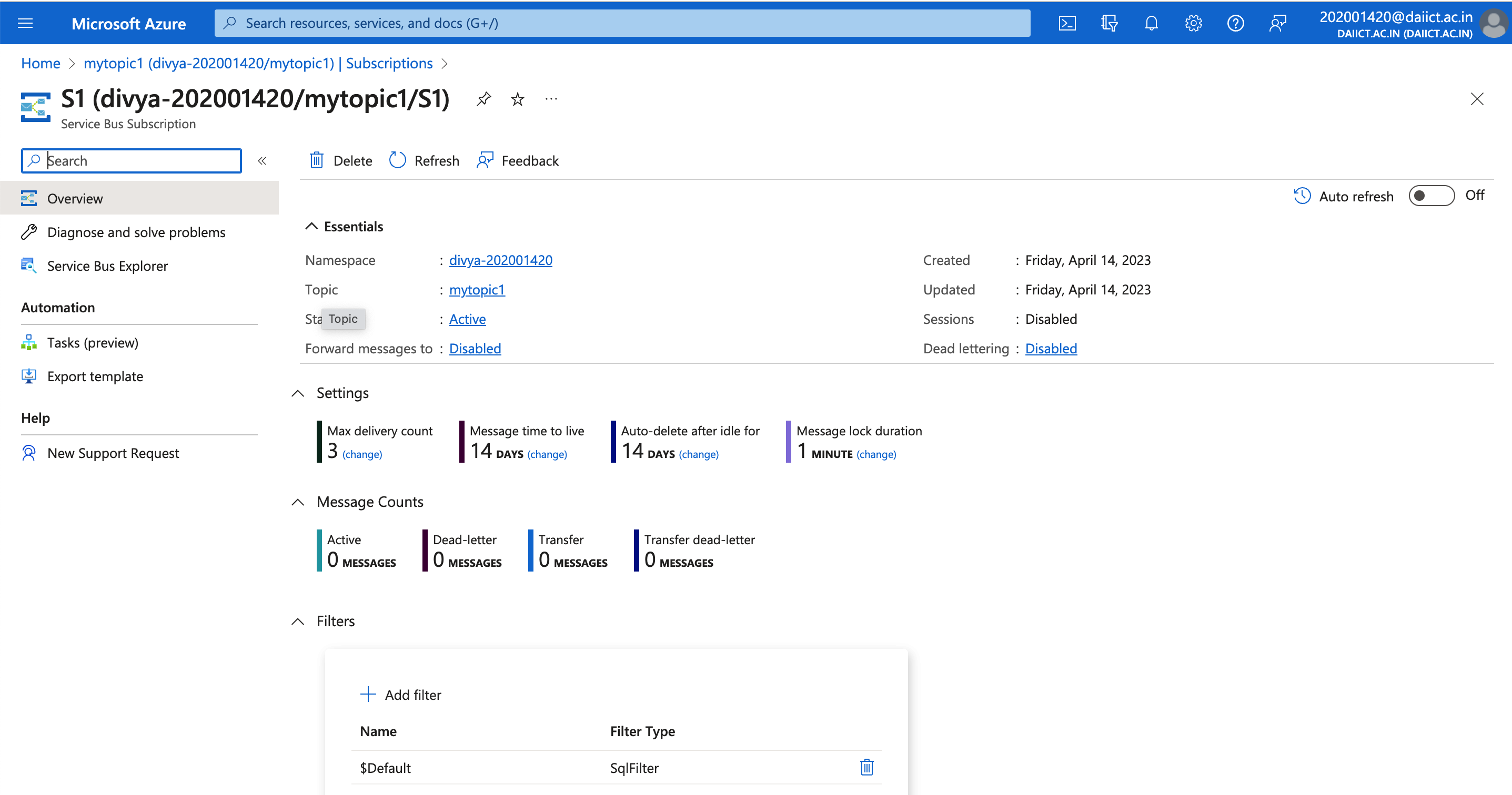
1. Service Bus Dashboard



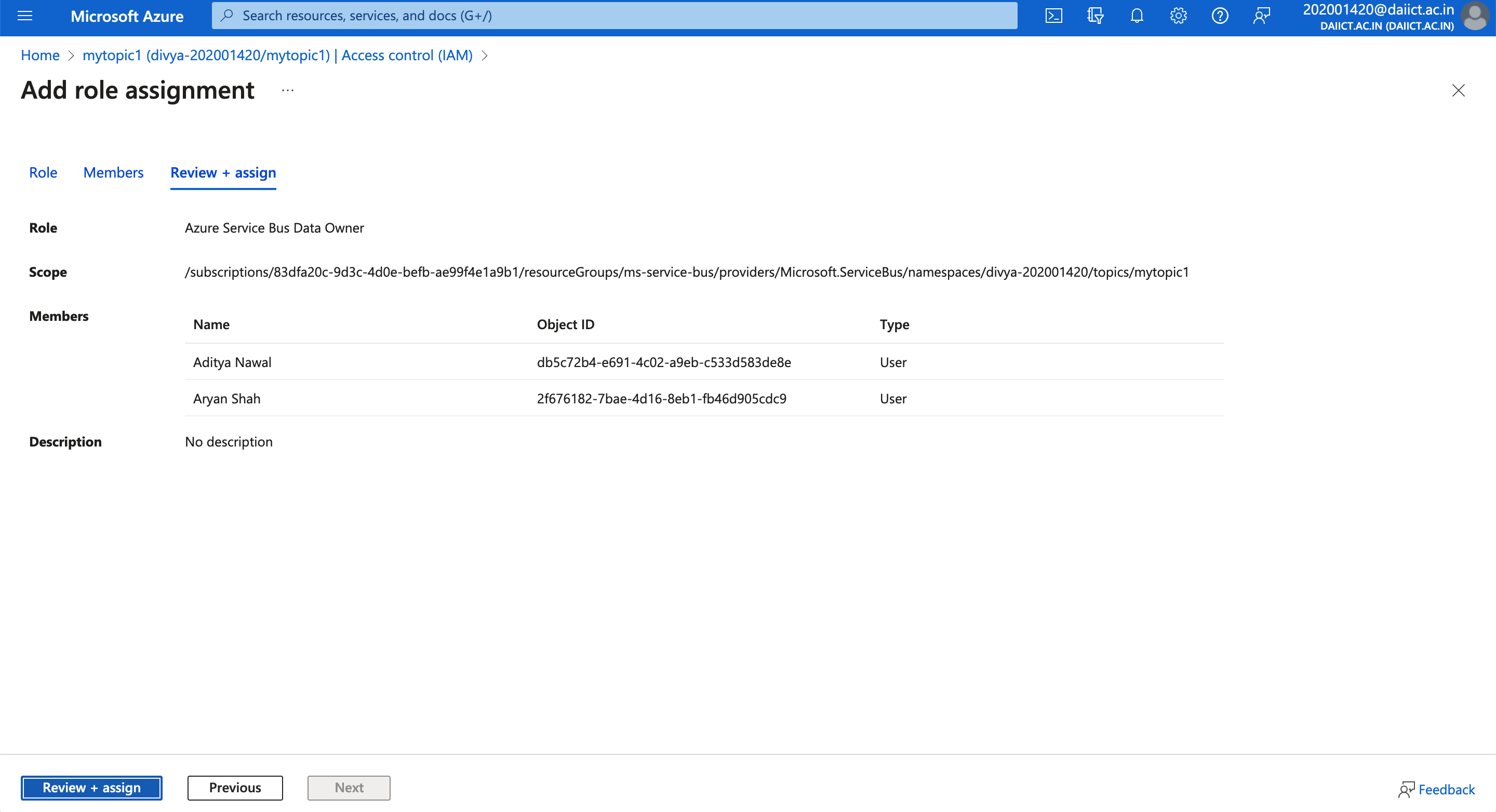
1. Create New Topic in Service Bus



1. Create new subscription in topic



1. Add Role Assignment



**Send.py**

import asyncio

from azure.servicebus.aio import ServiceBusClient

from azure.servicebus import ServiceBusMessage

NAMESPACE\_CONNECTION\_STR = "Endpoint=sb://divya-202001420.servicebus.windows.net/;SharedAccessKeyName=newpolicy;SharedAccessKey=6Q1uwO/JyLevdQ7vF2z7jBYk14c0EspLy+ASbF9dlSQ=;EntityPath=mytopic1"

TOPIC\_NAME = "mytopic1"

async def send\_single\_message(sender):

# Create a Service Bus message

message = ServiceBusMessage("Single Message")

# send the message to the topic

await sender.send\_messages(message)

print("Sent a single message")

async def send\_a\_list\_of\_messages(sender):

# Create a list of messages

messages = [ServiceBusMessage("Message in list") for \_ in range(5)]

# send the list of messages to the topic

await sender.send\_messages(messages)

print("Sent a list of 5 messages")

async def send\_batch\_message(sender):

# Create a batch of messages

async with sender:

batch\_message = await sender.create\_message\_batch()

for \_ in range(10):

try:

# Add a message to the batch

batch\_message.add\_message(ServiceBusMessage(

"Message inside a ServiceBusMessageBatch"))

except ValueError:

# ServiceBusMessageBatch object reaches max\_size.

# New ServiceBusMessageBatch object can be created here to send more data.

break

# Send the batch of messages to the topic

await sender.send\_messages(batch\_message)

print("Sent a batch of 10 messages")

async def run():

# create a Service Bus client using the connection string

async with ServiceBusClient.from\_connection\_string(

conn\_str=NAMESPACE\_CONNECTION\_STR,

logging\_enable=True) as servicebus\_client:

# Get a Topic Sender object to send messages to the topic

sender = servicebus\_client.get\_topic\_sender(topic\_name=TOPIC\_NAME)

async with sender:

# Send one message

await send\_single\_message(sender)

# Send a list of messages

await send\_a\_list\_of\_messages(sender)

# Send a batch of messages

await send\_batch\_message(sender)

asyncio.run(run())

print("Done sending messages")

print("-----------------------")

**recv.py**

import asyncio

from azure.servicebus.aio import ServiceBusClient

NAMESPACE\_CONNECTION\_STR = "Endpoint=sb://divya-202001420.servicebus.windows.net/;SharedAccessKeyName=newpolicy;SharedAccessKey=6Q1uwO/JyLevdQ7vF2z7jBYk14c0EspLy+ASbF9dlSQ=;EntityPath=mytopic1"

SUBSCRIPTION\_NAME = "S1"

TOPIC\_NAME = "mytopic1"

async def run():

# create a Service Bus client using the connection string

async with ServiceBusClient.from\_connection\_string(

conn\_str=NAMESPACE\_CONNECTION\_STR,

logging\_enable=True) as servicebus\_client:

async with servicebus\_client:

# get the Subscription Receiver object for the subscription

receiver = servicebus\_client.get\_subscription\_receiver( topic\_name=TOPIC\_NAME, subscription\_name= SUBSCRIPTION\_NAME, max\_wait\_time=5)

async with receiver:

received\_msgs = await receiver.receive\_messages(max\_wait\_time=5, max\_message\_count=20)

for msg in received\_msgs:

print("Received: " + str(msg))

# complete the message so that the message is removed from the subscription

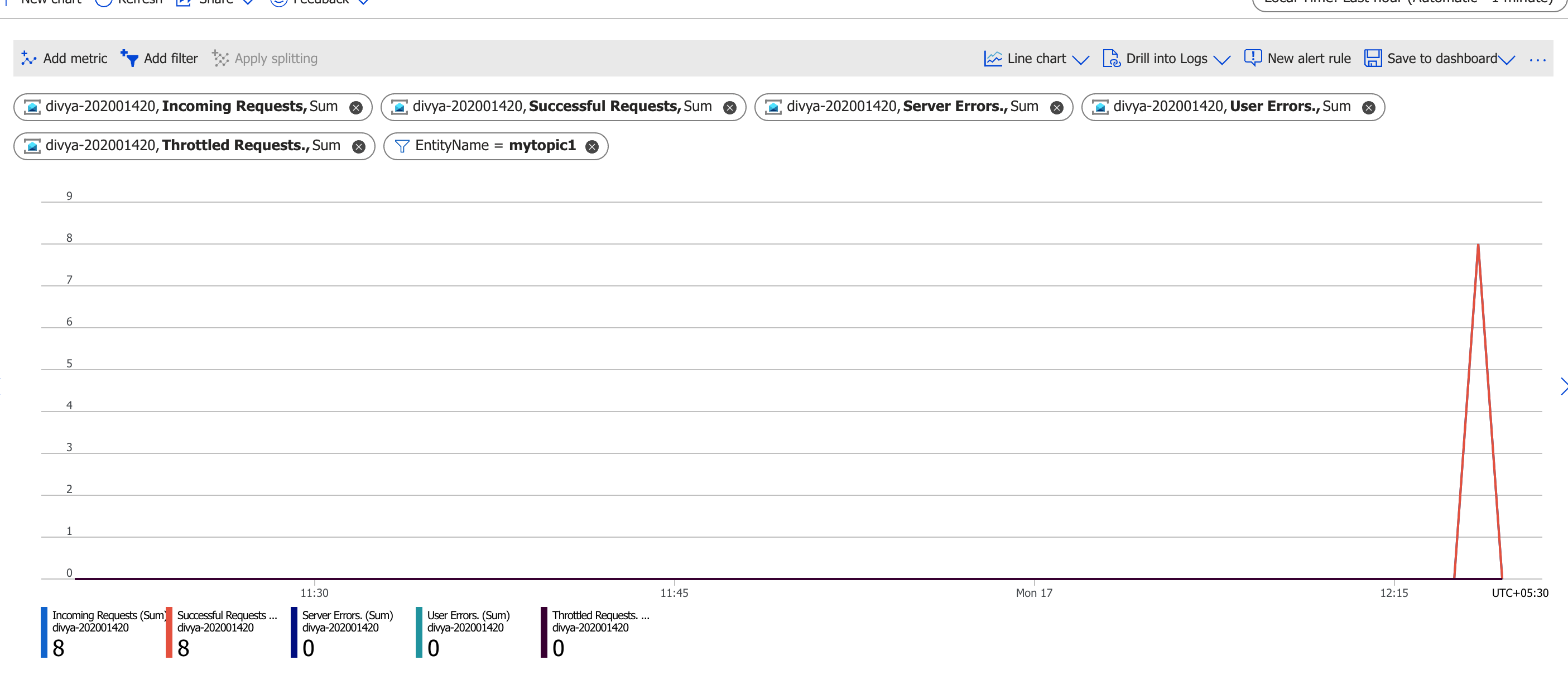
receiver.complete\_message(msg)

asyncio.run(run())

**Running Scripts**

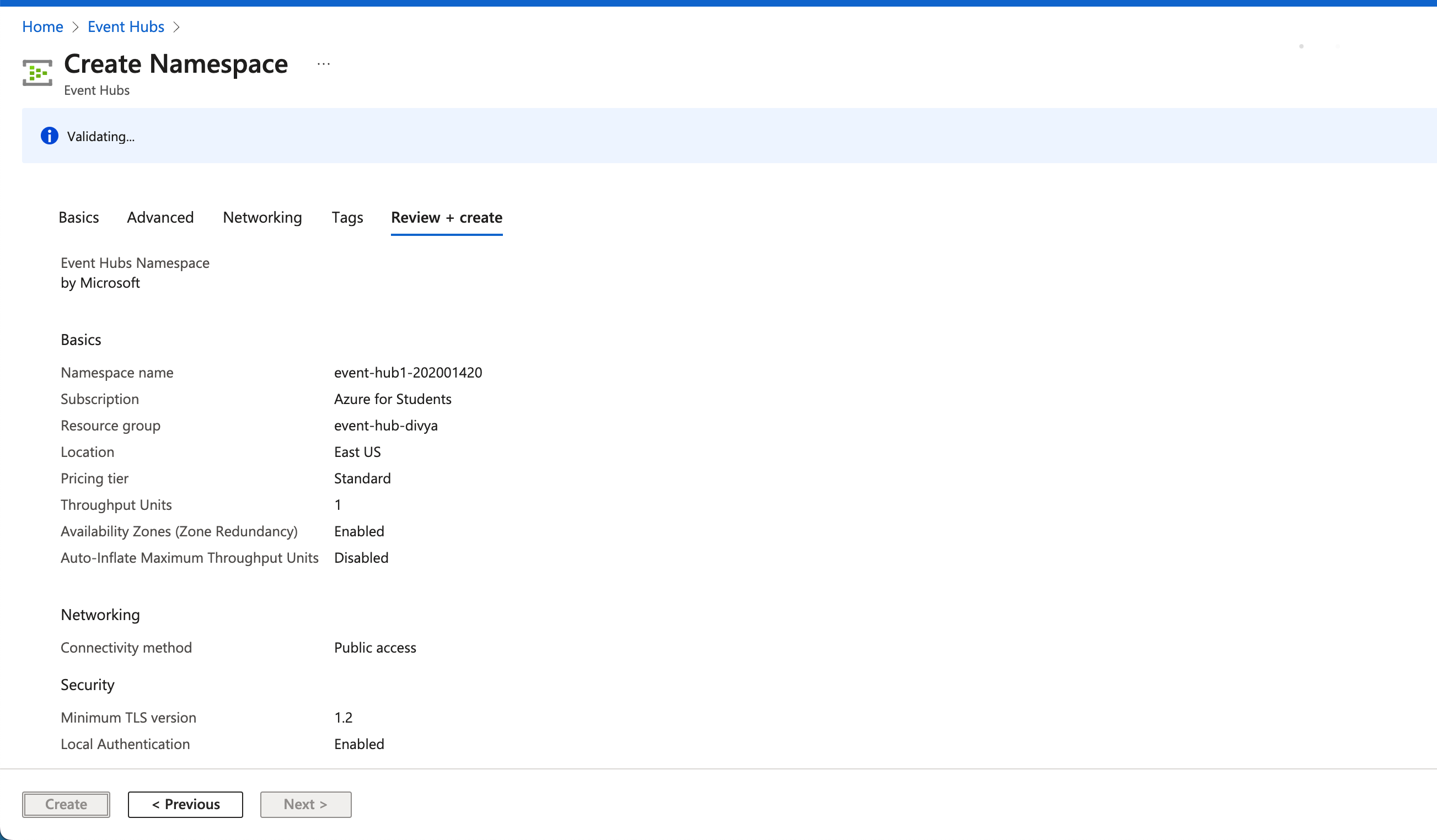


**Dashboard of Topic**

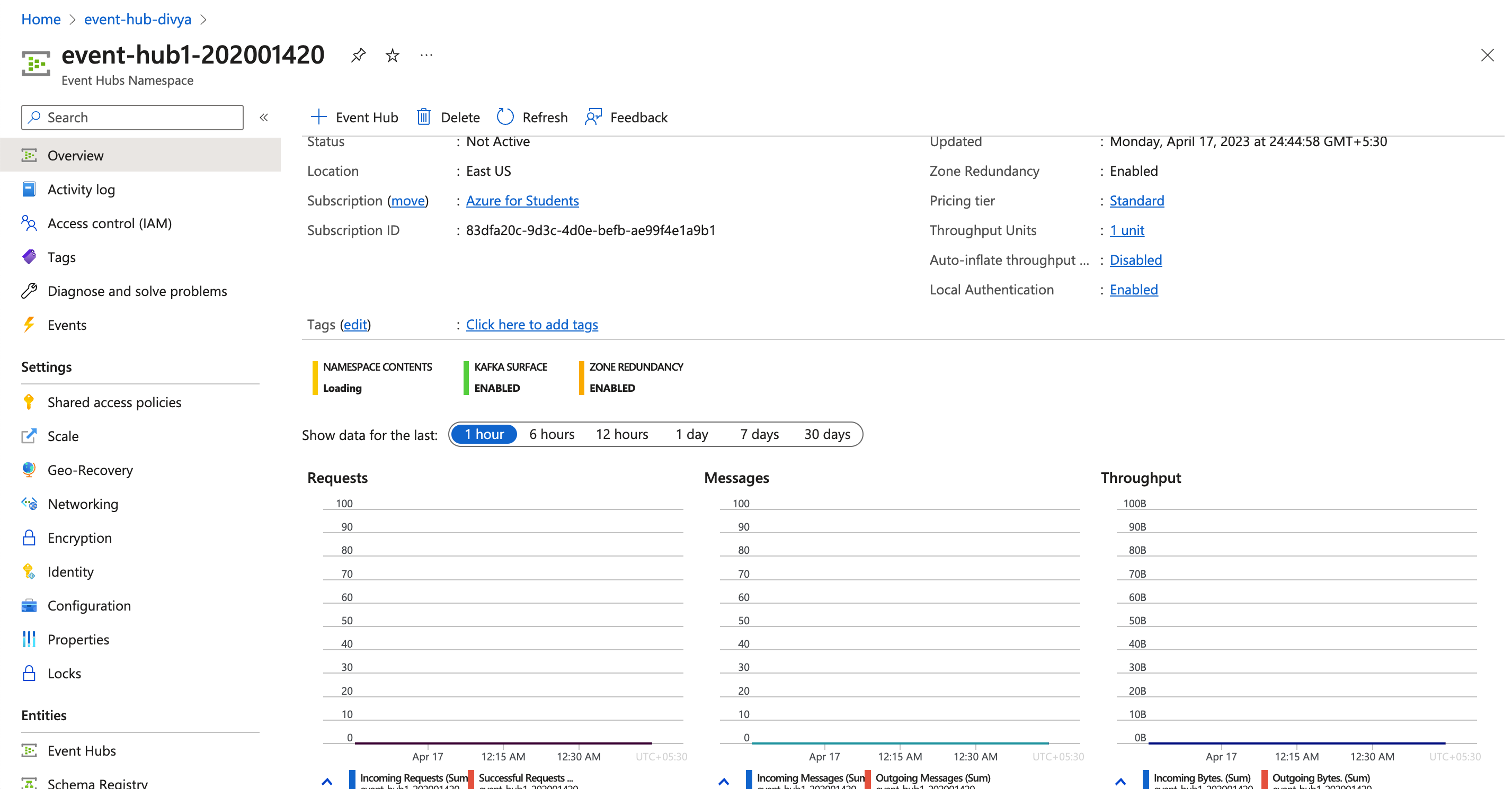


# 2. Send events to or receive events from event hubs by using Python

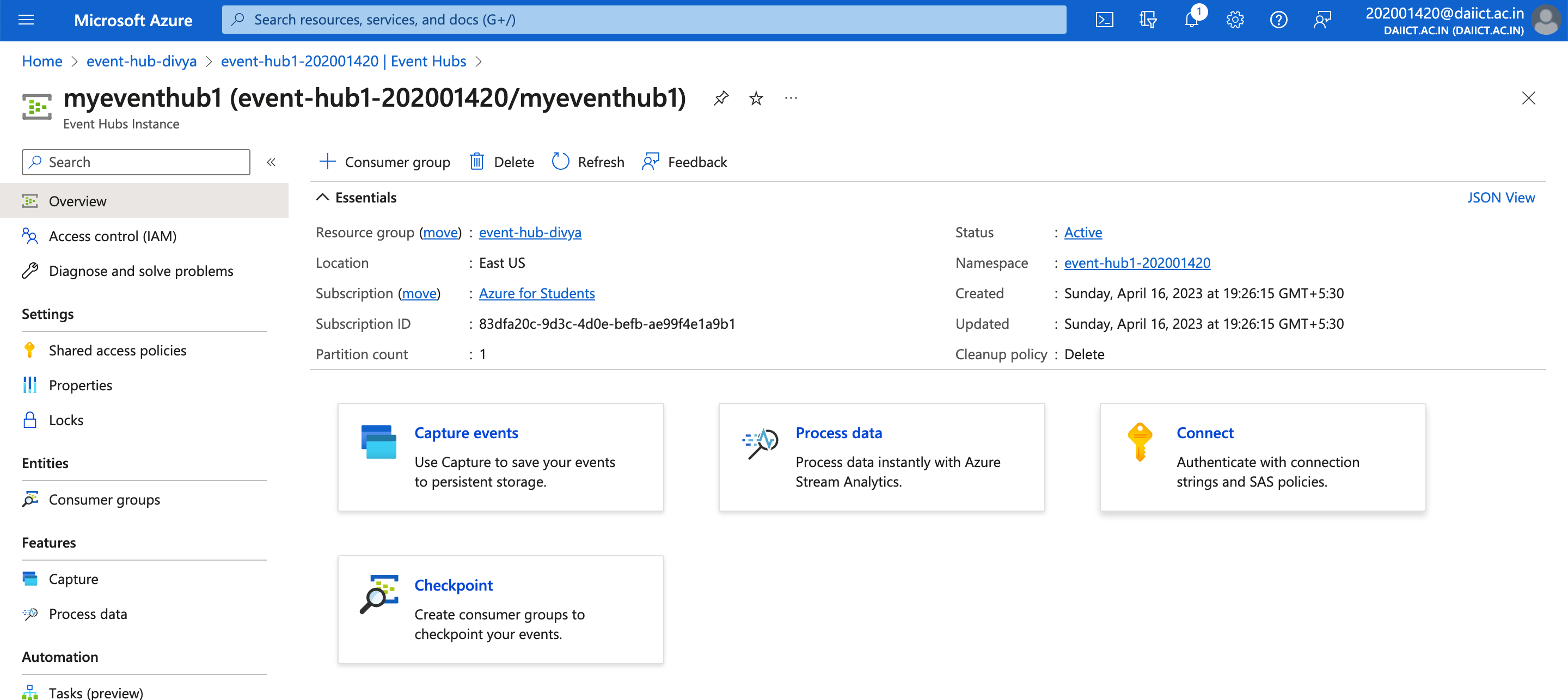
1. Create event-hub namespace



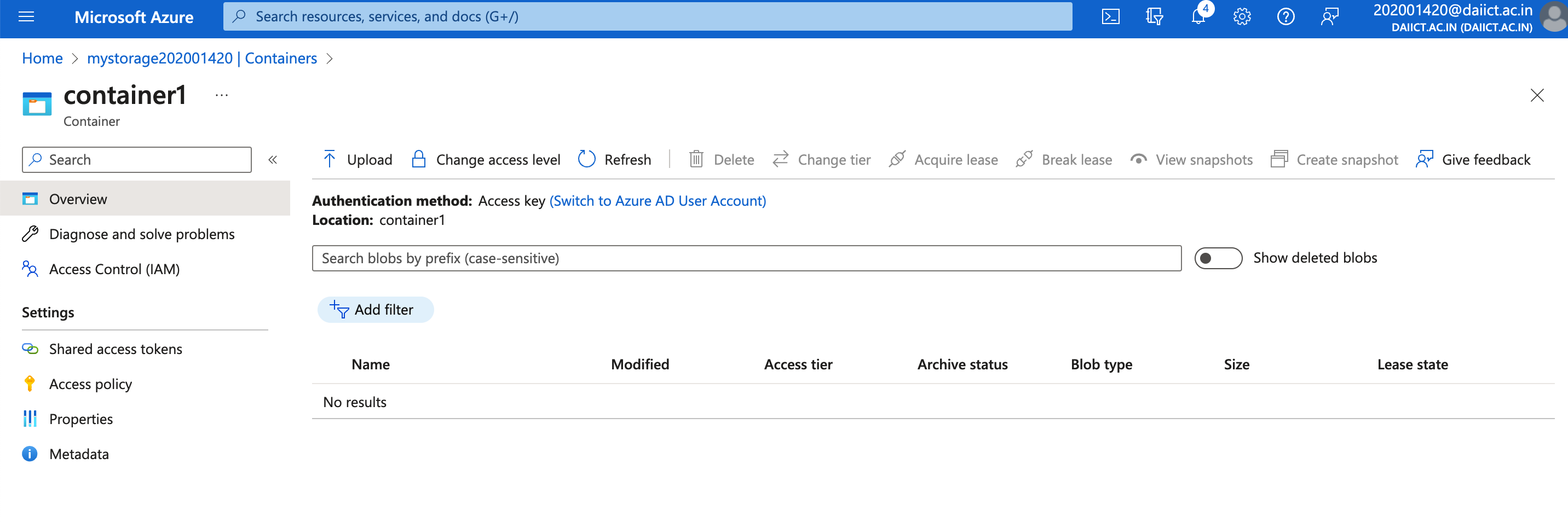
1. Create Event-hub



1. Event-hub details



1. Container Details



**Run.py**

import asyncio

from azure.eventhub import EventData

from azure.eventhub.aio import EventHubProducerClient

from azure.identity import DefaultAzureCredential

EVENT\_HUB\_FULLY\_QUALIFIED\_NAMESPACE = "event-hub1-202001420"

EVENT\_HUB\_NAME = "myeventhub1"

credential = DefaultAzureCredential()

async def run():

# Create a producer client to send messages to the event hub.

# Specify a credential that has correct role assigned to access

# event hubs namespace and the event hub name.

producer = EventHubProducerClient(

fully\_qualified\_namespace=EVENT\_HUB\_FULLY\_QUALIFIED\_NAMESPACE,

eventhub\_name=EVENT\_HUB\_NAME,

credential=credential,

)

async with producer:

# Create a batch.

event\_data\_batch = await producer.create\_batch()

# Add events to the batch.

event\_data\_batch.add(EventData("First event "))

event\_data\_batch.add(EventData("Second event"))

event\_data\_batch.add(EventData("Third event"))

# Send the batch of events to the event hub.

await producer.send\_batch(event\_data\_batch)

# Close credential when no longer needed.

await credential.close()

asyncio.run(run())

**Recv.py**

from azure.identity.aio import DefaultAzureCredential

from azure.eventhub.extensions.checkpointstoreblobaio import (

BlobCheckpointStore,

)

from azure.eventhub.aio import EventHubConsumerClient

import asyncio

from azure.eventhub import EventData

from azure.eventhub.aio import EventHubProducerClient

from azure.identity import DefaultAzureCredential

EVENT\_HUB\_FULLY\_QUALIFIED\_NAMESPACE = "event-hub1-202001420"

EVENT\_HUB\_NAME = "myeventhub1"

BLOB\_STORAGE\_ACCOUNT\_URL = "DefaultEndpointsProtocol=https;AccountName=mystorage202001420;AccountKey=SSu5MB7UY1C7fClikzk4eT4zoGnuIqrSmMNRL2simnrOp6UdfWZl9bZm3EcZJJqh8VebJnYuFZ6G+AStD912Ug==;EndpointSuffix=core.windows.net"

BLOB\_CONTAINER\_NAME = "container1"

credential = DefaultAzureCredential()

async def on\_event(partition\_context, event):

# Print the event data.

print(

'Received the event: "{}" from the partition with ID: "{}"'.format(

event.body\_as\_str(encoding="UTF-8"), partition\_context.partition\_id

)

)

# Update the checkpoint so that the program doesn't read the events

# that it has already read when you run it next time.

await partition\_context.update\_checkpoint(event)

async def main():

# Create an Azure blob checkpoint store to store the checkpoints.

checkpoint\_store = BlobCheckpointStore(

blob\_account\_url=BLOB\_STORAGE\_ACCOUNT\_URL,

container\_name=BLOB\_CONTAINER\_NAME,

credential=credential,

)

# Create a consumer client for the event hub.

client = EventHubConsumerClient(

fully\_qualified\_namespace=EVENT\_HUB\_FULLY\_QUALIFIED\_NAMESPACE,

eventhub\_name=EVENT\_HUB\_NAME,

consumer\_group="$Default",

checkpoint\_store=checkpoint\_store,

credential=credential,

)

async with client:

# Call the receive method. Read from the beginning of the partition

# (starting\_position: "-1")

await client.receive(on\_event=on\_event, starting\_position="-1")

# Close credential when no longer needed.

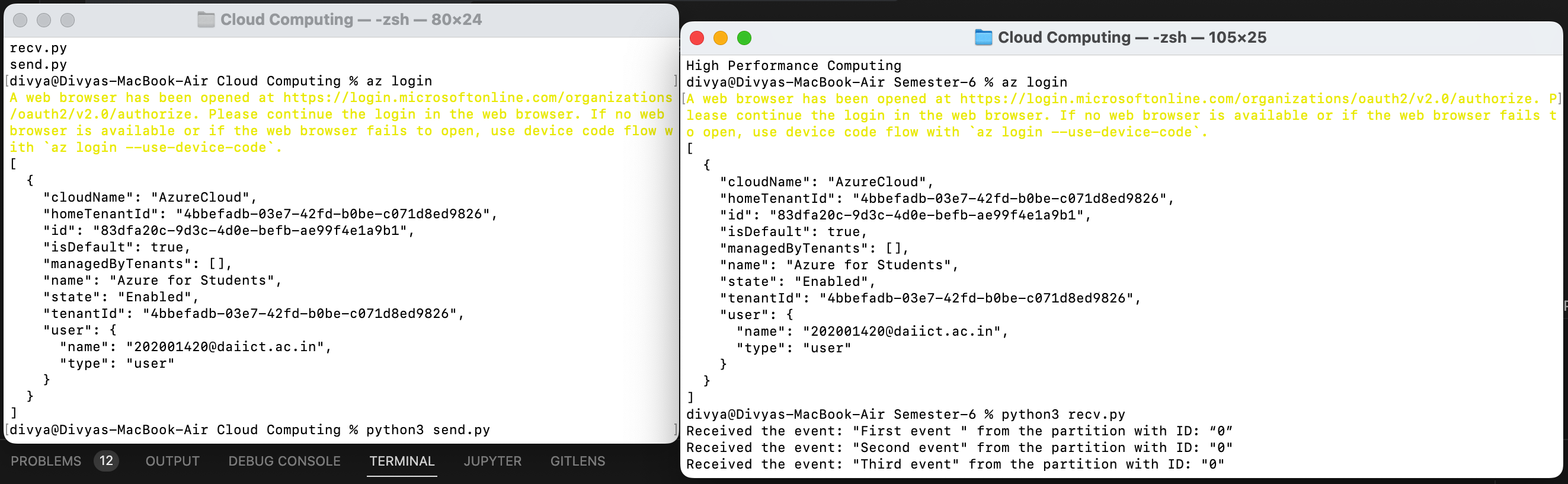
await credential.close()

if \_\_name\_\_ == "\_\_main\_\_":

# Run the main method.

asyncio.run(main())

**Running send.py**



**Running recv.py**

